

Transport Area Working Group  
Internet-Draft  
Intended status: Experimental  
Expires: January 9, 2020

M. Amend  
Deutsche Telekom  
A. Brunstrom  
A. Kassler  
Karlstad University  
V. Rakocevic  
City University of London  
July 08, 2019

Lossless and overhead free DCCP - UDP header conversion (U-DCCP)  
draft-amend-tsvwg-dccp-udp-header-conversion-01

## Abstract

The Datagram Congestion Control Protocol (DCCP) is a transport-layer protocol that provides upper layers with the ability to use non-reliable congestion-controlled flows. DCCP is not widely deployed in the Internet, and the reason for that can be defined as a typical example of a chicken-egg problem. Even if an application developer decided to use DCCP, the middle-boxes like firewalls and NATs would prevent DCCP end-to-end since they lack support for DCCP. Moreover, as long as the protocol penetration of DCCP does not increase, the middle-boxes will not handle DCCP properly. To overcome this challenge, NAT/NATP traversal and UDP encapsulation for DCCP is already defined. However, the former requires special middle-box support and the latter introduces overhead. The recent proposal of a multipath extension for DCCP further underlines the challenge of efficient middle-box passing as its main goal is to be applied over the Internet, traversing numerous uncontrolled middle-boxes. This document introduces a new solution which disguises DCCP during transmission as UDP without requiring middle-box modification or introducing any overhead.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 9, 2020.

#### Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

#### Table of Contents

1. Introduction . . . . .	2
2. Terminology . . . . .	3
3. U-DCCP . . . . .	3
3.1. Overview . . . . .	3
3.2. The DCCP Generic header . . . . .	4
3.3. UDP header . . . . .	5
3.4. U-DCCP conversion considerations . . . . .	6
3.5. U-DCCP header . . . . .	6
3.6. Implementation . . . . .	7
3.7. Pseudo-code DCCP to U-DCCP conversion . . . . .	7
3.8. Pseudo-code U-DCCP to DCCP restoration . . . . .	8
3.9. U-DCCP negotiation (required???) . . . . .	9
4. Security Considerations . . . . .	9
5. IANA Considerations . . . . .	9
6. Notes . . . . .	9
7. Acknowledgments . . . . .	9
8. Informative References . . . . .	9
Authors' Addresses . . . . .	10

#### 1. Introduction

The Datagram Congestion Control Protocol (DCCP) [RFC4340] is a transport-layer protocol that provides upper layers with the ability to use non-reliable congestion-controlled flows. The current specification for DCCP [RFC4340] specifies a direct native encapsulation in IPv4 or IPv6 packets.

DCCP support has been specified for devices that use Network Address Translation (NAT) or Network Address and Port Translation (NAPT)

[RFC5597]. However, there is a significant installed base of NAT/NAPT devices that do not support [RFC5597]. An UDP encapsulation for DCCP [RFC6773] circumvents such limitations and makes DCCP compatible with any UDP [RFC0768] compliant device that supports [RFC4787] but does not support [RFC5597]. For convenience, the standard encapsulation for DCCP [RFC4340] (including [RFC5596] and [RFC5597] as required) is referred to as DCCP-STD, whereas the UDP encapsulation for DCCP [RFC6773] is referred to as DCCP-UDP.

It can be stated that DCCP-STD and DCCP-UDP are techniques which increase the success rate of DCCP transmissions significantly. However, DCCP-STD fails on devices that block DCCP for any reasons. On the other hand, DCCP-UDP uses the well-accepted UDP to let devices assume they are handling the UDP protocol, but at the cost of a reduced goodput/throughput ratio.

To compensate for the inefficiency of DCCP-STD (device blocking) and DCCP-UDP (overhead), this document proposes a beneficial modification scheme relying on UDP (like DCCP-UDP), but with no overhead. This goal is reached by re-arranging DCCP's extended header to make it look like UDP, without losing critical information. This solution is referred to as U-DCCP.

U-DCCP is limited to DCCP's extended header, requiring X is set to 1. Otherwise U-DCCP relies on the NAT/NATP functionalities specified for UDP in [RFC4787], [RFC6888] and [RFC7857].

## 2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

## 3. U-DCCP

### 3.1. Overview

The basic approach of U-DCCP is to modify the extended header of a DCCP packet so that it appears like UDP [RFC0768]. In particular, this takes place without losing any header information, but requires a U-DCCP termination before the packet is delivered to the DCCP end system. This method does not change the 4-tuple of IP and port addressing, however it changes the protocol carried over IP from DCCP to UDP. As a consequence, the length of the packet remains unchanged and behaves like DCCP-STD. The solution is not a tunneling approach. It requires that the same port used by DCCP can be used by UDP.

The method is designed to support use when the IP addresses are modified by a device that implements NAT/NAPT. A NAT translates the IP addresses, which impacts the transport-layer checksum. A NAPT device may also translate the port values (usually the source port). In both cases, the outer transport header that includes these values would need to be updated by the NAT/NAPT.

U-DCCP supports IPv4 and IPv6.

The basic format of a U-DCCP packet is:

+-----+		
	IP Header (IPv4 or IPv6)	Variable length
+-----+		
	UDP like arranged DCCP ext. Header	8 bytes \
+-----+		
	Rest of rearranged DCCP ext. Header	8 bytes /
+-----+		
	Additional (type-specific) Fields	Variable length (could be 0)
+-----+		
	DCCP Options	Variable length (could be 0)
+-----+		
	Application Data Area	Variable length (could be 0)
+-----+		

Figure 1: Format of U-DCCP packet

The U-DCCP header is described in Section 3.4 after introducing the traditional DCCP header in Section 3.1 and its target appearance of a UDP header in Section 3.2. Section 3.3 discusses considerations for building the U-DCCP header upfront.

### 3.2. The DCCP Generic header

The DCCP Generic Header [RFC4340] takes two forms: one with long sequence numbers (48 bits) and the other with short sequence numbers (24 bits). The short one is not part of U-DCCP's modification.

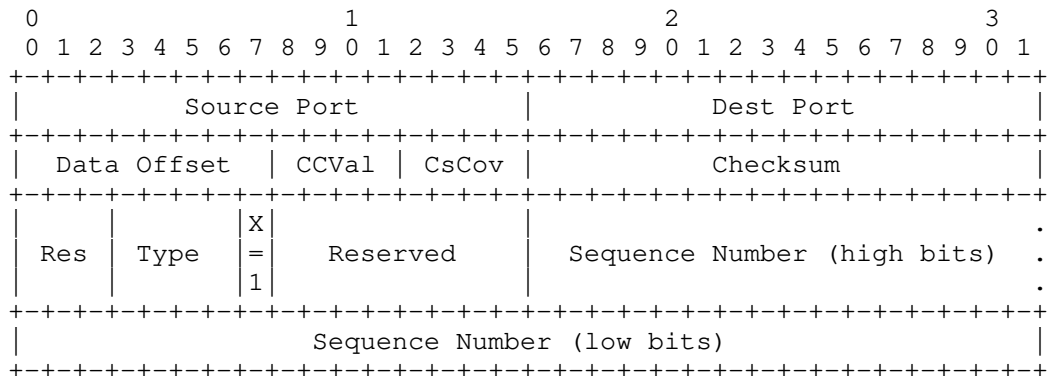


Figure 2: The extended DCCP Header with Long Sequence Numbers  
[RFC4340]

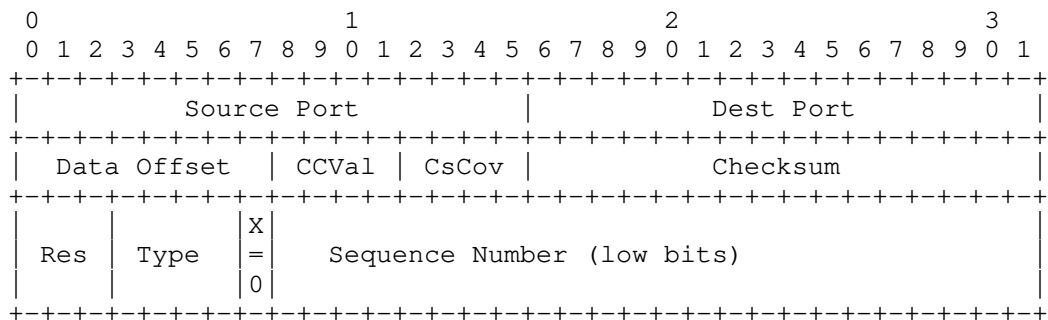


Figure 3: The short DCCP Header with Short Sequence Numbers [RFC4340]

All generic header fields have the meaning specified in [RFC4340], updated by [RFC5596].

### 3.3. UDP header

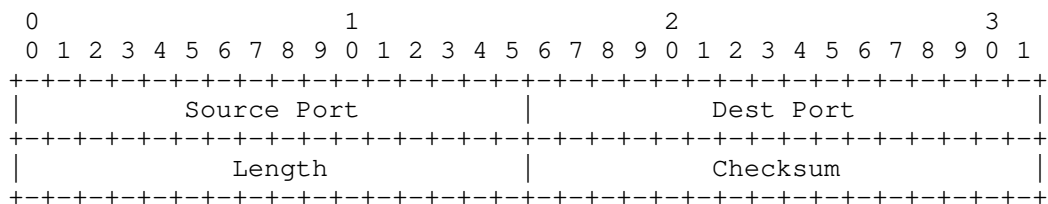


Figure 4: The UDP Header [RFC768]

All header fields have the meaning specified in [RFC0768].

### 3.4. U-DCCP conversion considerations

The U-DCCP header has the goal to merge the information of DCCP's extended header (Section 3.1) and imitates in the first 64 bits the UDP header (Section 3.2). Information required to restore a DCCP header from any conversion, which must not be lost, includes: source and destination port, Data Offset, CCVal, CsCov, Checksum, Type, X and the Sequence Number.

Compared with the UDP header, the DCCP extended header shows similarities in source and destination port and checksum. The length field of UDP (bits 33-48) is not part of the DCCP header and contains in case of DCCP the fields Data Offset, CCVal and CsCov.

For the goal of imitating UDP, the checksum must cover the whole datagram, which renders any limitation by CsCov useless. The checksum itself is required to re-calculate after conversion anyway.

If the conversion is limited to DCCP'S extended header only, X is always "1".

Thus, Data Offset, CCVal, Type and Sequence Number must be re-arranged in a way that the Length field of UDP can be applied.

### 3.5. U-DCCP header

The considerations of Section 3.3 leads to the following header, denoted as U-DCCP header.

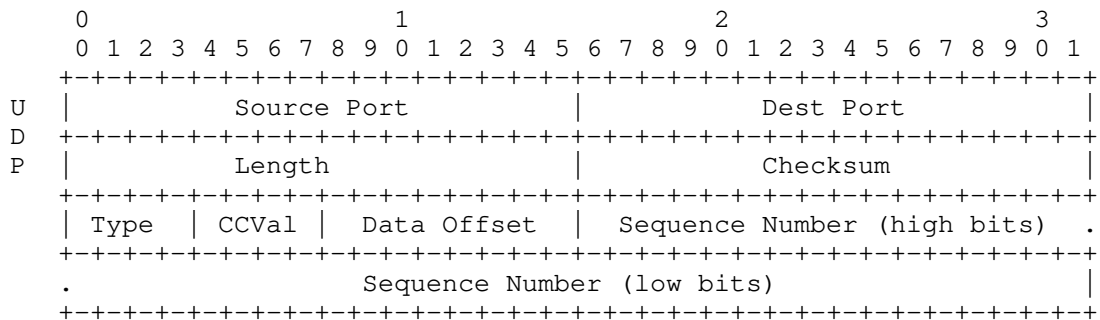


Figure 5: The U-DCCDP Header

The first 8 bytes of the U-DCCP header corresponds to [RFC0768] and the fields are interpreted as follows:

Source and Dest(ination) Ports: 16 bits each

These fields identify the UDP ports used by the source and destination (respectively) of the packet to listen for incoming UDP packets. The UDP port values identify the DCCP source and destination ports.

Length: 16 bits

This field is the length of the UDP datagram, including the UDP header and the payload (for U-DCCP, the payload comprises the payload of the original DCCP datagram and part of its header).

Checksum: 16 bits

This field is the Internet checksum of a network-layer pseudoheader and Length bytes of the UDP packet [RFC0768]. The UDP checksum MUST NOT be zero for a U-DCCP packet.

The remaining 8 bytes of the U-DCCP header contains:

Type, CCVal, Data Offset, Seq. Number: As specified in [RFC4340]

In case U-DCCP is applied, the IP layer must be instructed to carry an UDP datagram and its checksum must be re-calculated. For detailed information see Section 3.7.

### 3.6. Implementation

The process of applying U-DCCP is defined as follows:

DCCP generation -> U-DCCP conversion -> UDP transmission -> U-DCCP reception and restoration -> DCCP reception

The conversion can be integrated into DCCP endpoints directly or as an additional component on the way along the transmission route. Depending on the degree of integration, especially the process of checksum calculation and validation can be optimized. Section 3.7 and Section 3.8 provide a possible pseudo-code for the conversion without any optimized integration into the sender's network stack or into the receiver's network stack. The pseudo-code assumes explicit knowledge on which U-DCCP flows need conversion between the sender and the receiver.

### 3.7. Pseudo-code DCCP to U-DCCP conversion

A possible processing of an already generated DCCP datagram for U-DCCP conversion:

1. Receive DCCP datagram.

2. Check eligibility for conversion; otherwise bypass conversion.
3. Verify consistency, e.g. checksum; otherwise drop.
4. Shift Type and CCVal field to the ninth octet.
5. Shift Data Offset field to the tenth octet.
6. Place a length information at octet 5+6 corresponding to [RFC0768].
7. Modify the IP header's encapsulated protocol from DCCP to UDP.
8. Re-calculate IP header checksum.
9. Reset DCCP checksum field: octet 7+8 = 0.
10. Generate new checksum at octet 7+8 as described in [RFC0768].
11. Forward to destination based on the unmodified 4-tuple of IP-addresses and ports.

### 3.8. Pseudo-code U-DCCP to DCCP restoration

A possible processing of an already converted U-DCCP datagram for DCCP restoration:

1. Receive UDP datagram.
2. Check eligibility for restoration; otherwise bypass restoration
3. Validate UDP checksum; otherwise drop.
4. Restore Data Offset field according to [RFC4340].
5. Restore CCVal field according to [RFC4340].
6. Set CsCov field according to [RFC4340] to "0".
7. Restore Type field according to [RFC4340].
8. Set Reserved bits according to [RFC4340] to "0".
9. Set X according to [RFC4340] to "1".
10. Modify the IP header's encapsulated protocol from UDP to DCCP.
11. Re-calculate IP header checksum.



12. Reset DCCP checksum field: octet 7+8 = 0.
13. Generate new checksum at octet 7+8 as described in [RFC0768].
14. Forward to destination based on the unmodified 4-tuple of IP-addresses and ports.

### 3.9. U-DCCP negotiation (required???)

Tbd later if required. Otherwise assumes explicit knowledge about the U-DCCP conversion between sender and receiver.

## 4. Security Considerations

TBD.

## 5. IANA Considerations

## 6. Notes

This document is inspired by [RFC6773] and some text passages for the -00 version are copied unmodified.

## 7. Acknowledgments

## 8. Informative References

- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <<https://www.rfc-editor.org/info/rfc768>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI 10.17487/RFC4340, March 2006, <<https://www.rfc-editor.org/info/rfc4340>>.
- [RFC4787] Audet, F., Ed. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, DOI 10.17487/RFC4787, January 2007, <<https://www.rfc-editor.org/info/rfc4787>>.

- [RFC5596] Fairhurst, G., "Datagram Congestion Control Protocol (DCCP) Simultaneous-Open Technique to Facilitate NAT/Middlebox Traversal", RFC 5596, DOI 10.17487/RFC5596, September 2009, <<https://www.rfc-editor.org/info/rfc5596>>.
- [RFC5597] Denis-Courmont, R., "Network Address Translation (NAT) Behavioral Requirements for the Datagram Congestion Control Protocol", BCP 150, RFC 5597, DOI 10.17487/RFC5597, September 2009, <<https://www.rfc-editor.org/info/rfc5597>>.
- [RFC6773] Phelan, T., Fairhurst, G., and C. Perkins, "DCCP-UDP: A Datagram Congestion Control Protocol UDP Encapsulation for NAT Traversal", RFC 6773, DOI 10.17487/RFC6773, November 2012, <<https://www.rfc-editor.org/info/rfc6773>>.
- [RFC6888] Perreault, S., Ed., Yamagata, I., Miyakawa, S., Nakagawa, A., and H. Ashida, "Common Requirements for Carrier-Grade NATs (CGNs)", BCP 127, RFC 6888, DOI 10.17487/RFC6888, April 2013, <<https://www.rfc-editor.org/info/rfc6888>>.
- [RFC7857] Penno, R., Perreault, S., Boucadair, M., Ed., Sivakumar, S., and K. Naito, "Updates to Network Address Translation (NAT) Behavioral Requirements", BCP 127, RFC 7857, DOI 10.17487/RFC7857, April 2016, <<https://www.rfc-editor.org/info/rfc7857>>.

#### Authors' Addresses

Markus Amend  
Deutsche Telekom  
Deutsche-Telekom-Allee 7  
64295 Darmstadt  
Germany

Email: [Markus.Amend@telekom.de](mailto:Markus.Amend@telekom.de)

Anna Brunstrom  
Karlstad University  
Universitetsgatan 2  
651 88 Karlstad  
Sweden

Email: [anna.brunstrom@kau.se](mailto:anna.brunstrom@kau.se)

Andreas Kassler  
Karlstad University  
Universitetsgatan 2  
651 88 Karlstad  
Sweden

Email: andreas.kassler@kau.se

Veselin Rakocevic  
City University of London  
Northampton Square  
London  
United Kingdom

Email: veselin.rakocevic.1@city.ac.uk

Transport Area Working Group  
Internet-Draft  
Intended status: Experimental  
Expires: 13 January 2022

M. Amend  
D. Hugo  
DT  
A. Brunstrom  
A. Kassler  
Karlstad University  
V. Rakocevic  
City University of London  
S. Johnson  
BT  
12 July 2021

DCCP Extensions for Multipath Operation with Multiple Addresses  
draft-amend-tsvwg-multipath-dccp-05

Abstract

DCCP communication is currently restricted to a single path per connection, yet multiple paths often exist between peers. The simultaneous use of these multiple paths for a DCCP session could improve resource usage within the network and, thus, improve user experience through higher throughput and improved resilience to network failures. Use cases for a Multipath DCCP (MP-DCCP) are mobile devices (handsets, vehicles) and residential home gateways simultaneously connected to distinct paths as, e.g., a cellular link and a WiFi link or to a mobile radio station and a fixed access network. Compared to existing multipath protocols such as MPTCP, MP-DCCP provides specific support for non-TCP user traffic as UDP or plain IP. More details on potential use cases are provided in [website], [slide] and [paper]. All these use cases profit from an Open Source Linux reference implementation provided under [website].

This document presents a set of extensions to traditional DCCP to support multipath operation. Multipath DCCP provides the ability to simultaneously use multiple paths between peers. The protocol offers the same type of service to applications as DCCP and it provides the components necessary to establish and use multiple DCCP flows across potentially disjoint paths.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 13 January 2022.

#### Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

#### Table of Contents

1. Introduction . . . . .	3
1.1. Multipath DCCP in the Networking Stack . . . . .	4
1.2. Terminology . . . . .	4
1.3. MP-DCCP Concept . . . . .	5
1.4. Differences from Multipath TCP . . . . .	5
1.5. Requirements Language . . . . .	9
2. Operation Overview . . . . .	9
3. MP-DCCP Protocol . . . . .	9
3.1. Multipath Capable Feature . . . . .	12
3.2. Multipath Option . . . . .	12
3.2.1. MP_CONFIRM . . . . .	13
3.2.2. MP_JOIN . . . . .	13
3.2.3. MP_FAST_CLOSE . . . . .	14
3.2.4. MP_KEY . . . . .	14
3.2.5. MP_SEQ . . . . .	15
3.2.6. MP_HMAC . . . . .	15
3.2.7. MP_RTT . . . . .	16
3.2.8. MP_ADDADDR . . . . .	17
3.2.9. MP_REMOVEADDR . . . . .	18
3.2.10. MP_PRIO . . . . .	19

3.3. MP-DCCP Handshaking Procedure . . . . .	19
4. Security Considerations . . . . .	21
5. Interactions with Middleboxes . . . . .	22
6. Implementation . . . . .	22
7. Acknowledgments . . . . .	22
8. IANA Considerations . . . . .	23
9. Informative References . . . . .	25
Authors' Addresses . . . . .	28

## 1. Introduction

Multipath DCCP (MP-DCCP) is a set of extensions to regular DCCP [RFC4340], i.e. the Datagram Congestion Control Protocol denoting a transport protocol that provides bidirectional unicast connections of congestion-controlled unreliable datagrams. A multipath extension to DCCP enables the transport of user data across multiple paths simultaneously. This is beneficial to applications that transfer fairly large amounts of data, due to the possibility to aggregate capacity of the multiple paths. In addition, it enables to tradeoff timeliness and reliability, which is important for low latency applications that do not require guaranteed delivery services such as Audio/Video streaming. DCCP multipath operation is suggested in the context of ongoing 3GPP work on 5G multi-access solutions [I-D.amend-tsvwg-multipath-framework-mpdccp] and for hybrid access networks [I-D.lhwxyz-hybrid-access-network-architecture][I-D.muley-net-work-based-bonding-hybrid-access]. It can be applied for load-balancing, seamless session handover, and aggregation purposes (referred to as ATSSS; Access steering, switching, and splitting in 3GPP terminology [TS23.501]).

This document presents the protocol changes required to add multipath capability to DCCP; specifically, those for signaling and setting up multiple paths ("subflows"), managing these subflows, re-assembly of data, and termination of sessions. DCCP, as stated in [RFC4340] does not provide reliable and ordered delivery. Consequently, multiple application subflows may be multiplexed over a single DCCP connection with no inherent performance penalty for flows that do not require in-ordered delivery. DCCP does not provide built-in support for those multiple application subflows.

In the following, use of the term subflow will refer to physical separate DCCP subflows transmitted via different paths, but not to application subflows. Application subflows are differing content-wise by source and destination port per application as, for example, enabled by Service Codes introduced to DCCP in [RFC5595], and those subflows can be multiplexed over a single DCCP connection. For sake of consistency we assume that only a single application is served by a DCCP connection here as shown in Figure 1 while use of that feature should not impact DCCP operation on each single path as noted in ([RFC5595], sect. 2.4).

### 1.1. Multipath DCCP in the Networking Stack

MP-DCCP operates at the transport layer and aims to be transparent to both higher and lower layers. It is a set of additional features on top of standard DCCP; Figure 1 illustrates this layering. MP-DCCP is designed to be used by applications in the same way as DCCP with no changes to the application itself.

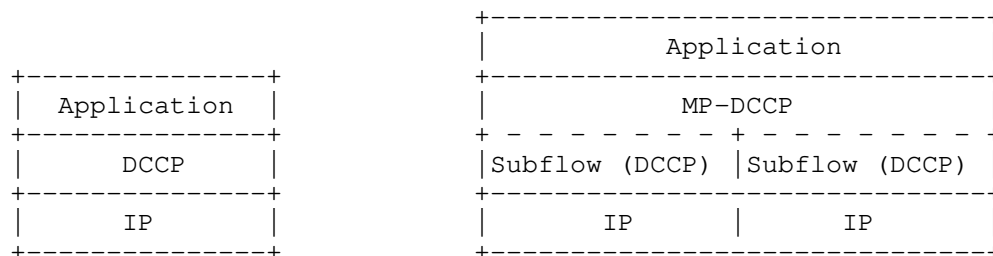


Figure 1: Comparison of Standard DCCP and MP-DCCP Protocol Stacks

### 1.2. Terminology

Throughout this document we make use of terms that are either specific for multipath transport or are defined in the context of MP-DCCP, similar to [RFC8684], as follows:

**Path:** A sequence of links between a sender and a receiver, defined in this context by a 4-tuple of source and destination address/ port pairs.

**Subflow:** A flow of DCCP segments operating over an individual path, which forms part of a larger MP-DCCP connection. A subflow is started and terminated similar to a regular (single-path) DCCP connection.

(MP-DCCP) Connection: A set of one or more subflows, over which an application can communicate between two hosts. There is a one-to-one mapping between a connection and an application socket.

Token: A locally unique identifier given to a multipath connection by a host. May also be referred to as a "Connection ID".

Host: An end host operating an MP-DCCP implementation, and either initiating or accepting an MP-DCCP connection. In addition to these terms, within framework of MP-DCCP the interpretation of, and effect on, regular single-path DCCP semantics is discussed in Section 3.

### 1.3. MP-DCCP Concept

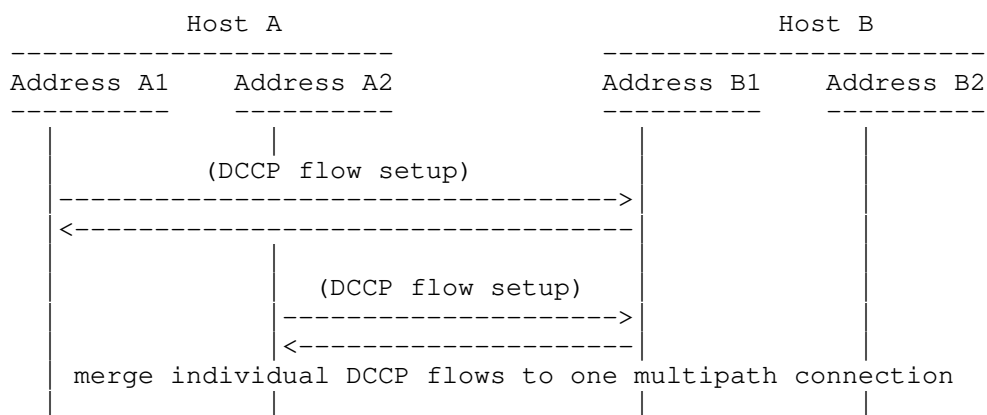


Figure 2: Example MP-DCCP Usage Scenario

### 1.4. Differences from Multipath TCP

Multipath DCCP is similar to Multipath TCP [RFC8684], in that it extends the related basic DCCP transport protocol [RFC4340] with multipath capabilities in the same way as Multipath TCP extends TCP [RFC0793]. However, because of the differences between the underlying TCP and DCCP protocols, the transport characteristics of MPTCP and MP-DCCP are different.



Table 1 compares the protocol characteristics of TCP and DCCP, which are by nature inherited by their respective multipath extensions. A major difference lies in the delivery of payload, which is for TCP an exact copy of the generated byte-stream. DCCP behaves in a different way and does not guarantee to deliver any payload nor the order of delivery. Since this is mainly affecting the receiving endpoint of a TCP or DCCP communication, many similarities on the sender side can be identified. Both transport protocols share the 3-way initiation of a communication and both employ congestion control to adapt the sending rate to the path characteristics.

Feature	TCP	DCCP
Full-Duplex	yes	yes
Connection-Oriented	yes	yes
Header option space	40 bytes	< 1008 bytes or PMTU
Data transfer	reliable	unreliable
Packet-loss handling	re-transmission	report only
Ordered data delivery	yes	no
Sequence numbers	one per byte	one per PDU
Flow control	yes	no
Congestion control	yes	yes
ECN support	yes	yes
Selective ACK	yes	depends on congestion control
Fix message boundaries	no	yes
Path MTU discovery	yes	yes
Fragmentation	yes	no
SYN flood protection	yes	no
Half-open connections	yes	no

Table 1: TCP and DCCP protocol comparison

Consequently, the multipath features, shown in Table 2, are the same, supporting volatile paths having varying capacity and latency, session handover and path aggregation capabilities. All of them profit by the existence of congestion control.

Feature	MPTCP	MP-DCCP
Volatile paths	yes	yes
Session handover	yes	yes
Path aggregation	yes	yes
Robust session establishment	no	yes
Data re-assembly	yes	optional / modular
Expandability	limited by TCP header	flexible

Table 2: MPTCP and MP-DCCP protocol comparison

Therefore, the sender logic is not much different between MP-DCCP and MPTCP, even if the multipath session initiation differs. MP-DCCP inherits a robust session establishment feature, which guarantees communication establishment if at least one functional path is available. MPTCP relies on an initial path, which has to work; otherwise no communication can be established.

The receiver side for MP-DCCP has to deal with the unreliable transport character of DCCP and a possible re-assembly of the data stream while not advocating it. As many unreliable applications have built-in application support for reordering (such as adaptive audio and video buffers), those applications might not need support for re-assembly. However, for applications that benefit from partial or full support of reordering, MP-DCCP can provide flexible support for re-assembly, even if for DCCP the order of delivery is unreliable by nature. Such optional re-assembly mechanisms may account for the fact that packet loss may occur for any of the DCCP subflows. Another issue may occur as packet reordering may happen when the different DCCP subflows are routed across paths with different latencies. In theory, applications using DCCP are aware that packet reordering might happen, since DCCP has no mechanisms to prevent it.

The receiving process for MPTCP is on the other hand a rigid "just wait" approach, since TCP guarantees reliable delivery.

### 1.5. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

## 2. Operation Overview

RFC 4340 states that some applications might want to share congestion control state among multiple DCCP flows between same source and destination addresses. This functionality could be provided by the Congestion Manager (CM) [RFC3124], a generic multiplexing facility. However, the CM would not fully support MP-DCCP without change; it does not gracefully handle multiple congestion control mechanisms, for example.

The operation of MP-DCCP for data transfer takes one input data stream from an application, and splits it into one or more subflows, with sufficient control information to allow received data to be re-assembled and delivered in order to the recipient application. The following subsections define this behavior in detail.

The Multipath Capability for MP-DCCP can be negotiated with a new DCCP feature, as described in Section 3. Once negotiated, all subsequent MP-DCCP operations are signalled with a variable length multipath-related option, as described in Section 3.1.

## 3. MP-DCCP Protocol

The DCCP protocol feature list ([RFC4340] section 6.4) will be enhanced by a new Multipath related feature with Feature number 10, as shown in Table 3.

Number	Meaning	Rule	Rec'n Value	Initial Req'd
0	Reserved			
1	Congestion Control ID (CCID)	SP	2	Y
2	Allow Short Seqnos	SP	0	Y
3	Sequence Window	NN	100	Y
4	ECN Incapable	SP	0	N
5	Ack Ratio	NN	2	N
6	Send Ack Vector	SP	0	N
7	Send NDP Count	SP	0	N
8	Minimum Checksum Coverage	SP	0	N
9	Check Data Checksum	SP	0	N
10	Multipath Capable	SP	0	N
11-127	Reserved			
128-255	CCID-specific features			

Table 3: Proposed Feature Set

The DCCP protocol options as defined in ([RFC4340] section 5.8) and ([RFC5634] section 2.2.1) will be enhanced by a new Multipath related variable-length option with option type 46, as shown in Table 4.

Type	Option Length	Meaning	DCCP-Data?
0	1	Padding	Y
1	1	Mandatory	N
2	1	Slow Receiver	Y
3-31	1	Reserved	
32	variable	Change L	N
33	variable	Confirm L	N
34	variable	Change R	N
35	variable	Confirm R	N
36	variable	Init Cookie	N
37	3-8	NDP Count	Y
38	variable	Ack Vector [Nonce 0]	N
39	variable	Ack Vector [Nonce 1]	N
40	variable	Data Dropped	N
41	6	Timestamp	Y
42	6/8/10	Timestamp Echo	Y
43	4/6	Elapsed Time	N
44	6	Data Checksum	Y
45	8	Quick-Start Response	?
46	variable	Multipath	Y
47-127	variable	Reserved	
128-255	variable	CCID-specific options	-

Table 4: Proposed Option Set

[Tbd/tbv] In addition to the multipath option, MP-DCCP requires particular considerations for:

- \* The minimum PMTU of the individual paths must be announced to the application. Changes of individual path PMTUs must be re-announced to the application if they result in a value lower than the currently announced PMTU.
- \* Overall sequencing for optional re-assembly procedure
- \* Congestion control
- \* Robust MP-DCCP session establishment (no dependency on an initial path setup)

### 3.1. Multipath Capable Feature

DCCP endpoints are multipath-disabled by default and multipath capability can be negotiated with the Multipath Capable Feature.

Multipath Capable has feature number 10 and is server-priority. It takes one-byte values. The first four bits are used to specify compatible versions of the MP-DCCP implementation. The following four bits are reserved for further use.

### 3.2. Multipath Option

```

+-----+-----+-----+-----+-----+
|00101110| Length | MP_OPT | Value(s) ...
+-----+-----+-----+-----+-----+
Type=46

```

Type	Option Length	MP_OPT	Meaning
46	var	0 =MP_CONFIRM	Confirm reception and processing of an MP_OPT option
46	11	1 =MP_JOIN	Join path to an existing MP-DCCP flow
46	3	2 =MP_FAST_CLOSE	Close MP-DCCP flow
46	var	3 =MP_KEY	Exchange key material for MP_HMAC
46	7	4 =MP_SEQ	Multipath Sequence Number
46	23	5 =MP_HMAC	HMA Code for authentication
46	12	6 =MP_RTT	Transmit RTT values
46	var	7 =MP_ADDADDR	Advertise additional Address
46	var	8 =MP_REMOVEADDR	Remove Address
46	4	9 =MP_PRIO	Change Subflow Priority

Table 5: MP\_OPT Option Types

## 3.2.1. MP\_CONFIRM

00101110	Length	00000000	List of options ...
Type=46	MP_OPT=0		

MP\_CONFIRM can be used to send confirmation of received and processed options. Confirmed options are copied verbatim and appended as List of options. The length varies dependent on the amount of options.

[Tbd] Encoding "list of options"

## 3.2.2. MP\_JOIN



```

+-----+-----+-----+-----+-----+-----+
|00101110|00001011|00000001| Path Token |
+-----+-----+-----+-----+-----+-----+
| Nonce |
+-----+-----+-----+-----+
Type=46 Length=11 MP_OPT=1

```

The MP\_JOIN option is used to add a new path to an existing MP-DCCP flow. The Path Token is the SHA-1 HASH of the derived key (d-key), which was previously exchanged with the MP\_KEY option. MP\_HMAC MUST be set when using MP\_JOIN to provide authentication (See MP\_HMAC for details). Also MP\_KEY MUST be set to provide key material for authentication purposes.

### 3.2.3. MP\_FAST\_CLOSE

```

+-----+-----+-----+
|00101110|00000011|00000010|
+-----+-----+-----+
Type=46 Length=3 MP_OPT=2

```

MP\_FAST\_CLOSE terminates the MP-DCCP flow and all corresponding subflows.

### 3.2.4. MP\_KEY

```

+-----+-----+-----+-----+-----+-----+
|00101110| Length |00000011|Key Type| Key Data ...
+-----+-----+-----+-----+-----+-----+
Type=46 MP_OPT=3

```

The MP\_KEY suboption is used to exchange key material between hosts. The Length varies between 5 and 8 Bytes. The Key Type field is used to specify the key type. Key types are shown in Table 6.

Key Type	Key Length	Meaning
0 =Plain Text	8	Plain Text Key
1 =ECDHE-C25519-SHA256	32	ECDHE with SHA256 and Curve25519
2 =ECDHE-C25519-SHA512	32	ECDHE with SHA512 and Curve25519
3-255		Reserved

Table 6: MP\_KEY Key Types

#### Plain Text

Key Material is exchanged in plain text between hosts, and the key parts (key-a, key-b) are used by each host to generate the derived key (d-key) by concatenating the two parts with the local key in front (e.g. hostA d-key=(key-a+key-b), hostB d-key=(key-b+key-a)).

#### ECDHE-SHA256-C25519

Key Material is exchanged via ECDHE key exchange with SHA256 and Curve 25519 to generate the derived key (d-key).

#### ECDHE-SHA512-C25519

Key Material is exchanged via ECDHE key exchange with SHA512 and Curve 25519 to generate the derived key (d-key).

#### 3.2.5. MP\_SEQ

```

+-----+-----+-----+-----+
| 00101110 | 00000111 | 00000100 | Multipath Sequence Number |
+-----+-----+-----+-----+
Type=46 Length=7 MP_OPT=4

```

The MP\_SEQ option is used for end-to-end datagram-based sequence numbers of an MP-DCCP connection. The initial data sequence number (IDSN) SHOULD be set randomly. The MP\_SEQ number space is different from path individual sequence number space.

#### 3.2.6. MP\_HMAC

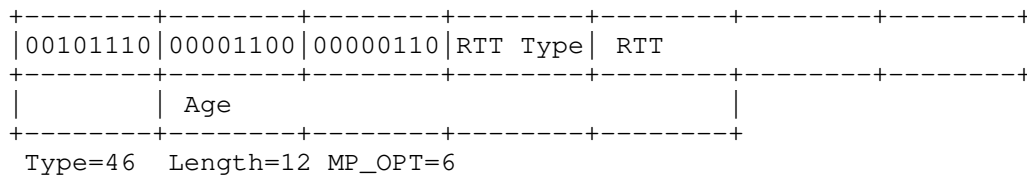
```

+-----+-----+-----+-----+
| 00101110 | 00001011 | 00000101 | HMAC-SHA1 (20 bytes) ... |
+-----+-----+-----+-----+
Type=46 Length=23 MP_OPT=5

```

The MP\_HMAC option is used to provide authentication for the MP\_JOIN option. The HMAC is built using the derived key (d-key) calculated previously from the handshake key material exchanged with the MP\_KEY option. The Message for the HMAC is the header of the MP\_JOIN for which authentication shall be performed. By including a nonce in these datagrams, possible replay-attacks are remedied.

### 3.2.7. MP\_RTT



The MP\_RTT option is used to transmit RTT values in milliseconds and MUST belong to the path over which this information is transmitted. Additionally, the age of the measurement is specified in milliseconds.

#### Raw RTT (=0)

Raw RTT value of the last Datagram Round-Trip. The Age parameter is set to the age of when the Ack for the datagram was received.

#### Min RTT (=1)

Min RTT value. The period for computing the Minimum can be specified by the Age parameter.

#### Max RTT (=2)

Max RTT value. The period for computing the Maximum can be specified by the Age parameter.

#### Smooth RTT (=3)

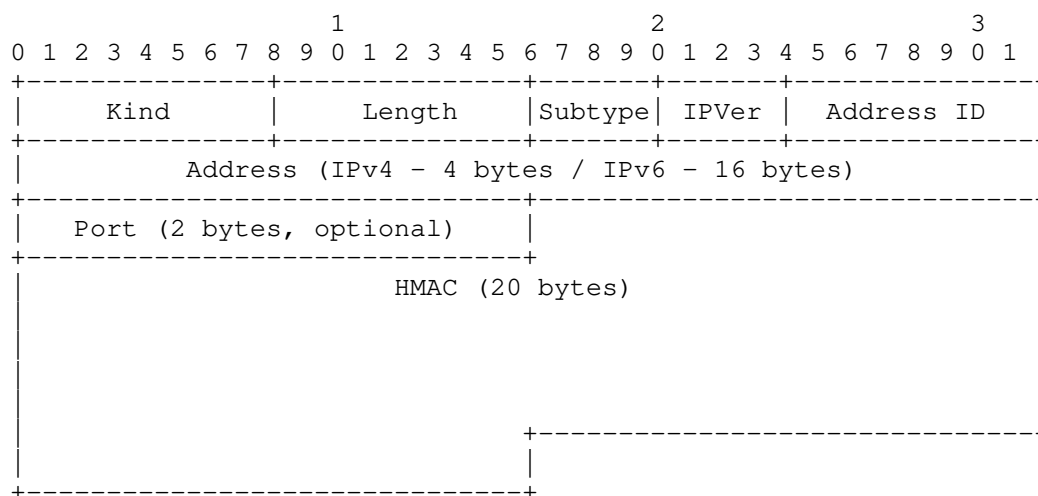
Averaged RTT value. The period for computing the smoothed RTT can be specified by the Age parameter.

#### Age (=4)

The Age parameter is a 4-byte value which is set to the age or timestamp when the Ack for the datagram was received in case of RTT type = 0 and may contain the periods for computing of derived RTT values depending on other RTT types, i.e., the Minimum (=1) and Maximum (=2) as well as the averaged smoothed RTT value (=3). [TBD/TBV]

## 3.2.8. MP\_ADDADDR

The MP\_ADDADDR option announces additional addresses (and, optionally, ports) on which a host can be reached. This option can be used at any time during an existing DCCP connection, when the sender wishes to enable multiple paths and/or when additional paths become available. Length is variable depending on IPv4 or IPv6 and whether port number is used and is in range between 28 and 42 bytes.



Every address has an Address ID that can be used for uniquely identifying the address within a connection for address removal. The Address ID is also used to identify MP\_JOIN options (see Section 3.2.2) relating to the same address, even when address translators are in use. The Address ID MUST uniquely identify the address for the sender of the option (within the scope of the connection); the mechanism for allocating such IDs is implementation specific.

All Address IDs learned via either MP\_JOIN or ADD\_ADDR SHOULD be stored by the receiver in a data structure that gathers all the Address-ID-to-address mappings for a connection (identified by a token pair). In this way, there is a stored mapping between the Address ID, observed source address, and token pair for future processing of control information for a connection.

Ideally, ADD\_ADDR and REMOVE\_ADDR options would be sent reliably, and in order, to the other end. This would ensure that this address management does not unnecessarily cause an outage in the connection when remove/add addresses are processed in reverse order, and also to ensure that all possible paths are used. Note, however, that losing

reliability and ordering will not break the multipath connections, it will just reduce the opportunity to open new paths and to survive different patterns of path failures.

Therefore, implementing reliability signals for these DCCP options is not necessary. In order to minimize the impact of the loss of these options, however, it is RECOMMENDED that a sender should send these options on all available subflows. If these options need to be received in order, an implementation SHOULD only send one ADD\_ADDR/REMOVE\_ADDR option per RTT, to minimize the risk of misordering. A host that receives an ADD\_ADDR but finds a connection set up to that IP address and port number is unsuccessful SHOULD NOT perform further connection attempts to this address/port combination for this connection. A sender that wants to trigger a new incoming connection attempt on a previously advertised address/port combination can therefore refresh ADD\_ADDR information by sending the option again.

[TBD/TBV]

### 3.2.9. MP\_REMOVEADDR

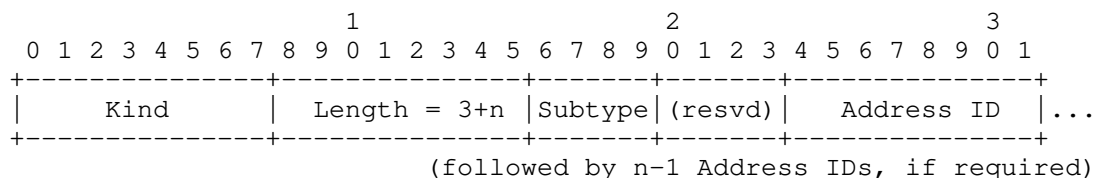
If, during the lifetime of an MP-DCCP connection, a previously announced address becomes invalid (e.g., if the interface disappears), the affected host SHOULD announce this so that the peer can remove subflows related to this address.

This is achieved through the Remove Address (REMOVE\_ADDR) option which will remove a previously added address (or list of addresses) from a connection and terminate any subflows currently using that address.

For security purposes, if a host receives a REMOVE\_ADDR option, it must ensure the affected path(s) are no longer in use before it instigates closure. Typical DCCP validity tests on the subflow (e.g., packet type specific sequence and acknowledgement number check) MUST also be undertaken. An implementation can use indications of these test failures as part of intrusion detection or error logging.

The sending and receipt of this message SHOULD trigger the sending of DCCP-Close and DCCP-Reset by client and server, respectively on the affected subflow(s) (if possible), as a courtesy to cleaning up middlebox state, before cleaning up any local state.

Address removal is undertaken by ID, so as to permit the use of NATs and other middleboxes that rewrite source addresses. If there is no address at the requested ID, the receiver will silently ignore the request.

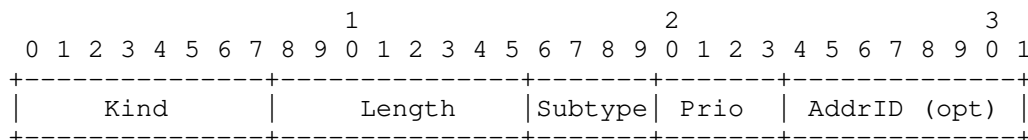


Minimum length of this option is 4 bytes (for one address to remove).

[TBD/TBV]

### 3.2.10. MP\_PRIO

In the event that a single specific path out of the set of available paths shall be treated with higher priority compared to the others, a host may wish to signal such change in priority of subflows to the peer. Therefore, the MP\_PRIO option, shown below, can be used to set a priority flag for the subflow on which it is sent.



Whether more than two values for priority (e.g., B for backup and P for prioritized path) are defined in case of more than two parallel paths is for further consideration.

[TBD/TBV]

### 3.3. MP-DCCP Handshaking Procedure

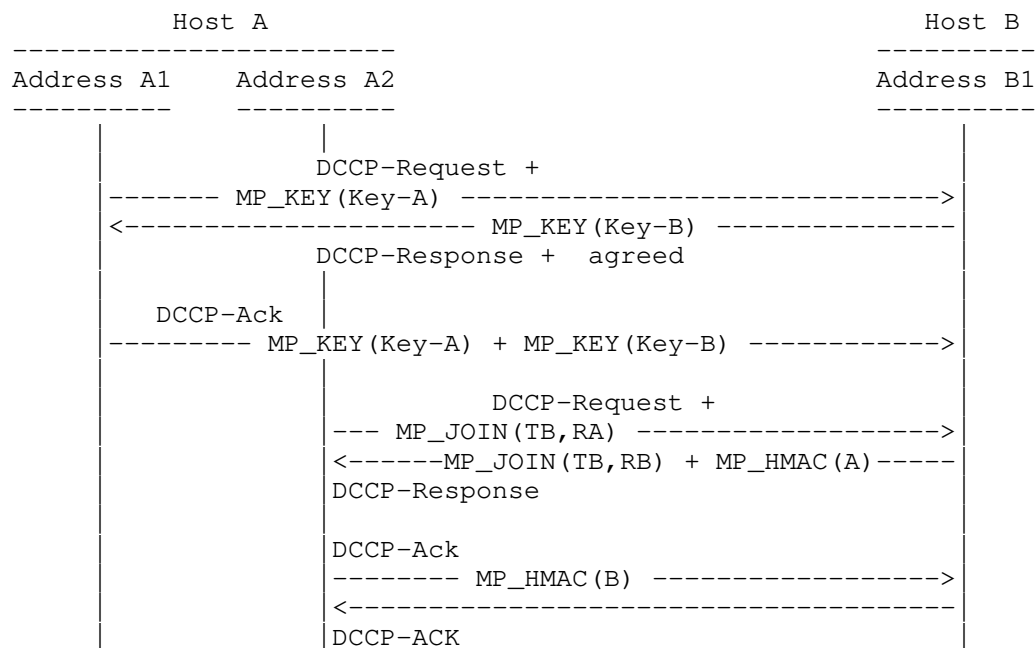


Figure 3: Example MP-DCCP Handshake

The basic initial handshake for the first flow is as follows:

- \* Host A sends a DCCP-Request with the MP-Capable feature Change request and the MP\_KEY option with Host-specific Key-A
- \* Host B sends a DCCP-Response with Confirm feature for MP-Capable and the MP\_Key option with Host-specific Key-B
- \* Host A sends a DCCP-Ack with both Keys echoed to Host B.

The handshake for subsequent flows based on a successful initial handshake is as follows:

- \* Host A sends a DCCP-Request with the MP-Capable feature Change request and the MP\_JOIN option with Host B's Token TB, generated from the derived key by applying a SHA-1 hash and truncating to the first 32 bits. Additionally, an own random nonce RA is transmitted with the MP\_JOIN.
- \* Host B computes the HMAC of the DCCP-Request and sends a DCCP-Response with Confirm feature option for MP-Capable and the MP\_JOIN option with the Token TB and a random nonce RB together with the computed MP\_HMAC.

- \* Host A sends a DCCP-Ack with the HMAC computed for the DCCP-Response.
- \* Host B sends a DCCP-Ack confirm the HMAC and to conclude the handshaking.

#### 4. Security Considerations

Similar to DCCP, MP-DCCP does not provide cryptographic security guarantees inherently. Thus, if applications need cryptographic security (integrity, authentication, confidentiality, access control, and anti-replay protection) the use of IPsec or some other kind of end-to-end security is recommended; Secure Real-time Transport Protocol (SRTP) [RFC3711] is one candidate protocol for authentication. Together with Encryption of Header Extensions in SRTP, as provided by [RFC6904], also integrity would be provided.

As described in [RFC4340], DCCP provides protection against hijacking and limits the potential impact of some denial-of-service attacks, but DCCP provides no inherent protection against attackers' snooping on data packets. Regarding the security of MP-DCCP no additional risks should be introduced compared to regular DCCP of today. Thereof derived are the following key security requirements to be fulfilled by MP-DCCP:

- \* Provide a mechanism to confirm that parties involved in a subflow handshake are identical to those in the original connection setup.
- \* Provide verification that the new address to be included in a MP connection is valid for a peer to receive traffic at before using it.
- \* Provide replay protection, i.e., ensure that a request to add/remove a subflow is 'fresh'.

In order to achieve these goals, MP-DCCP includes a hash-based handshake algorithm documented in Sections Section 3.2.4 and Section 3.3. The security of the MP-DCCP connection depends on the use of keys that are shared once at the start of the first subflow and are never sent again over the network. To ease demultiplexing while not giving away any cryptographic material, future subflows use a truncated cryptographic hash of this key as the connection identification "token". The keys are concatenated and used as keys for creating Hash-based Message Authentication Codes (HMACs) used on subflow setup, in order to verify that the parties in the handshake are the same as in the original connection setup. It also provides verification that the peer can receive traffic at this new address. Replay attacks would still be possible when only keys are used;



therefore, the handshakes use single-use random numbers (nonces) at both ends -- this ensures that the HMAC will never be the same on two handshakes. Guidance on generating random numbers suitable for use as keys is given in [RFC4086]. During normal operation, regular DCCP protection mechanisms (such as header checksum to protect DCCP headers against corruption) will provide the same level of protection against attacks on individual DCCP subflows as exists for regular DCCP today.

## 5. Interactions with Middleboxes

Issues from interaction with on-path middleboxes such as NATs, firewalls, proxies, intrusion detection systems (IDSs), and others have to be considered for all extensions to standard protocols since otherwise unexpected reactions of middleboxes may hinder its deployment. DCCP already provides means to mitigate the potential impact of middleboxes, also in comparison to TCP (see [RFC4043], sect. 16). In case, however, both hosts are located behind a NAT or firewall entity, specific measures have to be applied such as the [RFC5596]-specified simultaneous-open technique that update the (traditionally asymmetric) connection-establishment procedures for DCCP. Further standardized technologies addressing NAT type middleboxes are covered by [RFC5597].

[RFC6773] specifies UDP Encapsulation for NAT Traversal of DCCP sessions, similar to other UDP encapsulations such as for SCTP [RFC6951]. The alternative U-DCCP approach proposed in [I-D.amend-tsvwg-dccp-udp-header-conversion] would reduce tunneling overhead. The handshaking procedure for DCCP-UDP header conversion or use of a DCCP-UDP negotiation procedure to signal support for DCCP-UDP header conversion would require encapsulation during the handshakes and use of two additional port numbers out of the UDP port number space, but would require zero overhead afterwards.

## 6. Implementation

The approach described above has been implemented in open source across different testbeds and a new scheduling algorithm has been extensively tested. Also demonstrations of a laboratory setup have been executed and have been published at [website].

## 7. Acknowledgments

### 1. Notes

This document is inspired by Multipath TCP [RFC6824]/[RFC8684] and some text passages for the -00 version of the draft are copied almost unmodified.

## 8. IANA Considerations

This document defines one new value to DCCP feature list and one new DCCP Option with ten corresponding Subtypes as follows. This document defines a new DCCP feature parameter for negotiating the support of multipath capability for DCCP sessions between hosts as described in Section 3. The following entry in Table 7 should be added to the "Feature Numbers Registry" according to [RFC4340], Section 19.4. under the "DCCP Protocol" heading.

Value	Feature Name	Specification
0x10	MP-DCCP capability feature	Section 3.1

Table 7: Addition to DCCP Feature list Entries

This document defines a new DCCP protocol option of type=46 as described in Section 3.2 together with 10 additional sub-options. The following entries in Table 8 should be added to the "DCCP Protocol options" and assigned as "MP-DCCP sub-options", respectively.

Value	Symbol	Name	Reference
TBD or Type=46	MP_OPT	DCCP Multipath option	Section 3.2
TBD or MP_OPT=0	MP_CONFIRM	Confirm reception/ processing of an MP_OPT option	Section 3.2.1
TBD or MP_OPT=1	MP_JOIN	Join path to existing MP-DCCP flow	Section 3.2.2
TBD or MP_OPT=2	MP_FAST_CLOSE	Close MP-DCCP flow	Section 3.2.3
TBD or MP_OPT=3	MP_KEY	Exchange key material for MP_HMAC	Section 3.2.4
TBD or MP_OPT=4	MP_SEQ	Multipath Sequence Number	Section 3.2.5
TBD or MP_OPT=5	MP_HMAC	Hash-based Message Auth. Code for MP- DCCP	Section 3.2.6
TBD or MP_OPT=6	MP_RTT	Transmit RTT values and calculation parameters	Section 3.2.7
TBD or MP_OPT=7	MP_ADDADDR	Advertise additional Address(es)/Port(s)	Section 3.2.8
TBD or MP_OPT=8	MP_REMOVEADDR	Remove Address(es)/ Port(s)	Section 3.2.9
TBD or MP_OPT=9	MP_PRIO	Change Subflow Priority	Section 3.2.10

Table 8: Addition to DCCP Protocol options and  
corresponding sub-options

[Tbd], must include options for:

- \* handshaking procedure to indicate MP support
- \* handshaking procedure to indicate JOINING of an existing MP connection
- \* signaling of new or changed addresses
- \* setting handover or aggregation mode
- \* setting reordering on/off

should include options carrying:

- \* overall sequence number for restoring purposes
- \* sender time measurements for restoring purposes
- \* scheduler preferences
- \* reordering preferences

## 9. Informative References

[I-D.amend-tsvwg-dccp-udp-header-conversion]

Amend, M., Brunstrom, A., Kassler, A., and V. Rakocevic, "Lossless and overhead free DCCP - UDP header conversion (U-DCCP)", Work in Progress, Internet-Draft, draft-amend-tsvwg-dccp-udp-header-conversion-01, 8 July 2019, <<https://www.ietf.org/archive/id/draft-amend-tsvwg-dccp-udp-header-conversion-01.txt>>.

[I-D.amend-tsvwg-multipath-framework-mpdccp]

Amend, M., Bogenfeld, E., Brunstrom, A., Kassler, A., and V. Rakocevic, "A multipath framework for UDP traffic over heterogeneous access networks", Work in Progress, Internet-Draft, draft-amend-tsvwg-multipath-framework-mpdccp-01, 8 July 2019, <<https://www.ietf.org/archive/id/draft-amend-tsvwg-multipath-framework-mpdccp-01.txt>>.

[I-D.lhwxz-hybrid-access-network-architecture]

Leymann, N., Heidemann, C., Wesserman, M., Xue, L., and M. Zhang, "Hybrid Access Network Architecture", Work in Progress, Internet-Draft, draft-lhwxz-hybrid-access-network-architecture-02, 13 January 2015, <<https://www.ietf.org/archive/id/draft-lhwxz-hybrid-access-network-architecture-02.txt>>.

- [I-D.muley-network-based-bonding-hybrid-access]  
Muley, P., Henderickx, W., Liang, G., Liu, H., Cardullo, L., Newton, J., Seo, S., Draznin, S., and B. Patil, "Network based Bonding solution for Hybrid Access", Work in Progress, Internet-Draft, draft-muley-network-based-bonding-hybrid-access-03, 22 October 2018, <<https://www.ietf.org/archive/id/draft-muley-network-based-bonding-hybrid-access-03.txt>>.
- [paper] Amend, M., Bogenfeld, E., Cvjetkovic, M., Rakocevic, V., Pieska, M., Kassler, A., and A. Brunstrom, "A Framework for Multiaccess Support for Unreliable Internet Traffic using Multipath DCCP", DOI 10.1109/LCN44214.2019.8990746, October 2019, <<https://doi.org/10.1109/LCN44214.2019.8990746>>.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3124] Balakrishnan, H. and S. Seshan, "The Congestion Manager", RFC 3124, DOI 10.17487/RFC3124, June 2001, <<https://www.rfc-editor.org/info/rfc3124>>.
- [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <<https://www.rfc-editor.org/info/rfc3711>>.
- [RFC4043] Pinkas, D. and T. Gindin, "Internet X.509 Public Key Infrastructure Permanent Identifier", RFC 4043, DOI 10.17487/RFC4043, May 2005, <<https://www.rfc-editor.org/info/rfc4043>>.
- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <<https://www.rfc-editor.org/info/rfc4086>>.
- [RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI 10.17487/RFC4340, March 2006, <<https://www.rfc-editor.org/info/rfc4340>>.

- [RFC5595] Fairhurst, G., "The Datagram Congestion Control Protocol (DCCP) Service Codes", RFC 5595, DOI 10.17487/RFC5595, September 2009, <<https://www.rfc-editor.org/info/rfc5595>>.
- [RFC5596] Fairhurst, G., "Datagram Congestion Control Protocol (DCCP) Simultaneous-Open Technique to Facilitate NAT/Middlebox Traversal", RFC 5596, DOI 10.17487/RFC5596, September 2009, <<https://www.rfc-editor.org/info/rfc5596>>.
- [RFC5597] Denis-Courmont, R., "Network Address Translation (NAT) Behavioral Requirements for the Datagram Congestion Control Protocol", BCP 150, RFC 5597, DOI 10.17487/RFC5597, September 2009, <<https://www.rfc-editor.org/info/rfc5597>>.
- [RFC5634] Fairhurst, G. and A. Sathiaselalan, "Quick-Start for the Datagram Congestion Control Protocol (DCCP)", RFC 5634, DOI 10.17487/RFC5634, August 2009, <<https://www.rfc-editor.org/info/rfc5634>>.
- [RFC6773] Phelan, T., Fairhurst, G., and C. Perkins, "DCCP-UDP: A Datagram Congestion Control Protocol UDP Encapsulation for NAT Traversal", RFC 6773, DOI 10.17487/RFC6773, November 2012, <<https://www.rfc-editor.org/info/rfc6773>>.
- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <<https://www.rfc-editor.org/info/rfc6824>>.
- [RFC6904] Lennox, J., "Encryption of Header Extensions in the Secure Real-time Transport Protocol (SRTP)", RFC 6904, DOI 10.17487/RFC6904, April 2013, <<https://www.rfc-editor.org/info/rfc6904>>.
- [RFC6951] Tuexen, M. and R. Stewart, "UDP Encapsulation of Stream Control Transmission Protocol (SCTP) Packets for End-Host to End-Host Communication", RFC 6951, DOI 10.17487/RFC6951, May 2013, <<https://www.rfc-editor.org/info/rfc6951>>.
- [RFC8684] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 8684, DOI 10.17487/RFC8684, March 2020, <<https://www.rfc-editor.org/info/rfc8684>>.

- [slide] Amend, M., "MP-DCCP for enabling transfer of UDP/IP traffic over multiple data paths in multi-connectivity networks", IETF105 , n.d.,  
<<https://datatracker.ietf.org/meeting/105/materials/slides-105-tsvwg-sessa-62-dccp-extensions-for-multipath-operation-00>>.
- [TS23.501] 3GPP, "System architecture for the 5G System; Stage 2; Release 16", December 2020,  
<[https://www.3gpp.org/ftp//Specs/archive/23\\_series/23.501/23501-g70.zip](https://www.3gpp.org/ftp//Specs/archive/23_series/23.501/23501-g70.zip)>.
- [website] "Multipath extension for DCCP", n.d.,  
<<https://multipath-dccp.org/>>.

## Authors' Addresses

Markus Amend  
Deutsche Telekom  
Deutsche-Telekom-Allee 9  
64295 Darmstadt  
Germany

Email: Markus.Amend@telekom.de

Dirk von Hugo  
Deutsche Telekom  
Deutsche-Telekom-Allee 9  
64295 Darmstadt  
Germany

Email: Dirk.von-Hugo@telekom.de

Anna Brunstrom  
Karlstad University  
Universitetsgatan 2  
SE-651 88 Karlstad  
Sweden

Email: anna.brunstrom@kau.se

Andreas Kassler  
Karlstad University  
Universitetsgatan 2  
SE-651 88 Karlstad  
Sweden

Email: andreas.kassler@kau.se

Veselin Rakocevic  
City University of London  
Northampton Square  
London  
United Kingdom

Email: veselin.rakocevic.1@city.ac.uk

Stephen Johnson  
BT  
Adastral Park  
Martlesham Heath  
IP5 3RE  
United Kingdom

Email: stephen.h.johnson@bt.com



Transport Area Working Group  
Internet-Draft  
Intended status: Informational  
Expires: January 9, 2020

M. Amend  
E. Bogenfeld  
Deutsche Telekom  
A. Brunstrom  
A. Kassler  
Karlstad University  
V. Rakocevic  
City University of London  
July 08, 2019

A multipath framework for UDP traffic over heterogeneous access networks  
draft-amend-tsvwg-multipath-framework-mpdccp-01

## Abstract

More and more of today's devices are multi-homing capable, in particular 3GPP user equipment like smartphones. In the current standardization of the next upcoming mobile network generation 5G Rel.16, this is especially targeted in the study group Access Traffic Steering Switching Splitting [TR23.793]. ATSSS describes the flexible selection or combination of 3GPP untrusted access like Wi-Fi and cellular access, overcoming the single-access limitation of today's devices and services. Another multi-connectivity scenario is the Hybrid Access [I-D.lhwxyz-hybrid-access-network-architecture][I-D.muley-network-based-bonding-hybrid-access], providing multiple access for CPEs, which extends the traditional way of single access connectivity at home to dual-connectivity over 3GPP and fixed access. A missing piece in the ATSSS and Hybrid Access is the access and path measurement, which is required for efficient and beneficial traffic steering decisions. This becomes particularly important in heterogeneous access networks with a multitude of volatile access paths. While MP-TCP has been proposed to be used within ATSSS, there are drawbacks when being used to encapsulate unreliable traffic as it blindly retransmits each lost frame leading to excessive delay and potential head-of-line blocking. A decision for MP-TCP though leaves the increasing share of UDP in today's traffic mix (<<https://arxiv.org/abs/1801.05168>>) unconsidered. In this document, a multi-access framework is proposed leveraging the MP-DCCP network protocol, which enables flexible traffic steering, switching and splitting also for unreliable traffic. A benefit is the support for pluggable congestion control which enables our framework to be used either independent or complementary to MP-TCP.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 9, 2020.

## Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Requirements . . . . .	3
3. IP compatible multipath framework based on MP-DCCP . . . . .	4
4. Application and placement . . . . .	6
5. Conclusion . . . . .	7
6. Security Considerations . . . . .	8
7. Acknowledgments . . . . .	8
8. Informative References . . . . .	8
Authors' Addresses . . . . .	9

## 1. Introduction

Multi-connectivity access networks are evolving. Ongoing standardization at 3GPP for 5G mobile networks [TR23.793] or the so called Hybrid Access networks [I-D.lhwxyz-hybrid-access-network-architecture][I-D.muley-network-based-bonding-hybrid-access] already provides or will enable in the near future the possibility to use multi-connectivity for a very large number of mobile users. Multi-connectivity solutions come with many user benefits including superior resilience against network outages, higher capacities for user traffic and network cost optimizations. Since multi-connectivity architectures are almost mature, new network protocols are required to fully exploit multi-connectivity and maximise its potential. In the simplest case, multi-connectivity is used for load-balancing decisions in order to balance the user flows over multiple paths. However, this has no effect on resilience or capacity gain on those load balanced individual flows. More complex scenarios include the dynamic shifting of traffic flows seamlessly between multiple paths or even aggregating those paths for leveraging the available capacity of multiple individual paths. Like [TR23.793] this document refers to the three distribution schemes Steering (load balancing), Switching (seamless handover) and Splitting (capacity aggregation).

MP-TCP [RFC6824] is a protocol, which can be applied in the above mentioned use cases and supports load-balancing, traffic shifting among the multiple paths and also capacity aggregation. Further, it leverages the inherent congestion control from TCP which adapts the sending rate by observing congestion signals from the network. By design, MP-TCP is limited to TCP services as it blindly re-transmits lost packets. Consequently, when MP-TCP is used as a framework for ATSSS, it may re-transmit packets sent from unreliable services such as e.g. UDP unnecessarily. This may lead to head-of-line blocking and increased latency, which is detrimental to real-time services. As future multi-connectivity systems must support latency sensitive traffic that might be transported over unreliable transport, it is not sufficient anymore to rely on supporting only TCP. The increasing share of UDP traffic, mainly impacted by the QUIC introduction, may significantly reduce the share from TCP. It might be expected that with HTTP/3 carried over QUIC [I-D.ietf-quic-http], the previous strong dominance of TCP will be challenged by UDP.

## 2. Requirements

A multiaccess framework shall meet the following requirements:

- o IP compatibility: A multiaccess framework shall be able to transport IP packets and not make any assumptions on which transport protocol is encapsulated.
- o Support for unreliable traffic: A multiaccess framework should provide support for transporting unreliable traffic, such as QUIC or UDP based flows. Therefore, unreliable transmission should be supported.
- o Support for flexible re-ordering: A multiaccess framework should support flexible re-ordering of user traffic, including no re-ordering at all. This requirement is important to support low latency traffic, where the re-creation of packet order may negatively impact delivery latency.
- o Support for flexible congestion control: A multiaccess framework should support flexible congestion control, including the disabling of the congestion control, if the inner traffic is known to be congestion controlled.
- o Support for flexible packet scheduling: A multiaccess framework should support different packet scheduling mechanisms, which should be configurable from the control plane. Examples are cheapest path first, or other more sophisticated schedulers.
- o Lightweight: A multiaccess framework should be lightweight in computational resources and limit the encapsulation overhead.

To use QUIC as part of a multiaccess framework, by for example providing multipath support for QUIC, it could be beneficial if unreliable transmission is supported as well as being able to influence or disable QUIC's congestion control. In addition, it would be beneficial if the encryption of QUIC can be disabled. This is because for ATSSS, it is foreseen that the underlying tunnel from the mobile over public WLANs is based on IPsec.

### 3. IP compatible multipath framework based on MP-DCCP

We propose a new multiaccess framework, which overcomes MP-TCP's restriction to TCP services and provides IP compatibility in Figure 1. The framework employs MP-DCCP [I-D.amend-tsvwg-multipath-dccp] in combination with an encapsulation scheme. For simplification, Figure 1 assumes a traffic direction from the left (sender) to the right (receiver) and requires application in each direction for bi-directional transmission. The framework in Figure 1 can replace or complement MP-TCP to reach IP compatibility.

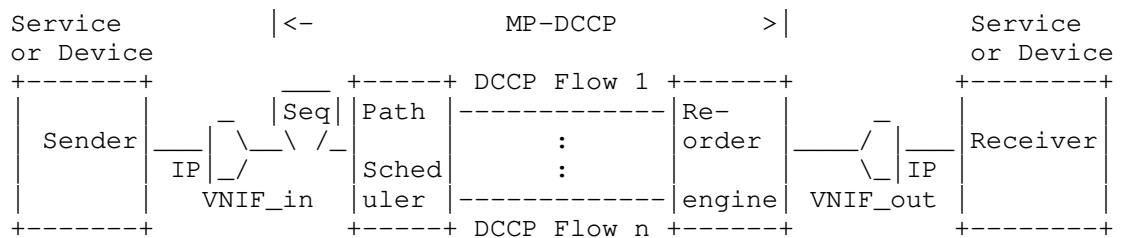


Figure 1: IP compatible multipath framework based on MP-DCCP

PDUs generated from the sender and travelling through the framework in Figure 1 pass the components in the following order:

1. Sender: Generates any PDU based on the IP protocol.
2. VNIF\_in: IP based Virtual Network Interface as entry point to the multipath framework. A simple routing logic in front (between (1) and (2)) can act as gatekeeper and decides upon redirecting traffic through the VNIF or bypassing it. The VNIF adds an extra IP header to reach the multi-connectivity termination point.
3. Seq: Sequencing of the PDUs passed through (2) depending on the incoming order. Adds an incrementing number, which is later added to the DCCP encapsulation in (4).
4. Path Scheduler: Decision logic for scheduling sequenced PDUs over the individual connected DCCP flows for multipath transmission. The path scheduler can use the information from the DCCP flows (see (5)) inherent congestion control information like CWND, packet loss, RTT, Jitter, etc.. After selection of a DCCP flow, the PDU is encapsulated into the individual flow. Further information, at least the sequencing, is added on top as DCCP option.
5. DCCP Flow(s): Responsible to transmit the encapsulated PDUs to the MP-DCCP exit point.
6. Reorder engine: Depending on the sequencing information of (3), a re-assembly of the PDU stream can be applied. Different re-order algorithms should be supported in a configurable way, including no re-ordering.
7. VNIF\_out: Releases PDUs that have passed the re-ordering engine and strips the DCCP specific overhead. Again, routing is responsible to deliver the PDUs to the receiver based on the destination information in the PDU.

8. Receiver: Receive the PDU as generated in (1).

The simple enclosing of the MP-DCCP with Virtual Network Interface (VNIF) provides the IP compatibility. However, a service or protocol classifier between sender and VNIF can reduce the scope to particular traffic, e.g. UDP, by simple routing decisions. The MP-DCCP takes over responsibility for the multi-path transfer of the traffic, which is directed through the VNIF\_in. For possible re-assembly operations, the IP packets may be stamped with a continuously incremented sequence number. This is not mandatory, but assumed required in most seamless handover and capacity aggregation use cases. The path scheduler decides for each IP packet, which DCCP flow it should use for encapsulation, based on a configurable decision logic and supported by the congestion control information of the DCCP flows available for transmission. A DCCP flow selection for a PDU leads to its encapsulation into the respective DCCP flow and adding extra information required for the multipath transmission, e.g. the sequence number. Encapsulation also means, that a DCCP and IP header is added to the original PDU to reach the multi-connectivity end-point. When the encapsulated PDUs arrive at the multi-path termination point, they are re-ordered depending on the carried sequence number and a configurable logic. The re-ordering engine may also include a logic in which packets are just forwarded (no re-ordering). Re-ordering needs to be considered carefully since any active intervention changes the latency responsiveness. The multi-path termination is finally completed when the DCCP overhead is stripped and the PDU leaves VNIF\_out. Further routing depends again on the IP layer of the original PDU.

#### 4. Application and placement

The framework of Figure 1 is very flexible in applying multipath support in different architectures and allows MP-DCCP to be applied at any place between sender and receiver. Figure 2 to Figure 5 provide several architectural options for the deployment of the framework.

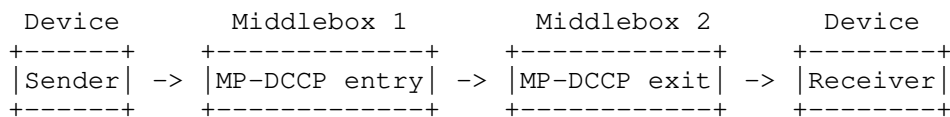


Figure 2: Sender and receiver independent MP-DCCP

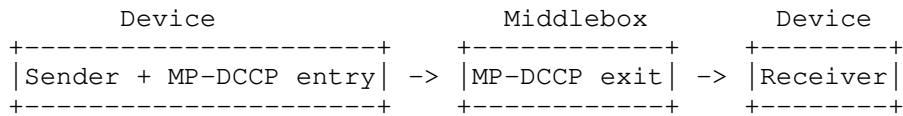


Figure 3: Sender integrated but receiver independent MP-DCCP

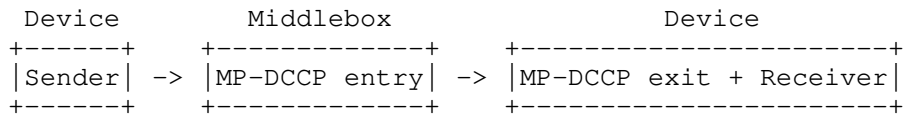


Figure 4: Sender independent and receiver integrated MP-DCCP

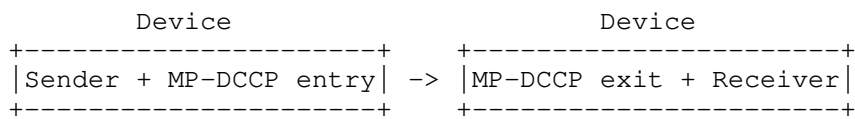


Figure 5: Sender and receiver integrated MP-DCCP

## 5. Conclusion

The specified IP compatible multipath framework based on MP-DCCP in this document comprises several benefits:

- o Pure routing
- o Inherent path estimation and measurement
- o Imposes no constraints on reliability or in-order delivery of application PDUs
- o Modular re-ordering
- o Modular scheduling
- o IP compatible
- o Based on the standardized DCCP.

Middle-box traversing, when the framework is applied in uncontrolled environments, is addressed in [RFC6733] and [I-D.amend-tsvwg-dccp-udp-header-conversion].

## 6. Security Considerations

[Tbd]

## 7. Acknowledgments

## 8. Informative References

- [I-D.amend-tsvwg-dccp-udp-header-conversion]  
Amend, M., Brunstrom, A., Kassler, A., and V. Rakocevic,  
"Lossless and overhead free DCCP - UDP header conversion  
(U-DCCP)", draft-amend-tsvwg-dccp-udp-header-conversion-01  
(work in progress), July 2019.
- [I-D.amend-tsvwg-multipath-dccp]  
Amend, M., Brunstrom, A., Kassler, A., and V. Rakocevic,  
"DCCP Extensions for Multipath Operation with Multiple  
Addresses", draft-amend-tsvwg-multipath-dccp-01 (work in  
progress), March 2019.
- [I-D.ietf-quic-http]  
Bishop, M., "Hypertext Transfer Protocol Version 3  
(HTTP/3)", draft-ietf-quic-http-18 (work in progress),  
January 2019.
- [I-D.lhwxyz-hybrid-access-network-architecture]  
Leymann, N., Heidemann, C., Wasserman, M., Xue, L., and M.  
Zhang, "Hybrid Access Network Architecture", draft-lhwxyz-  
hybrid-access-network-architecture-02 (work in progress),  
January 2015.
- [I-D.muley-network-based-bonding-hybrid-access]  
Muley, P., Henderickx, W., Geng, L., Liu, H., Cardullo,  
L., Newton, J., Seo, S., Draznin, S., and B. Patil,  
"Network based Bonding solution for Hybrid Access", draft-  
muley-network-based-bonding-hybrid-access-03 (work in  
progress), October 2018.
- [RFC6733] Fajardo, V., Ed., Arkko, J., Loughney, J., and G. Zorn,  
Ed., "Diameter Base Protocol", RFC 6733,  
DOI 10.17487/RFC6733, October 2012,  
<<https://www.rfc-editor.org/info/rfc6733>>.
- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure,  
"TCP Extensions for Multipath Operation with Multiple  
Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013,  
<<https://www.rfc-editor.org/info/rfc6824>>.



[TR23.793]

3GPP, "Study on access traffic steering, switch and splitting support in the 5G System (5GS) architecture", December 2018.

Authors' Addresses

Markus Amend  
Deutsche Telekom  
Deutsche-Telekom-Allee 7  
64295 Darmstadt  
Germany

Email: Markus.Amend@telekom.de

Eckard Bogenfeld  
Deutsche Telekom  
Deutsche-Telekom-Allee 7  
64295 Darmstadt  
Germany

Email: Eckard.Bogenfeld@telekom.de

Anna Brunstrom  
Karlstad University  
Universitetsgatan 2  
651 88 Karlstad  
Sweden

Email: anna.brunstrom@kau.se

Andreas Kassler  
Karlstad University  
Universitetsgatan 2  
651 88 Karlstad  
Sweden

Email: andreas.kassler@kau.se

Veselin Rakocevic  
City University of London  
Northampton Square  
London  
United Kingdom

Email: veselin.rakocevic.1@city.ac.uk

Transport Area Working Group  
Internet-Draft  
Intended status: Informational  
Expires: May 5, 2019

B. Briscoe  
CableLabs  
November 1, 2018

Interactions between Low Latency, Low Loss, Scalable Throughput (L4S)  
and Differentiated Services  
draft-briscoe-tsvwg-l4s-diffserv-02

Abstract

L4S and Diffserv offer somewhat overlapping services (low latency and low loss), but bandwidth allocation is out of scope for L4S. Therefore there is scope for the two approaches to complement each other, but also to conflict. This informational document explains how the two approaches interact, how they can be arranged to complement each other and in which cases one can stand alone without needing the other.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 5, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
1.1. Terminology . . . . .	3
1.2. Document Roadmap . . . . .	4
2. Architectural Comparison of L4S and Diffserv . . . . .	4
2.1. Overlaps between L4S and Diffserv . . . . .	4
2.2. Differences between L4S and Diffserv . . . . .	4
3. Low Latency Diffserv Classes within a DualQ Bandwidth Pool . . . . .	5
4. DualQ Bandwidth Pool within a Hierarchy of (Diffserv) Bandwidth Queues . . . . .	9
4.1. DualQ Complemented by an Assured Bandwidth Service . . . . .	10
4.2. DualQ Complemented by a Guaranteed Low Latency Service . . . . .	12
4.3. DualQ Complemented by a Scavenger Service . . . . .	13
5. Coupling More than Two AQMs within a Bandwidth Pool . . . . .	14
6. Applicability of Coupled AQM to Global DiffServ PHBs . . . . .	14
7. Best Practice for Classification and Marking . . . . .	16
7.1. Never Re-Mark a DSCP . . . . .	16
7.2. Classification Order . . . . .	16
7.2.1. Classification Order: Problem . . . . .	17
7.2.2. Classification Order: Solutions . . . . .	17
8. Policing and Traffic Conditioning . . . . .	17
9. IANA Considerations . . . . .	18
10. Security Considerations . . . . .	18
11. Comments Solicited . . . . .	18
12. Acknowledgements . . . . .	18
13. References . . . . .	18
13.1. Normative References . . . . .	18
13.2. Informative References . . . . .	18
Appendix A. Open Issues . . . . .	20
Author's Address . . . . .	20

## 1. Introduction

The Low Latency Low Loss Scalable throughput (L4S) Internet service [I-D.ietf-tsvwg-l4s-arch] provides a new Internet service that could eventually replace best efforts, but with ultra-low queuing delay and loss. A structure called the Dual-Queue Coupled AQM manages to provide the L4S service alongside a second queue for Classic Internet traffic, but without prejudging the bandwidth allocations between them. L4S is orthogonal to allocation of bandwidth, so it can be complemented by various bandwidth allocation approaches without prejudging which one.

The Differentiated Services (Diffserv) architecture [RFC2475] provides for various service classes, some defined globally, others defined locally per network domain. Certain of these service classes offer low latency and low loss, as well as differentiated allocation of bandwidth.

Thus, L4S and Diffserv offer somewhat overlapping services (low latency and low loss), but bandwidth allocation is out of scope for L4S. Therefore there is scope for the two approaches to complement each other, but also to conflict. This informational document explains how the two approaches interact, how they can be arranged to complement each other and in which cases one can stand alone without needing the other.

### 1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

**Classic service:** The 'Classic' service is intended for all the congestion control behaviours that currently co-exist with TCP Reno [RFC5681] (e.g. TCP Cubic, Compound, SCTP, etc).

**Low-Latency, Low-Loss and Scalable (L4S) service:** The 'L4S' service is intended for traffic from scalable congestion control algorithms such as Data Centre TCP [RFC8257]. But it is also more general--it will allow a set of congestion controls with similar scaling properties to DCTCP to evolve.

Both Classic and L4S services can cope with a proportion of unresponsive or less-responsive traffic as well (e.g. DNS, VoIP, etc).

**Pure L4S:** L4S without unresponsive traffic.

**Scalable Congestion Control:** See [I-D.ietf-tsvwg-l4s-arch] for definition.

**Classic Congestion Control:** See [I-D.ietf-tsvwg-l4s-arch] for definition.

**DualQ:** Abbreviation for Dual-Queue Coupled AQM [I-D.ietf-tsvwg-aqm-dualq-coupled], which is not a specific AQM, but a framework for coupling two AQMs in order to provide L4S

service while doing no harm to 'Classic' traffic from traditional sources.

ECN field: The Explicit Congestion Notification field [RFC3168] in the IP header (v4 or v6). [RFC8311] has relaxed some of the restrictions that RFC 3168 placed on the use of ECN, in order to enable experiments like L4S, among others.

Site: A home, mobile device, small enterprise or campus, where the network bottleneck is typically the access link to the site. Not all network arrangements fit this model but it is a useful, widely applicable generalisation.

## 1.2. Document Roadmap

{ToDo}

## 2. Architectural Comparison of L4S and Diffserv

This section compares the L4S architecture [I-D.ietf-tsvwg-l4s-arch] with the Diffserv architecture [RFC2475].

L4S uses an identifier [I-D.ietf-tsvwg-ecn-l4s-id] in the ECN field in IP packet headers that is orthogonal to the Diffserv field [RFC2474]. This is because the two approaches can either overlap or complement each other, as outlined in the following two subsections.

### 2.1. Overlaps between L4S and Diffserv

L4S provides a low queuing latency, low loss Internet Service. Specific Diffserv service classes also provide low latency and low loss.

This means that it is possible to mix traffic from certain Diffserv classes in the same queue as L4S traffic (see Section 3).

### 2.2. Differences between L4S and Diffserv

Bandwidth allocation: L4S is orthogonal to allocation of bandwidth, so it can be complemented by various bandwidth allocation approaches without prejudging which one. In contrast, with Diffserv it was never possible to completely separate control of latency and loss from allocation of bandwidth. The only bandwidth-related aspect of L4S is that it ensures that the capacity seeking behaviour of end-systems can scale with increasing flow rate.

Differentiation vs. General improvement: Diffserv concerns give and take of bandwidth, latency and loss between traffic classes. In contrast, the separation of L4S from Classic traffic in separate queues concerns incremental deployment of a general improvement in latency and loss, without taking from the other queue.

Open vs. closed loop control: The Diffserv architecture requires the source to keep traffic within a contract and, failing that, it has mechanisms to enforce the contract. In this respect, Diffserv is an open-loop control system that is primarily concerned with keeping traffic within capacity limits. Nonetheless, there is an element of closed-loop control in Diffserv. The weighted AQM (e.g. WRED) used for Assured Forwarding [RFC2597] expects traffic to seek to fill capacity and exploits the response to feedback of congestion controllers at traffic sources (closed-loop). Nonetheless, the Diffserv architecture still provides for traffic conditioners that tag traffic that is outside the bandwidth contract for each AF class (open-loop). Then out-of-contract traffic can be discarded if it would otherwise lead to congestion.

L4S uses a similar closed-loop mechanism to the weighted AQM used in Diffserv AF in order to ensure roughly equal per-flow throughput between the L4S and Classic queues. That is, L4S relies on the source's closed-loop response to feedback, not any open-loop obligation of each source to keep within a traffic contract. With L4S, any enforcement of per-flow throughput (whether open-loop or closed) is set aside as a separate issue that may or may not be addressed by separate mechanisms, dependent on policy.

Per bottleneck vs. per domain: L4S can be independently and incrementally deployed at certain bottlenecks. In contrast a Diffserv system is domain-based consisting of the per-hop behaviour of interior nodes and the traffic conditioning behaviour of boundary nodes, which have to be deployed as a coordinated whole.

Degree of multiplexing: Diffserv components such as traffic conditioning are less applicable in access networks where statistical multiplexing is low, whereas L4S was initially designed for access networks, but is also applicable at larger pinch-points (e.g. public peerings).

### 3. Low Latency Diffserv Classes within a DualQ Bandwidth Pool

The experimental Dual-Queue Coupled AQM [I-D.ietf-tsvwg-aqm-dualq-coupled] consists of a pair of queues. One provides a low latency low loss service but both have full access to

the same pool of bandwidth. When Diffserv was defined no mechanism like this was available that could provide low latency without also requiring bandwidth controls. All Diffserv's mechanisms for low latency and low loss use some form of priority over bandwidth, then apply a bandwidth constraint to prevent the lower priority traffic from being starved.

This Diffserv bandwidth constraint has a flip side - it can also provide a bandwidth assurance. However, in turn, bandwidth assurance has both positive and negative aspects. It certainly prevents other traffic encroaching on the bandwidth of the low latency class, but it also carves off a partition within which low latency sessions are more prone to encroach on each other.

The DualQ offers an alternative where low latency traffic can access the whole pool of bandwidth (in effect, the largest possible bandwidth constraint). This is expected to be preferred by many network operators and users who would rather not set a bandwidth limit for their low latency traffic - particularly at links in access networks where the very low level of flow multiplexing makes the bandwidth shares of different traffic classes nearly impossible to predict. Nonetheless, if a bandwidth partition is required for bandwidth assurance purposes, it can still be provided separately (see Section 4).

The DualQ classifies packets with the ECN field set to ECT(1) or CE into the low latency low loss (L) queue. The L queue maintains a low latency low loss service primarily because an L4S source paces its packets and is linearly responsive to ECN markings, which earns it the right to set the ECT(1) codepoint [I-D.ietf-tsvwg-ecn-l4s-id] [RFC8311].

Nonetheless, a low level of non-L4S traffic can share the L queue without compromising the low latency and low loss of the service. Certain existing Diffserv classes are already intended as low latency and low loss services. An operator could use the DualQ instead of traditional Diffserv queues to give a few of these classes the benefit of low latency and access to the whole pool of bandwidth.

However, that would only be safe for those Diffserv service classes that would not risk ruining the low latency of the service. Therefore, an operator must take care to only classify a Diffserv traffic class into the L queue if it is expected to send smoothly without multi-packet bursts. Below we give examples of classes that should (and should not) be safe to mix into the L queue.

Table 1 lists the Diffserv service classes that have been allocated global use Diffserv codepoints (DSCPs) from Pool 1. They are



described in RFC 4594 [RFC4594] and its updates ([RFC5865] and [I-D.ietf-tsvwg-le-phb] so far). An operator that only deploys a DualQ [I-D.ietf-tsvwg-aqm-dualq-coupled] but not the relevant Diffserv PHBs could classify those with an 'L' in the 'Coupled Queue' column (or local use DSCPs with similar characteristics) into its L queue, irrespective of the setting of the ECN field.

Service Class Name	DSCP Name	DSCP	AQM?	Coupled Queue
Network Control{1}	CS7	111000	Y & N	L if L4S
Network Control	CS6	110000	Y & N	L if L4S
OAM	CS2	010000	Y & N	L if L4S
Signalling	CS5	101000	N	L if L4S{2}
Telephony	EF	101110	N	L
RFC 5865	Voice-Admit	101100	N	L{3}
R-T Interactive	CS4	100000	N	L if L4S{4}
MM Conferencing	AF4n	100nn0	Y	L if L4S
Broadcast Video	CS3	011000	N	L if L4S{4}
MM Streaming	AF3n	011nn0	Y	L if L4S
Low Latency Data	AF2n	010nn0	Y	L if L4S
High Thru'put Data	AF1n	001nn0	Y	L if L4S{5}
Standard	BE/DF/CS0	000000	Y	L if L4S
Low Priority Data	LE{6}	000001	Y	L if L4S{7}

Some service class names have been abbreviated to fit. Abbreviations are expanded in RFC 4594 or its updates. For the assured forwarding (AF) DSCP names, the digit 'n' represents 1, 2 or 3 and the corresponding binary digits 'nn' in the DSCP value represent 01, 10 or 11. The 'Coupled Queue' column is explained in the text.

Table 1: Mapping of RFC4594 Diffserv Service Classes in a Coupled AQM

Notes for Table 1:

- {1}: Reserved by RFC 2474 [RFC2474].
- {2}: Superficially, CS5 is a candidate for classification into the L queue irrespective of its ECN field, given application signalling is bursty but usually lightweight. However, at least one major equipment vendor uses CS5 by default to indicate unresponsive broadcast video traffic (to which RFC 4594 allocates CS3).
- {3}: Voice-Admit [RFC5865] could be given priority over Expedited Forwarding (EF) [RFC3246].

- {4}: The Real-Time Interactive and Broadcast Video service classes (or any equivalent local-use classes) are intended for inelastic traffic. Therefore they would not be expected to mark themselves as ECN-capable. If they did they would be claiming to be elastic and therefore eligible for classification into the L queue (subject to any policing). These classes should not be classified into the L queue on the basis of DSCP alone, because high bandwidth unresponsive traffic with potentially variable rate is not compatible with the L4S service.
- {5}: High Throughput Data (or any equivalent local-use class) might use the L4S service because of its support for scalable congestion control.
- {6}: [I-D.ietf-tsvwg-le-phb] updates RFC 4594 to deprecate using CS1 for Lower Effort (LE).
- {7}: If a packet is marked LE and ECT(1) and the operator has solely provided a DualQ, this recommends that the packet is classified into the L queue. This could result in LE traffic competing for bandwidth with other classes of traffic in the L queue, but at least it should not harm the latency of other traffic. This is because the ECT(1) marking means the source "MUST" use a scalable congestion control [I-D.ietf-tsvwg-ecn-l4s-id], but the LE marking only means it "SHOULD" use an LBE congestion control [I-D.ietf-tsvwg-le-phb].

Those classes with an 'L' in the 'DualQ-Coupled' column would not be expected to have the ECT(1) codepoint set because they are generally unresponsive to congestion. Nonetheless, they could coexist in the same queue as L4S traffic because traffic in all of these classes is expected to arrive smoothly, not in bursts of more than a few packets. Therefore an operator could configure a DualQ Coupled AQM to classify such packets into the L queue solely based on their DSCP, irrespective of their ECN codepoint [I-D.ietf-tsvwg-ecn-l4s-id].

Otherwise, [I-D.ietf-tsvwg-ecn-l4s-id] requires that any other DSCP has no effect on classification into the L queue. Thus a packet of any other DSCP will not be classified into the L queue unless it carries an ECT(1) or CE codepoint in the ECN field. This is shown as 'L if L4S' in the 'DualQ-Coupled' column of Table 1.

#### 4. DualQ Bandwidth Pool within a Hierarchy of (Diffserv) Bandwidth Queues

The DualQ Coupled AQM offers an L queue that provides low latency low loss service but it pools bandwidth with the Classic (C) service as if they shared a single FIFO. As explained earlier, unlike previous Diffserv low latency mechanisms, the L queue can offer low latency without needing to limit its bandwidth.

Typically the DualQ will be able to use all the bandwidth available to a customer site, e.g. a household, a campus or a mobile node, as a single pool. However, this section considers scenarios where the network operator might want to carve off a fraction of a site's bandwidth for other purposes, for instance:

1. to ensure that a particularly demanding application (e.g. a virtual reality session) survives even if excess traffic overloads the remainder of the site's bandwidth;
2. to give guaranteed low latency to a particular application (e.g. industrial process control), if the statistically assured low latency of the L queue is insufficiently stable;
3. to provide a bandwidth scavenger service that will have no effect on any other applications at the site, but will scavenge any unused bandwidth, for instance to transfer backups or large data sets.

In all cases, it is assumed that the DualQ has to be able to borrow back any of the carved off bandwidth that is unused by the other service.

The following three subsections present solutions for each of the above scenarios. Depending on the reader's viewpoint, each scenario can be seen as:

- o either taking a queue within an existing Diffserv hierarchy and splitting it into L4S and Classic queues;
- o or building a queuing hierarchy around a pre-existing dual L4S/Classic queue.

In each case, the DualQ remains as an indivisible 'atomic' component as if it were a single queue with a single pool of bandwidth (but that can either be used for low latency or classic service).

The three examples represent the three main ways that this queue-like 'atom' can be included in a hierarchy of other queues. Without loss

of generality only one other queue complements the DualQ in each case, but it would be straightforward to extend the examples with more queues.

Although these examples are framed in the context of IP and Diffserv, similar queuing hierarchies could be constructed at a lower layer, as long as it supported a similar capability to ECN and a similar Traffic Class identifier to Diffserv.

#### 4.1. DualQ Complemented by an Assured Bandwidth Service

Figure 1 shows a DualQ complemented by an additional queue to add a bandwidth assured service. It is assumed that the operator classifies certain packets into the assured bandwidth queue, perhaps by class of service, source address or 5-tuple flow ID.

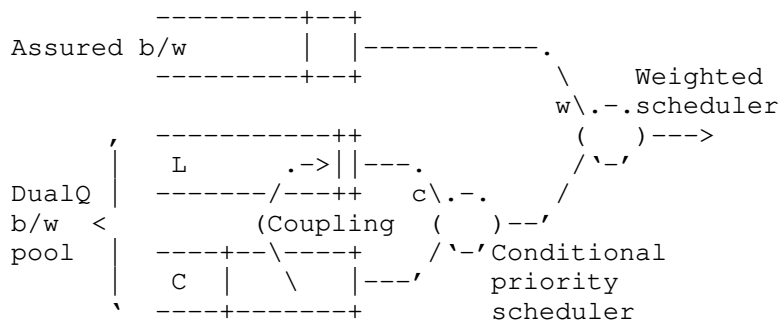


Figure 1: How to Complement a DualQ with an Assured Bandwidth Service

The DualQ is used as if it were an indivisible 'atomic' component, unchanged from its original description in [I-D.ietf-tsvwg-aqm-dualq-coupled]:

- o The outputs of the AQMs in the two queues (L and C) are coupled together so that L4S sources leave enough space for C packets so that all 'standard' flows get roughly equal throughput;
- o A scheduler recombines the outputs of the two queues, giving conditional priority to L packets (the condition prevents starvation of the C queue if any L traffic misbehaves).

A weighted scheduler, e.g. weighted round robin (WRR), is used to combine the outputs of the assured bandwidth queue and the DualQ. It is configured with weight w for the assured bandwidth queue. Then, packets requesting assured bandwidth will have priority access to fraction w of the link capacity. However, whenever the assured

bandwidth queue is idle or under-utilized, the DualQ can borrow the balance of the bandwidth. Likewise the assured bandwidth queue can borrow more than fraction  $w$  if the DualQ under-utilizes its remaining share.

Note that a weighted scheduler such as WRR can be used to implement the conditional priority scheduler between the L and C queues. However, the system will not work as intended if the two weighted schedulers in series are replaced by a single three-input weighted scheduler. This is because, whenever one queue under-uses its weighted share, a weighted scheduler allows the other queue to borrow unused capacity. Whenever traffic is present in the C queue, the coupling ensures that L traffic makes space for it by underutilizing its share of the first scheduler. If the assured bandwidth queue was also served by the same scheduler, the assured bandwidth service would continually borrow the spare capacity left by the L queue that was intended for the C queue.

The assured bandwidth service could itself also support applications using low latency low loss and scalable throughput (L4S). This would be done by serving assured bandwidth traffic with a DualQ (Figure 2) and, as usual, confining legacy queue-building traffic to the C queue.

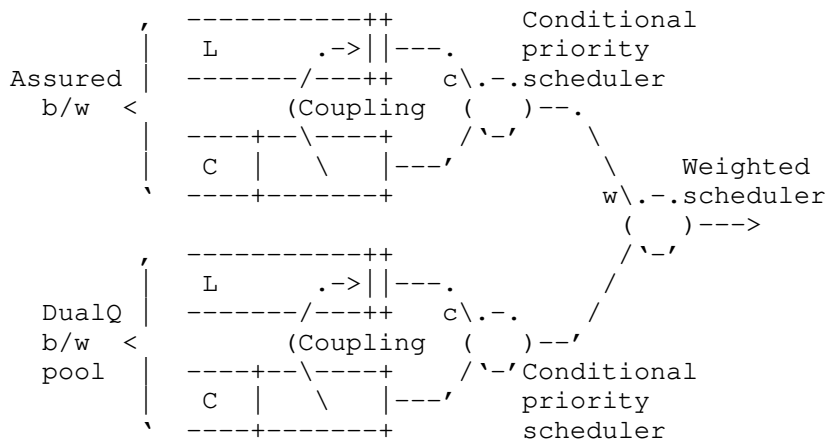


Figure 2: How to Complement a DualQ with an Assured Bandwidth Service that also Supports L4S

The symmetry of Figure 2 reveals that both DualQs actually have assured bandwidth. Nonetheless, the label 'Assured bandwidth' is only really meaningful from a per-application perspective if the

traffic classified into that DualQ is limited to a small number of application sessions at any one time.

#### 4.2. DualQ Complemented by a Guaranteed Low Latency Service

Figure 3 shows a DualQ complemented by an additional queue to add a guaranteed latency service. It is assumed that the operator classifies certain packets into the guaranteed latency queue, perhaps by class of service, source address or 5-tuple flow ID.

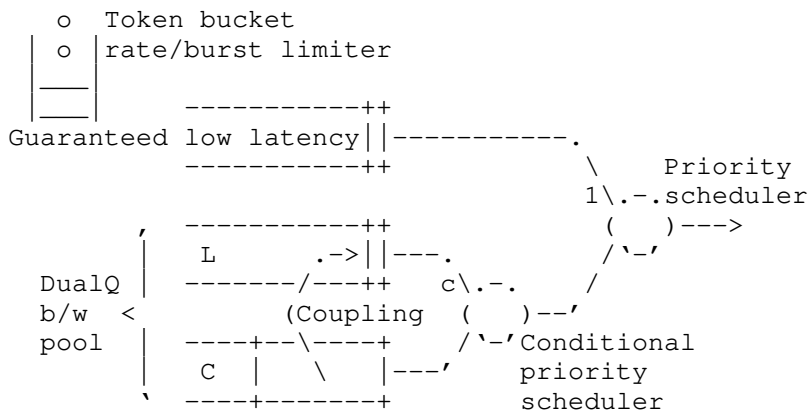


Figure 3: How to Complement a DualQ with a Guaranteed Low Latency Service

As in all the previous example, the DualQ is used as if it were an indivisible 'atomic' component.

A strict priority scheduler is used to combine the outputs of the guaranteed latency queue and the DualQ. Guaranteed low latency traffic is shown as subject to a token bucket that limits rate and tightly limits burst size, which ensures that:

- o Excessive guaranteed latency traffic cannot abuse its priority and cause the DualQ to starve;
- o Guaranteed latency traffic cannot ruin its own latency guarantees – it has to keep to a the traffic contract enforced by the token bucket.

In a traditional Diffserv architecture, the token bucket would be deployed at the ingress network edge, to limit traffic at each entry point. Alternatively, the token bucket could be deployed directly in front of the queue, where it would only limit the total traffic from

all entry points to the network. For an access link into a network, these two alternative would amount to the same thing.

Whenever the guaranteed latency queue is idle or under-utilized, the DualQ can borrow the balance of the bandwidth. However, the guaranteed latency queue cannot borrow more than the token bucket allows, even if the DualQ under-utilizes its remaining share.

#### 4.3. DualQ Complemented by a Scavenger Service

Figure 3 shows a DualQ complemented by an additional queue to add a bandwidth scavenger service. It is assumed that the operator classifies certain packets into the scavenger queue, probably by class of service, e.g. the global-use Lower Effort (LE) Diffserv codepoint [I-D.ietf-tsvwg-le-phb].

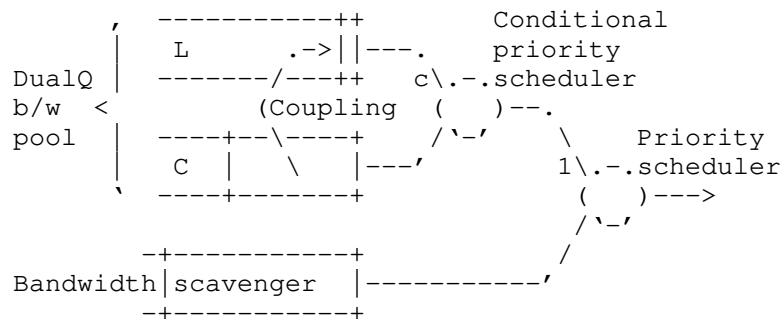


Figure 4: How to Complement a DualQ with a Bandwidth Scavenger Service

As in all the previous example, the DualQ is used as if it were an indivisible 'atomic' component.

A strict priority scheduler is used to combine the outputs of the DualQ and the scavenger service. Section 2 of [I-D.ietf-tsvwg-le-phb] suggests alternative mechanisms.

Whenever the DualQ is idle or under-utilized, the scavenger service can borrow the balance of the bandwidth. In contrast to the previous guaranteed latency example, no rate limiter is needed on the DualQ because, by definition, the scavenger service is expected to starve if the higher priority service is using all the capacity.

## 5. Coupling More than Two AQMs within a Bandwidth Pool

The Diffserv Assured Forwarding (AF) classes of service [RFC2597] use an AQM with differently weighted outputs, e.g. WRED, to provide weighted congestion feedback to the transport layer. Flows classified to use a higher weight AQM each take more of the available capacity, because the weighted AQM has fooled their congestion controller into detecting that the bottleneck is more lightly loaded.

A similar mechanism can be used to add throughput differentiation to either or both of the queues within a DualQ. Figure 5 illustrates an example with an AQM offering three weights within the L queue, where L1 gets the highest throughput per flow. It would be a matter of operator policy to choose which of the three L4S AQMs the Classic AQM would couple to. If it were coupled to L3, then C and L3 flows would get roughly equal throughput, while L2 and L1 flows would get more.

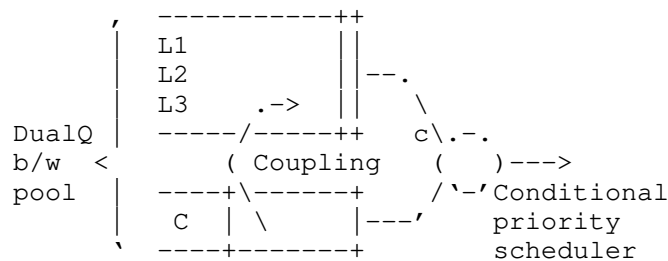


Figure 5: Coupling the Classic AQM to Multiple L4S AQMs

Note: this structure seems straightforward to implement, but the authors are not aware of any implementation or evaluation of AQMs that are both weighted and coupled to other AQMs.

## 6. Applicability of Coupled AQM to Global Diffserv PHBs

As has been explained, Diffserv always divides up bandwidth and divides up latency along the same lines as a consequence, whereas the DualQ Coupled AQM solely provides latency separation without bandwidth separation (the idea being that bandwidth separation can be added if needed, using Diffserv mechanisms).

In this draft so far, various queuing structures have been described in terms of the way they separate bandwidth and latency. Operators with existing Diffserv deployments may put the question the other way round and ask whether the DualQ Coupled AQM can be used to isolate low latency traffic within the bandwidth allocated to one of the standardized Diffserv PHBs. For instance:



- o Bandwidth has been allocated to Network Control traffic, but some BGP speakers have been upgraded to a low latency Scalable TCP while others still use Classic TCP. However it's not possible to predict how much bandwidth one or the other needs at any one time. So it would be useful to isolate the low latency BGP and all the control signalling from the delay caused by the legacy BGP speaker, without having to decide how to carve up the Network Control bandwidth.
- o Bandwidth has been allocated to Assured Forwarding (AF) traffic but it all shares the same WRED queue and therefore all suffers the same delay. So it would be useful to isolate the AF traffic that supports low latency congestion control from the rest. However, again, it is not possible to predict how many flows of each type there will be at any one time.

Table 2 lists all the PHBs with standardized global-use DSCPs from [RFC4594] and the right-hand 'Latency Separation?' column identifies all those that could benefit from an unknowable and variable fraction of their traffic being separated between ultra-low and regular delay using a DualQ Coupled AQM. There is no implication that it is sensible to do this in any of the cases; just that it is possible.

For convenience, the 'Mechanism' column also answers the question "How do PHBs for the global-use DSCPs map to the scenarios in this draft?"

Service Class Name	DSCP Name	AQM?	Mechanism	Latency Separation?
Network Control	CS7	Y&N	Figure 1 or Figure 2	Y
Network Control	CS6	Y&N	Figure 1 or Figure 2	Y
OAM	CS2	Y&N	Figure 1 or Figure 2	Y
Signalling	CS5	N	Figure 1	N
Telephony	EF	N	Section 4.2	N
RFC 5865	VA	N	Section 4.2	N
R-T	CS4	N	Figure 2	Y
Interactive MM	AF4n	Y	Section 5	Y
Conferencing	CS3	N	Figure 2	N
Broadcast Video	AF3n	Y	Section 5	Y
MM Streaming	AF2n	Y	Section 5	Y
Low Latency Data	AF1n	Y	Section 5	Y
High Thru'put Data	BE/DF/CS0	Y	Section 3	Y
Standard Low Priority Data	LE	Y	Section 4.3	n/a

Table 2: Applicability of a Coupled AQM to RFC4594 Diffserv PHBs

## 7. Best Practice for Classification and Marking

### 7.1. Never Re-Mark a DSCP

It is not a DualQ's job to alter Diffserv codepoints to attempt to make other downstream AQMs classify selected packets in certain ways. Each DualQ Coupled AQM is independently (but hopefully consistently) configured to select certain DSCPs for classification into the L queue. It never alters the DSCP nor the ECN codepoint (except setting CE to indicate that congestion was experienced) [I-D.ietf-tsvwg-aqm-dualq-coupled].

### 7.2. Classification Order

#### 7.2.1. Classification Order: Problem

The above wide range of possible structures raises the question of which order it would be more efficient for classifier rules to take: DSCP before ECN, ECN before DSCP or some hybrid.

On the one hand, for a structure like that in Figure 1 it would make sense to classify on DSCP first, then ECN. Otherwise, if packets were classified on ECN first, an extra merge stage would be required because the assured bandwidth queue handles all ECN codepoints for a particular DSCP.

On the other hand, for a structure like that in Figure 5 it would make sense to classify on ECN first, then DSCP. Otherwise, again an extra merge stage would be needed, because the C queue handles all DSCPs but only some ECN codepoints.

A hybrid of these two scenarios would be possible, for instance where the L queue in Figure 1 was further broken down into three weighted AQMs, as in Figure 5. In this case, the ideal matching order would be DSCP, ECN, DSCP.

#### 7.2.2. Classification Order: Solutions

Probably the most straightforward solution would be to classify in a single stage over all 8 octets of the IPv6 Traffic Class field or the former IPv4 TOS octet, irrespective of the boundary between the 6-bit DS field and the 2-bit ECN field [RFC3260]. As long as hardware supports this, it will be possible because all the inputs to the queues are at the same level of hierarchy, even though the outputs form a multi-level hierarchy of schedulers in some cases.

Pre-existing classifier hardware might consider the 6-bit and 2-bit fields as separate. Then it would seem most efficient for the order of the classifiers to depend on the structure of the queues being classified (given the structure has to have been designed before the classifiers are designed).

### 8. Policing and Traffic Conditioning

{ToDo: L4S latency policing is discussed in the Security Considerations section of [I-D.ietf-tsvwg-l4s-arch]. This section will compare Diffserv traffic conditioning with L4S latency policing.}

## 9. IANA Considerations

This specification contains no IANA considerations.

## 10. Security Considerations

{ToDo}

## 11. Comments Solicited

Comments and questions are encouraged and very welcome. They can be addressed to the IETF Transport Area working group mailing list <tsvwg@ietf.org>, and/or to the authors.

## 12. Acknowledgements

Thanks to Greg White, David Black, Wes Eddy and Gorrry Fairhurst for their useful discussions prior to this -00 draft.

## 13. References

### 13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

### 13.2. Informative References

[I-D.ietf-tsvwg-aqm-dualq-coupled]  
Schepper, K., Briscoe, B., Bondarenko, O., and I. Tsang, "DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)", draft-ietf-tsvwg-aqm-dualq-coupled-07 (work in progress), October 2018.

[I-D.ietf-tsvwg-ecn-l4s-id]  
Schepper, K. and B. Briscoe, "Identifying Modified Explicit Congestion Notification (ECN) Semantics for Ultra-Low Queuing Delay (L4S)", draft-ietf-tsvwg-ecn-l4s-id-03 (work in progress), July 2018.

[I-D.ietf-tsvwg-l4s-arch]  
Briscoe, B., Schepper, K., and M. Bagnulo, "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture", draft-ietf-tsvwg-l4s-arch-03 (work in progress), October 2018.

- [I-D.ietf-tsvwg-le-phb]  
Bless, R., "A Lower Effort Per-Hop Behavior (LE PHB)",  
draft-ietf-tsvwg-le-phb-06 (work in progress), October  
2018.
- [RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black,  
"Definition of the Differentiated Services Field (DS  
Field) in the IPv4 and IPv6 Headers", RFC 2474,  
DOI 10.17487/RFC2474, December 1998,  
<<https://www.rfc-editor.org/info/rfc2474>>.
- [RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z.,  
and W. Weiss, "An Architecture for Differentiated  
Services", RFC 2475, DOI 10.17487/RFC2475, December 1998,  
<<https://www.rfc-editor.org/info/rfc2475>>.
- [RFC2597] Heinanen, J., Baker, F., Weiss, W., and J. Wroclawski,  
"Assured Forwarding PHB Group", RFC 2597,  
DOI 10.17487/RFC2597, June 1999,  
<<https://www.rfc-editor.org/info/rfc2597>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition  
of Explicit Congestion Notification (ECN) to IP",  
RFC 3168, DOI 10.17487/RFC3168, September 2001,  
<<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC3246] Davie, B., Charny, A., Bennet, J., Benson, K., Le Boudec,  
J., Courtney, W., Davari, S., Firoiu, V., and D.  
Stiliadis, "An Expedited Forwarding PHB (Per-Hop  
Behavior)", RFC 3246, DOI 10.17487/RFC3246, March 2002,  
<<https://www.rfc-editor.org/info/rfc3246>>.
- [RFC3260] Grossman, D., "New Terminology and Clarifications for  
Diffserv", RFC 3260, DOI 10.17487/RFC3260, April 2002,  
<<https://www.rfc-editor.org/info/rfc3260>>.
- [RFC4594] Babiarz, J., Chan, K., and F. Baker, "Configuration  
Guidelines for DiffServ Service Classes", RFC 4594,  
DOI 10.17487/RFC4594, August 2006,  
<<https://www.rfc-editor.org/info/rfc4594>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion  
Control", RFC 5681, DOI 10.17487/RFC5681, September 2009,  
<<https://www.rfc-editor.org/info/rfc5681>>.

- [RFC5865] Baker, F., Polk, J., and M. Dolly, "A Differentiated Services Code Point (DSCP) for Capacity-Admitted Traffic", RFC 5865, DOI 10.17487/RFC5865, May 2010, <<https://www.rfc-editor.org/info/rfc5865>>.
- [RFC8257] Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <<https://www.rfc-editor.org/info/rfc8257>>.
- [RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <<https://www.rfc-editor.org/info/rfc8311>>.

#### Appendix A. Open Issues

- o The Abstract promises "in which cases one can stand alone without needing the other". That's in the text, but not highlighted as such.
- o Document Roadmap TBA
- o Mapping to 802.11 user priorities (or LTE QCI)s? Not strictly within the scope, but perhaps desirable to add, or at least to mention how L4S (experimental) would affect RFC8325 which gives (standards track) mappings between Diffserv and 802.11.
- o Identify L4S-friendly rate policers
- o Comparison between L4S policing and Diffserv traffic conditioning is TBA
- o Security Considerations are TBA (largely depends on the previous bullet)

#### Author's Address

Bob Briscoe  
CableLabs  
UK

Email: [ietf@bobbriscoe.net](mailto:ietf@bobbriscoe.net)  
URI: <http://bobbriscoe.net/>

INTERNET-DRAFT  
Intended status: Proposed Standard

Donald Eastlake  
Huawei  
Bob Briscoe  
Independent  
Andrew Malis  
Huawei  
February 5, 2019

Expires: August 4, 2019

Explicit Congestion Notification (ECN) and Congestion Feedback  
Using the Network Service Header (NSH)  
<draft-eastlake-sfc-nsh-ecn-support-03.txt>

Abstract

Explicit congestion notification (ECN) allows a forwarding element to notify downstream devices of the onset of congestion without having to drop packets. Coupled with a means to feed back information about congestion to upstream nodes, this can improve network efficiency through better congestion control, frequently without packet drops. This document specifies ECN and congestion feedback support within a Service Function Chaining (SFC) domain through use of the Network Service Header (NSH, RFC 8300) and IP Flow Information Export (IPFIX, draft-ietf-tsvwg-tunnel-congestion-feedback).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Distribution of this document is unlimited. Comments should be sent to the SFC Working Group mailing list <sfc@ietf.org> or to the authors.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/lid-abstracts.html>. The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

## Table of Contents

1. Introduction.....	3
1.1 NSH Background.....	3
1.2 ECN Background.....	5
1.3 Tunnel Congestion Feedback Background.....	5
1.4 Conventions Used in This Document.....	7
2. The NSH ECN Field.....	8
3. ECN Support in the NSH.....	10
3.1 At The Ingress.....	11
3.2 At Transit Nodes.....	12
3.2.1 At NSH Transit Nodes.....	12
3.2.2 At an SF/Proxy.....	13
3.2.3 At Other Forwarding Nodes.....	13
3.3 At Exit/Egress.....	13
3.4 Conservation of Packets.....	14
4. Tunnel Congestion Feedback Support.....	15
5. IANA Considerations.....	16
6. Security Considerations.....	17
7. Acknowledgements.....	17
Normative References.....	18
Informative References.....	19
Authors' Addresses.....	20



## 1. Introduction

Explicit congestion notification (ECN [RFC3168]) allows a forwarding element to notify downstream devices of the onset of congestion without having to drop packets. Coupled with a means to feed back information about congestion to upstream nodes, this can improve network efficiency through better congestion control, frequently without packet drops. This document specifies ECN and congestion feedback support within a Service Function Chaining (SFC [RFC7665]) domain through use of the Network Service Header (NSH [RFC8300]) and IP Flow Information Export (IPFIX [TunnelCongFeedback]).

It requires that all ingress and egress nodes of the SFC domain implement ECN. While congestion management will be the most effective if all interior nodes of the SFC domain implement ECN, some benefit is obtained even if some interior nodes do not implement ECN. In particular, congestion at any bottleneck where ECN marking is not implemented will be unmanaged.

The subsections below in this section provide background information on NSH, ECN, congestion feedback, and terminology used in this document.

### 1.1 NSH Background

The Service Function Chaining (SFC [RFC7665]) architecture calls for the encapsulation of traffic within a service function chaining domain with a Network Service Header (NSH [RFC8300]) added by the "Classifier" (ingress node) on entry to the domain and the NSH being removed on exit from the domain at the egress node. The NSH is used to control the path of a packet in an SFC domain. The NSH is a natural place, in a domain where traffic is NSH encapsulated, to note congestion, avoiding possible confusion due, for example, to changes in the outer transport header in different parts of the domain.

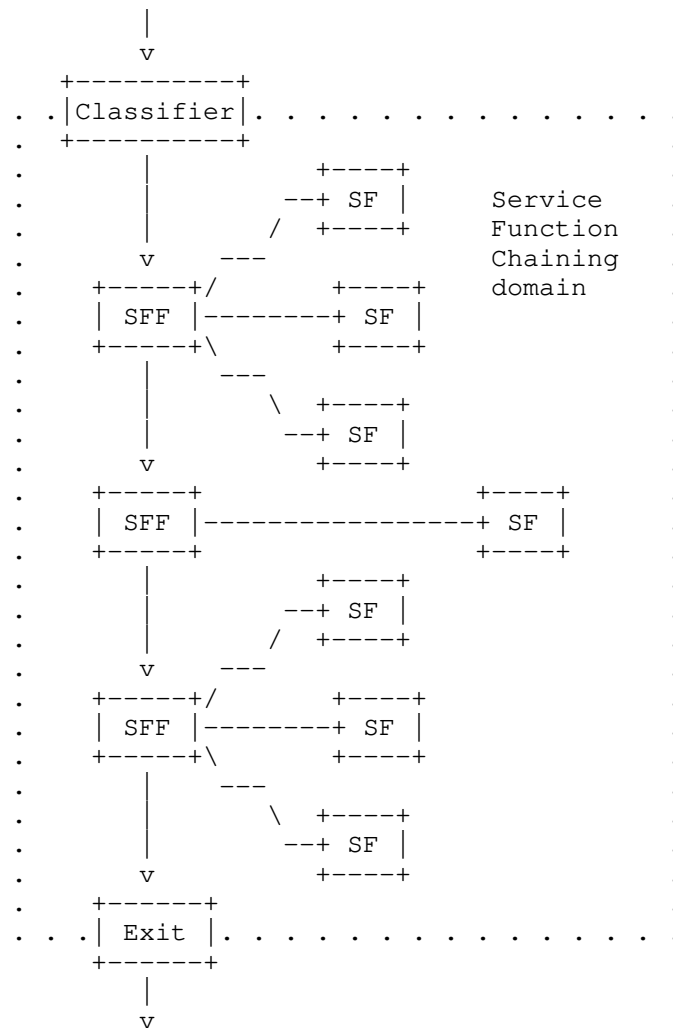


Figure 1. Example SFC Path Forwarding Nodes

Figure 1 shows an SFC domain for the purpose of illustrating the use of NSH. Traffic passes through a sequence of Service Function Forwarders (SFFs) each of which sends the traffic to one or more Service Functions (SFs). Each SF performs some operation on the traffic, for example firewall or Network Address Translation (NAT), and then returns it to the SFF from which it was received.

Logically, during the transit of each SFF, the outer transport header that got the packet to the SFF is stripped, the SFF decides on the next forwarding step, either adding a transport header or, if the SFF is the exit/egress, removing the NSH header. The transport headers

added may be different in different regions of the SFC domain. For example, IP could be used for some SFF-to-SFF communication and MPLS used for other such communication.

## 1.2 ECN Background

Explicit congestion notification (ECN [RFC3168]) allows a forwarding element (such as a router or an Service Function Forwarder (SFF) or Service Function (SF)) to notify downstream devices of the onset of congestion without having to drop packets. This can be used as an element in active queue management (AQM) [RFC7567] to improve network efficiency through better traffic control without packet drops. The forwarding element can explicitly mark some packets in an ECN field instead of dropping the packet. For example, a two-bit field is available for ECN marking in IP headers [RFC3168].

## 1.3 Tunnel Congestion Feedback Background

Tunnel Congestion Feedback [TunnelCongFeedback] is a building block for various congestion mitigation methods. It supports feedback of congestion information from an egress node to an ingress node. Examples of actions that can be taken by an ingress node when it has knowledge of downstream congestion include those listed below. Details of implementing these traffic control methods, beyond those given here, are outside the scope of this document.

Any action by the ingress to reduce congestion needs to allow sufficient time for the end-to-end congestion control loop to respond first, for instance by the ingress taking a smoothed average of the level of congestion signalled by feedback from the tunnel egress.

- (1) Traffic throttling (policing), where the downstream traffic flowing out of the ingress node is limited to reduce or eliminate congestion.
- (2) Upstream congestion feedback, where the ingress node sends messages upstream to or towards the ultimate traffic source, a function that can throttle traffic generation/transmission.
- (3) Traffic re-direction, where the ingress node configures the NSH of some future traffic so that it avoids congested paths. Great care must be taken to avoid (a) significant re-ordering of traffic in flows that it is desirable to keep in order and (b) oscillation/instability in traffic paths due to alternate congestion of previously idle paths and the idling of previously congested paths. For example, it is preferable to classify

traffic into flows of a sufficiently coarse granularity that the flows are long lived and use a stable path per flow sending only newly appearing flows on apparently uncongested paths.

Figure 2 shows an example path from an origin sender to a final receiver passing through an example chain of service functions between the ingress and egress of an SFC domain. The path is also likely to pass through other network nodes outside the SFC domain (not shown). The figure shows typical congestion feedback that would be expected from the final receiver to the origin sender, which controls the load the origin sender applies to all elements on the path. The figure also shows the congestion feedback from the egress to the ingress of the SFC domain that is described in this document, to control or balance load within the SFC domain.

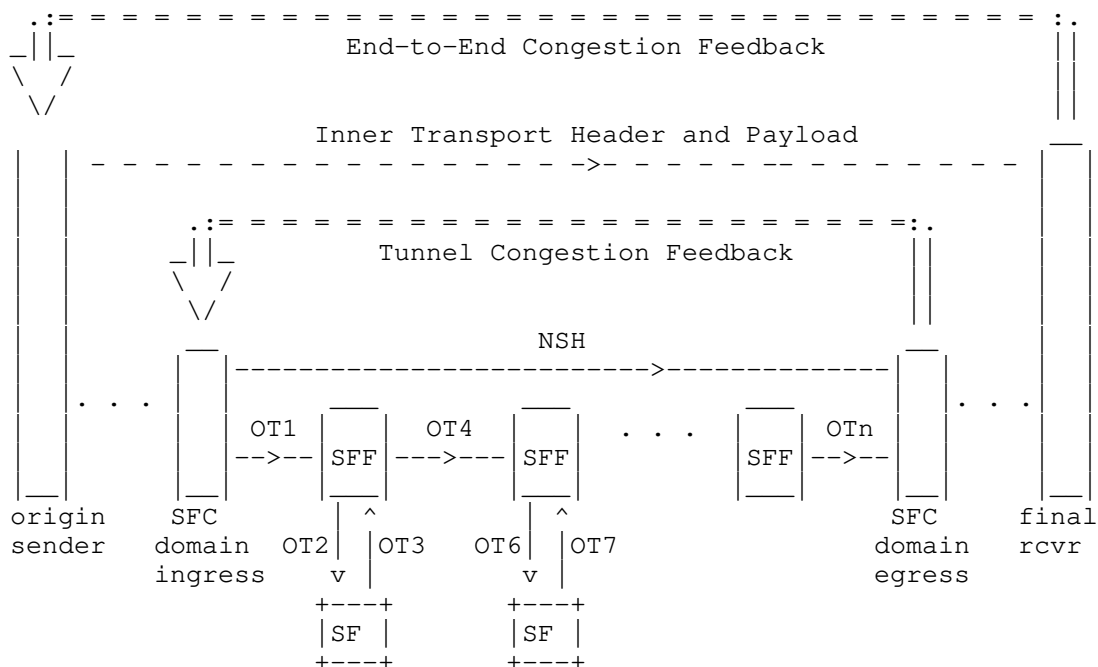


Figure 2: Congestion Feedback across an SFC Domain

SFC Domain congestion feedback in Figure 2 is shown within the context of an end-to-end congestion feedback loop. Also shown is the encapsulated layering of NSH headers within a series of outer transport headers (OT1, OT2, ... OTn).

#### 1.4 Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

##### Acronyms:

AQM - Active Queue Management [RFC7567]

CE - Congestion Experienced [RFC3168]

downstream - The direction from ingress to egress

ECN - Explicit Congestion Notification [RFC3168]

ECT - ECN Capable Transport [RFC3168]

IPFIX - IP Flow Information Export [RFC7011]

Not-ECT - Not ECN-Capable Transport [RFC3168]

NSH - Network Service Header [RFC8300]

SF - Service Function [RFC7665]

SFC - Service Function Chaining [RFC7665]

SFF - Service Function Forwarder [RFC7665] - A type of node that forwards based on the NSH.

TLV - Type Length Value

upstream - The direction from egress to ingress

## 2. The NSH ECN Field

The NSH header is used to encapsulate and control the subsequent path of traffic (see Section 2 of [RFC8300]). The NSH also provides for metadata inclusion, as shown in Figure 3.

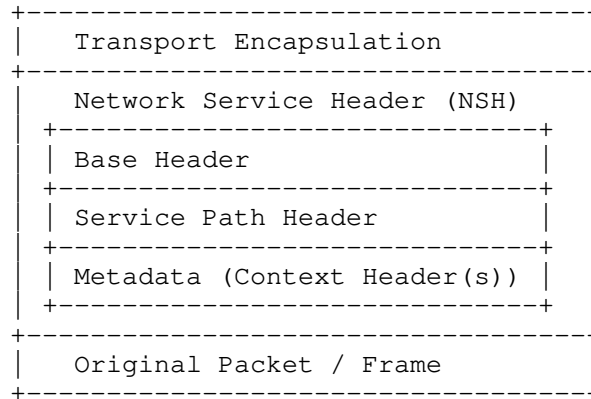


Figure 3. Data Encapsulation with the NSH

Two currently unused bits (indicated by "U") in the NSH Base Header (Section 2.2 of [RFC8300]) are allocated for ECN as shown in Figure 4.

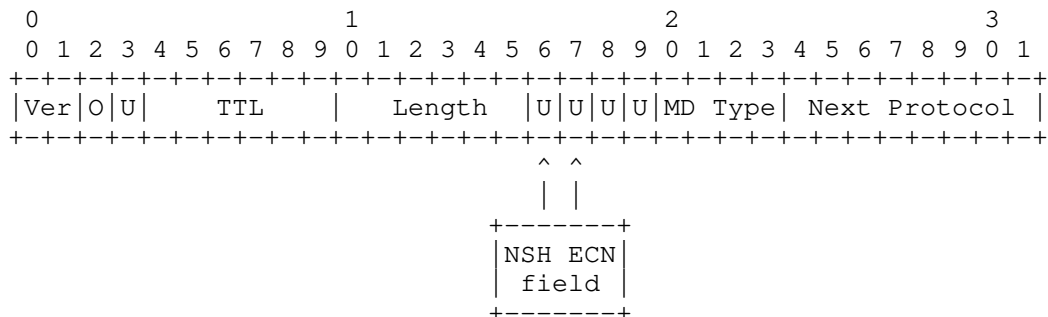


Figure 4: NSH Base Header

Note to RFC Editor: The above figure should be adjusted based on the bits assigned by IANA (see Section 5) and this note deleted.

Table 1 shows the meaning of the code points in the NSH ECN field. These have the same meaning as the ECN field code points in the IPv4 or IPv6 header as defined in [RFC3168].

Binary	Name	Meaning
00	Not-ECT	Not ECN-Capable Transport
01	ECT(1)	ECN-Capable Transport
10	ECT(0)	ECN-Capable Transport
11	CE	Congestion Experienced

Table 1. ECN Field Code Points

### 3. ECN Support in the NSH

This section describes the required behavior to support ECN using the NSH. There are two aspects to ECN support:

1. ECN propagation during encapsulation or decapsulation
2. ECN marking during congestion at bottlenecks.

While this section covers all combinations of ECN-aware and not ECN-aware, it is expected that in most cases the NSH domain will be uniform so that, if this document is applicable, all SFFs will support ECN; however, some legacy SFs might not support ECN.

#### ECN Propagation:

The specification of ECN tunneling [RFC6040] explains that an ingress must not propagate ECN support into an encapsulating header unless the egress supports correct onward propagation of the ECN field during decapsulation. We define Compliant ECN Decapsulation here as decapsulation compliant with either [RFC6040] or an earlier compatible equivalent ([RFC4301], or full functionality mode of [RFC3168]).

The procedures in Section 3.2.1 ensure that each ingress of the large number of possible transport links within the SFC domain does not propagate ECN support into the encapsulating outer transport header unless the corresponding egress of that link supports Compliant ECN Decapsulation.

Section 3.3 requires that all the egress nodes of the SFC domain support Compliant ECN Decapsulation in conjunction with tunnel congestion feedback, otherwise the scheme in this document will not work.

#### ECN Marking:

At transit nodes the marking behavior specified in 3.2.1 is recommended and if not implemented at such transit nodes, there may be unmanaged congestion.

Detection of congestion will be most effective if ECN marking is supported by all potential bottlenecks inside the domain in which NSH is being used to route traffic as well as at the ingress and egress. Nodes that do not support ECN marking, or that support AQM but not ECN, will naturally use drop to relieve congestion. The gap in the end-to-end packet sequence will be detected as congestion by the final receiving endpoint, but not by the NSH egress (see Figure 2).



### 3.1 At The Ingress

When the ingress/Classifier encapsulates an incoming IP packet with an NSH, it MUST set the NSH ECN field using the "Normal mode" specified in [RFC6040] (i.e., copied from the incoming IP header).

Then, if the resulting NSH ECN field is Not-ECT, the ingress SHOULD set it to ECT(0). This indicates that, even though the end-to-end transport is not ECN-capable, the egress and ingress of the SFC domain are acting as an ECN-capable transport. This approach will inherently support all known variants of ECN, including the experimental L4S capability [RFC8311], [ecnL4S].

Packets arriving at the ingress might not use IP. If the protocol of arriving packets supports an ECN field similar to IP, the procedures for IP packets can be used. If arriving packets do not support an ECN field similar to IP, they MUST be treated as if they are Not-ECT IP packets.

Then, as the NSH encapsulated packet is further encapsulated with a transport header, if ECN marking is available for that transport (as it is for IP [RFC3168] and MPLS [RFC5129]), the ECN field of the transport header MUST be set using the "Normal mode" specified in [RFC6040] (i.e., copied from the NSH ECN field).

A summary of these normative steps is given in Table 2.

Incoming Header (also equal to departing Inner Header)	Departing NSH and Outer Headers
Not-ECT	ECT(0)
ECT(0)	ECT(0)
ECT(1)	ECT(1)
CE	CE

Table 2. Setting of ECN fields by an ingress/Classifier

The requirements in this section apply to all ingress nodes for the domain in which NSH is being used to route traffic.

### 3.2 At Transit Nodes

This section described behavior at nodes that forward based on the NSH such as SFF and other forwarding nodes such as IP routers. Figure 5 shows a packet on the wire between forwarding nodes.

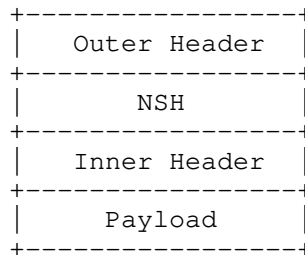


Figure 5. Packet in Transit

#### 3.2.1 At NSH Transit Nodes

When a packet is received at an NSH based forwarding node N1, such as an SFF, the outer transport encapsulation is removed and its ECN marking SHOULD be combined into the NSH ECN marking as specified in [RFC6040]. If this is not done, any congestion encountered at non-NSH transit nodes between N1 and the next upstream NSH based forwarding node will be lost and not transmitted downstream.

The NSH forwarding node SHOULD use a recognized AQM algorithm [RFC7567] to detect congestion. If the NSH ECN field indicates ECT, it will probabilistically set the NSH ECN field to the Congestion Experienced (CE) value or, in cases of extreme congestion, drop the packet.

When the NSH encapsulated packet is further encapsulated for transmission to the next SFF or SF, ECN marking behavior depends on whether or not the node that will decapsulate the outer header supports Compliant ECN Decapsulation (see Section 3). If it does, then the ingress node propagates the NSH ECN field to this outer encapsulation using the "Normal Mode" of ECN encapsulation [RFC6040] (it copies the ECN field). If it does not, then the ingress MUST clear ECN in the outer encapsulation to non-ECT (the "Compatibility Mode" of [RFC6040]).

### 3.2.2 At an SF/Proxy

If the SF is NSH and ECN-aware, the processing is essentially the same at the SF as at an SFF as discussed in Section 3.2.1.

If the SF is NSH-aware but not ECN-aware, then the SFF transmitting the packet to the SF will use Compatibility Mode. Congestion encountered in the SFF to SF and SF to SFF paths will be unmanaged.

If the SF is not NSH-aware, then an NSH proxy will be between the SFF and the SF to avoid exposure of the NSH at the SF that does not understand NSHs. This is described in Section 4.6 of [RFC7665]. The SF and proxy together look to the SFF like an NSH-aware SF. The behavior at the proxy and SF in this case is as below:

If such a proxy is not ECN-aware then congestion in the entire path from SFF to proxy to SF back to proxy to SFF will be unmanaged.

If the proxy is ECN-aware the proxy uses an AQM to indicate congestion in the proxy itself in the NSH that it returns to the SFF. The outer header used for the proxy to SF path uses Normal Mode. The outer head used for the proxy return to SFF path uses Normal Mode based copying the NSH ECN field to the outer header. Thus congestion in the proxy will be managed. Congestion in the SF will be managed only if the SF is ECN-aware implementing an AQM.

### 3.2.3 At Other Forwarding Nodes

Other forwarding nodes, that is non-NSH forwarding nodes between NSH forwarding nodes, such as IP routers, might also be potential bottlenecks. If so, they SHOULD implement an AQM algorithm to update the ECN marking in the outer transport header as specified in [RFC3168].

### 3.3 At Exit/Egress

First, any actions are taken based on Congestion Experienced such as forwarding statistics back to the ingress (see Section 4). If the packet being carried inside the NSH is IP, when the NSH is removed the NSH ECN field MUST be combined with IP ECN field as specified in Table 3 that was extracted from [RFC6040]. This requirement applies to all egress nodes for the domain in which NSH is being used to route traffic.

Arriving Inner Header	Arriving Outer Header			
	Not-ECT	ECT (0)	ECT (1)	CE
Not-ECT	Not-ECT	Not-ECT	Not-ECT	<drop>
ECT (0)	ECT (0)	ECT (0)	ECT (0)	CE
ECT (1)	ECT (1)	ECT (1)	ECT (1)	CE
CE	CE	CE	CE	CE

Table 3. Exit ECN Fields Merger

All the egress nodes of the SFC domain MUST support Compliant ECN Decapsulation as specified in this section. If this is not the case, the scheme described in this document will not work, and cannot be used.

### 3.4 Conservation of Packets

The SFC specification permits an SF to absorb packets and to generate new packets as well as to process and forward the packets it receives. Such actions might appear to be packet loss due to congestion or might mask the loss of packets by generating additional packets.

The tunnel congestion feedback approach [TunnelCongFeedback] detects loss by counting payload bytes in at the ingress and counting them out at the egress. This does not work unless nodes conserve the amount of payload bytes. Therefore, it will not be possible to detect loss using this technique if they are not conserved.

Nonetheless, if a bottleneck supports ECN marking, it will be possible to detect the very high level of CE markings that are associated with congestion that is so excessive that it leads to loss. However, it will not be possible for the tunnel congestion feedback approach to detect any congestion, whether slight or severe, if it occurs at a bottleneck that does not support ECN marking.

#### 4. Tunnel Congestion Feedback Support

The collection and storage of congestion information may be useful for later analysis but, unless it can be fed back to a point which can take action to reduce congestion, it will not be useful in real time. Such congestion feedback to the ingress enables it to take actions such as those listed in Section 1.3.

IP Flow Information Export (IPFIX [RFC7011]) provides a standard for communicating traffic flow statistics. As extended by [TunnelCongFeedback], IPFIX can be used to determine the extent of congestion between an ingress and egress.

IPFIX recommends use of SCTP [RFC4960] in partial reliability mode. This mode allows loss of some packets, which is tolerable because IPFIX communicates cumulative statistics. IPFIX over SCTP SHOULD be used directly where there is IP connectivity between the ingress and egress; however, there might be different transport protocols or address spaces used in different regions of an SFC domain that make such direct IP connectivity problematic. The NSH provides the general method of routing of traffic within such domain so the IPFIX over SCTP over IP traffic should be encapsulated in NSH when necessary.

## 5. IANA Considerations

IANA is requested to assign two contiguous bits in the NSH Base Header Bits registry for ECN (bits 16 and 17 suggested) and note this assignment as follows:

Bit	Description	Reference
-----	-----	-----
tbd(16-17)	NSH ECN	[this document]

## 6. Security Considerations

For general NSH security considerations, see [RFC8300].

For security considerations concerning tampering with ECN signaling, see [RFC3168]. For security considerations concerning ECN encapsulation, see [RFC6040].

For general IPFIX security considerations, see [RFC7011]. If deployed in an untrusted environment, the signaling traffic between ingress and egress can be protected utilizing the security mechanisms provided by IPFIX (see section 11 in RFC7011).

The solution in this document does not introduce any greater potential to invade privacy than would have been possible without the solution.

## 7. Acknowledgements

The authors wish to thank the following for their comments and suggestion:

Joel Halpern, Tal Mizrahi, Xinpeng Wei

## Normative References

- [RFC2119] - Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3168] - Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<http://www.rfc-editor.org/info/rfc3168>>.
- [RFC5129] - Davie, B., Briscoe, B., and J. Tay, "Explicit Congestion Marking in MPLS", RFC 5129, DOI 10.17487/RFC5129, January 2008, <<https://www.rfc-editor.org/info/rfc5129>>.
- [RFC6040] - Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <<http://www.rfc-editor.org/info/rfc6040>>.
- [RFC7011] - Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, DOI 10.17487/RFC7011, September 2013, <<https://www.rfc-editor.org/info/rfc7011>>.
- [RFC7567] - Baker, F., Ed., and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <<http://www.rfc-editor.org/info/rfc7567>>.
- [RFC8174] - Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<http://www.rfc-editor.org/info/rfc8174>>.
- [RFC8300] - Quinn, P., Ed., Elzur, U., Ed., and C. Pignataro, Ed., "Network Service Header (NSH)", RFC 8300, DOI 10.17487/RFC8300, January 2018, <<https://www.rfc-editor.org/info/rfc8300>>.
- [TunnelCongFeedback] - Wei, X., Zhu, L., and L. Deng, "Tunnel Congestion Feedback", draft-ietf-tsvwg-tunnel-congestion-feedback, work in progress.



## Informative References

- [RFC4301] - Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<https://www.rfc-editor.org/info/rfc4301>>.
- [RFC4960] - Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.
- [RFC7665] - Halpern, J., Ed., and C. Pignataro, Ed., "Service Function Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <<https://www.rfc-editor.org/info/rfc7665>>.
- [RFC8311] - Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <<https://www.rfc-editor.org/info/rfc8311>>.
- [ecnL4S] - De Schepper, K., and B. Briscoe, "Identifying Modified Explicit Congestion Notification (ECN) Semantics for Ultra-Low Queuing Delay (L4S)", draft-ietf-tsvwg-ecn-l4s-id, work in progress.

Authors' Addresses

Donald E. Eastlake, 3rd  
Huawei Technologies  
1424 Pro Shop Court  
Davenport, FL 33896 USA

Tel: +1-508-333-2270  
Email: d3e3e3@gmail.com

Bob Briscoe  
Independent  
UK

Email: ietf@bobbriscoe.net  
URI: <http://bobbriscoe.net/>

Andrew G. Malis  
Huawei Technologies

Email: agmalis@gmail.com

## Copyright and IPR Provisions

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License. The definitive version of an IETF Document is that published by, or under the auspices of, the IETF. Versions of IETF Documents that are published by third parties, including those that are translated into other languages, should not be considered to be definitive versions of IETF Documents. The definitive version of these Legal Provisions is that published by, or under the auspices of, the IETF. Versions of these Legal Provisions that are published by third parties, including those that are translated into other languages, should not be considered to be definitive versions of these Legal Provisions. For the avoidance of doubt, each Contributor to the IETF Standards Process licenses each Contribution that he or she makes as part of the IETF Standards Process to the IETF Trust pursuant to the provisions of RFC 5378. No language to the contrary, or terms, conditions or rights that differ from or are inconsistent with the rights and licenses granted under RFC 5378, shall have any effect and shall be null and void, whether published or posted by such Contributor, or included with or in such Contribution.



Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: April 22, 2019

G. Fairhurst  
T. Jones  
R. Zullo  
University of Aberdeen  
October 19, 2018

Checksum Compensation Options for UDP Options  
draft-fairhurst-udp-options-cco-00.txt

Abstract

This document describes a robust method for calculating checksums for use with UDP Options. The new method proposes an alternative checksum calculation for coverage of the option space. This is based on the IP checksum calculation, but uses an updated pseudoheader. The new method only checks the option portion of a UDP packet, but creates a checksum that compensates for the range of IP and UDP checksum validation methods that have been deployed, in this way the new method enhances the probability of NAPT traversal for packets that carry UDP-Options.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 22, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Terminology . . . . .	3
3. Middlebox Pathologies . . . . .	3
4. Checksum Compensation Option . . . . .	4
4.1. Calculating the CCO . . . . .	6
4.2. Validating CCO . . . . .	7
4.3. CCO Calculation Examples . . . . .	8
4.4. Interaction with other UDP Options . . . . .	9
5. Acknowledgements . . . . .	9
6. IANA Considerations . . . . .	9
7. Security Considerations . . . . .	10
8. Normative References . . . . .	10
Appendix A. Revision Notes . . . . .	10
Authors' Addresses . . . . .	10

## 1. Introduction

UDP Options [I-D.ietf-tsvwg-udp-options] adds support for transport options in UDP [RFC0768]. When UDP is carried in IP two length fields describe the UDP datagram, the IP transport carries a payload length and the UDP header carries the length of the UDP datagram. In most datagrams currently forwarded by network devices the IP payload length is equal to the UDP length, UDP Options [I-D.ietf-tsvwg-udp-options] creates a surplus area by increasing the IP payload length while not varying the UDP length. Transport Options are then added in this surplus area in the form of a TLV encoded list.

The current specification for UDP permits sending datagrams with surplus data, but are not commonly observed, and many network devices assume that IP payload length is equal to UDP length and have used this value when calculating UDP checksums. This leads to the case where some middlebox devices (e.g. Firewalls, NAT) and some endpoint implementations check or modify the UDP checksum in a way that leads to discard of UDP datagrams that carry UDP options.

This document describes common pathologies of network devices that incorrectly calculate the UDP checksum and proposes a new UDP Option to compensate for incorrect UDP checksum calculation.

## 2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

## 3. Middlebox Pathologies

Middleboxes and network interfaces can compute the UDP Checksum incorrectly in the presence of UDP Options based on the assumption that IP Payload Length and UDP Length coincide (an assumption that was equivalent before UDP Options).

These middleboxes use the IP Payload Length (obtained as IP Total Length - IP Header Length) to fill UDP pseudo-header Length field and also compute the checksum over the all IP Payload bytes.

This can lead to UDP Options packets that carry a correctly calculated checksum to be discarded by end-hosts or by middleboxes along the path.

Figure 1 shows UDP Checksum computation based on UDP Length and based on IP Payload Length and the fields that are different for the two calculation methods.

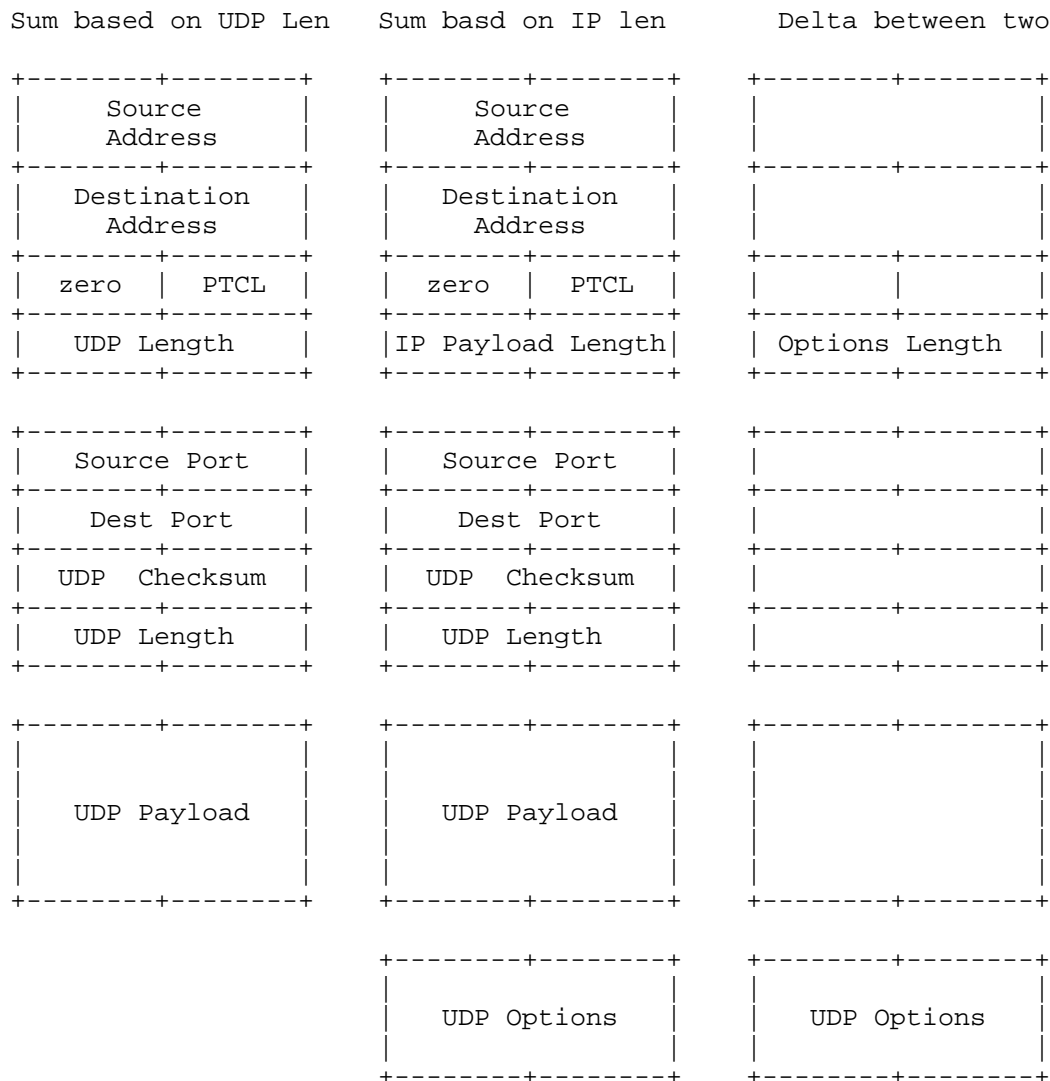


Figure 1: Checksum calculations

#### 4. Checksum Compensation Option

This section introduces the Checksum Compensation Option (CCO), which suggests a new way to calculate the checksum for the option field.

The design of the CCO seeks to increase UDP Options compatibility with middleboxes and other existing network equipment, while at the same time providing error detection on UDP Options area in the same



way that the UDP Checksum provides an integrity check for the UDP Header and UDP Payload.

CCO provides a checksum for UDP Option packets that is compatible with both variants of the checksum computation making the final value of the UDP Checksum computed on the whole IP Payload coincide with the the value that would be correctly computed soley on the UDP Length.

The Checksum Compensation Option (CCO) is the 2 byte one's complement sum of the one's complement sum of all 2 byte words in the UDP Options. Figure 2 describes the format of the CCO. The UDP Options area is divided into 2 byte words based on their alignment with the first byte of UDP packet (and not the first byte of UDP Options). This means that the first and the last byte of UDP Options can not preceded or be followed by another byte: in these cases the unpaired byte must padded respectively on the left and on the right with zero to form a 2 byte word.

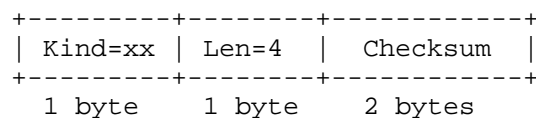


Figure 2: UDP CCO Option Format

[RFC0793] specifies: "The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16 bit word for checksum purposes. The pad is not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros." This method is equivalent to that specified for UDP [RFC0768].

The checksum also covers a 2 byte pseudo header conceptually prefixed to the UDP Options area. This pseudo header contains the length of UDP Options area. (The length also forms a part of the TCP and UDP pseudo field [RFC0793]).

Figure 3 shows the bytes on which CCO is computed and how, when present, the unpaired byte at the start and/or at end of Options area are included in the sum.

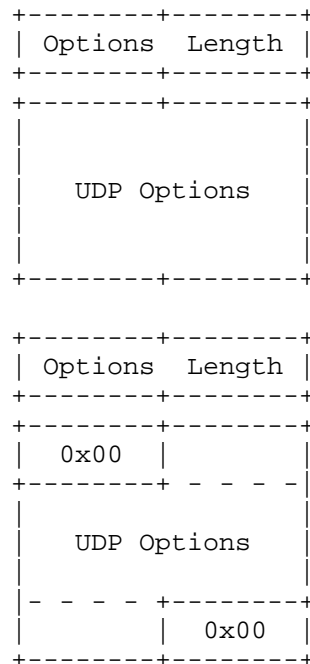


Figure 3: UDP ECHORES Option Format

When this CCO checksum and the UDP Options field are covered by the UDP checksum calculation [RFC0768], the resulting UDP checksum value is numerically the same as when the UDP checksum calculation is calculated over only the UDP Payload. That is, the result returned by both checksum computations Figure 1 coincide.

#### 4.1. Calculating the CCO

The CCO can be present at any position within the Options space, the checksum field of the CCO MUST be aligned on a 2 byte boundary. This condition can be achieved by placing a NOP Option before the CCO in the case the number of bytes preceding the CCO (UDP Payload + UDP Options placed before CCO) is odd (see Figure 4).

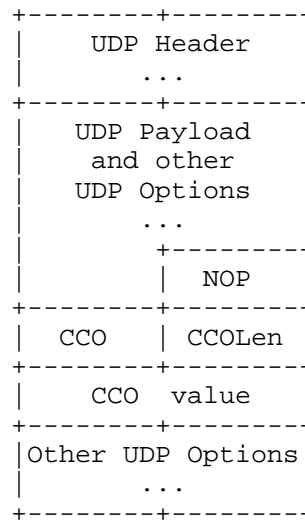


Figure 4: Option Space layout

When calculated in this way, the CCO value is initialized to zero and the checksum is calculated over the UDP Options and the pseudo-header: the one's complement of the result is then stored in the CCO field.

An alternative implementation could be to initialise the CCO field with the size of the UDP Options area (instead of initialising the CCO value to zero and combining with a pseudo header). This produces the same result, but allows the checksum to be performed using solely the UDP Options area.

#### 4.2. Validating CCO

When a UDP packet containing CCO is received the Internet Checksum should be computed on the UDP Options area (2 byte aligned as described in Section 4.3) and the pseudo-header (the length of the received UDP Options), and the Options is valid if the one's complement of the result is zero.

If the option checksum fails, all options MUST be ignored and any trailing surplus data (and Lite data, if used) silently discarded. UDP data that is validated by a correct UDP checksum MUST be delivered to the application layer, even if the UDP option checksum fails.

### 4.3. CCO Calculation Examples

This section provides examples of calculating the Checksum Compensation Option, similar to those presented in [RFC1071].

XXX IANA NOTE: The type of the CCO option has yet too be assigned, and may change. XXX

These examples use 204 (0xCC) as the type for the CCO option

In the first example the UDP Payload length is even and a MSS Option has been already placed in UDP Options area. CCO value is initialized with UDP Options Length (0x0008).

UDP Length:	Even
Preceding UDP Options:	MSS (kind 5, len 4, val 0x5c0)
Following UDP Options:	None
NOP Padding before CCO:	No
Total UDP Options Length:	8
UDP Options bytes 0/1:	0504
UDP Options bytes 2/3:	05c0
UDP Options bytes 4/5:	cc04
UDP Options bytes 6/7:	0008
	----
Sum:	d6d0
CCO:	292f

Figure 5: Checksum calculations

In the second example the UDP Payload length is odd and a MSS Option has been already placed in UDP Options area. The available space for CCO starts at an odd byte (NOP padding before CCO) and also UDP options space starts at odd byte (left zero padding of first byte). CCO value is initialized with UDP Options Length (0x0009).

UDP Length:	Odd
Preceding UDP Options:	MSS (kind 5, len 4, val 0x5c0)
Following UDP Options:	None
NOP Padding before CCO:	Yes
Total UDP Options Length:	9
UDP Options bytes 0:	0005
UDP Options bytes 1/2:	0405
UDP Options bytes 3/4:	c001
UDP Options bytes 5/6:	cc04
UDP Options bytes 7/8:	0009
	----
Sum:	9019
CCO:	6fe6

Figure 6: Checksum calculations

#### 4.4. Interaction with other UDP Options

##### Interaction with other UDP Options

AE: Similarly to what happens with OCS, AE can be computed as if the AE hash and CCO value are zero. CCO value can be computed as if the CCO value is zero and after the AE hash has been computed.

ACS: The CCO has no interference with ACS since an ACS is computed only on UDP Payload bytes (no Header, no Options). The CCO value must be computed after the ACS has already been computed.

LITE: The CCO covers the entire UDP Option area, including any LITE option as formatted after swapping (or relocation) for transmission (or, equivalently, before the swap/relocation after reception). The CCO is computed after LITE swapping/relocation to guarantee the checksum compensation of the packet actually sent.

#### 5. Acknowledgements

This work is partially supported by the European Commission under Horizon 2020 grant agreement no. 688421 Measurement and Architecture for a Middleboxed Internet (MAMI).

#### 6. IANA Considerations

This memo includes no requests to IANA

## 7. Security Considerations

The security considerations for are described in [I-D.ietf-tsvwg-udp-options]. The proposed new method does not change the integrity protection offered by the UDP options method.

## 8. Normative References

- [I-D.ietf-tsvwg-udp-options]  
Touch, J., "Transport Options for UDP", draft-ietf-tsvwg-udp-options-05 (work in progress), July 2018.
- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <<https://www.rfc-editor.org/info/rfc768>>.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC1071] Braden, R., Borman, D., and C. Partridge, "Computing the Internet checksum", RFC 1071, DOI 10.17487/RFC1071, September 1988, <<https://www.rfc-editor.org/info/rfc1071>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

## Appendix A. Revision Notes

Note to RFC-Editor: please remove this entire section prior to publication.

Individual draft -00:

- o Comments and corrections are welcome directly to the authors or via the IETF TSVWG working group mailing list.
- o This update is proposed for WG comments.

Authors' Addresses

Godred Fairhurst  
University of Aberdeen  
School of Engineering  
Fraser Noble Building  
Aberdeen AB24 3UE  
UK

Email: [gorry@erg.abdn.ac.uk](mailto:gorry@erg.abdn.ac.uk)

Tom Jones  
University of Aberdeen  
School of Engineering  
Fraser Noble Building  
Aberdeen AB24 3UE  
UK

Email: [tom@erg.abdn.ac.uk](mailto:tom@erg.abdn.ac.uk)

Raffaele Zullo  
University of Aberdeen  
School of Engineering  
Fraser Noble Building  
Aberdeen AB24 3UE  
UK

Email: [raffaele@erg.abdn.ac.uk](mailto:raffaele@erg.abdn.ac.uk)

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: June 19, 2019

T. Fossati  
Nokia  
G. Fairhurst  
University of Aberdeen  
P. Aranda Gutierrez  
Universidad Carlos III de Madrid  
M. Kuehlewind  
ETH Zurich  
December 16, 2018

A Loss-Latency Trade-off Signal for the Mobile Network  
draft-fossati-tsvwg-lola-00

Abstract

This document proposes a marking scheme for tagging low-latency flows (for example: interactive voice and video, gaming, machine to machine applications) that is safe to use by the mobile network for matching such flows to suitable per-hop behaviors (EPS bearers defined by 3GPP) in its core and radio segments. The suggested scheme re-uses NQB, a DiffServ-based signalling scheme with compatible rate-delay trade-off semantics that has been recently introduced in the context of fixed access to allow differential treatment of non-queue building vs queue building flows.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 19, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.



This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Terminology . . . . .	3
3. DiffServ Code . . . . .	4
4. Marking . . . . .	4
5. Relationship to a Mobile DiffServ Domain . . . . .	4
6. On Remarking and Bleaching . . . . .	4
7. IANA Considerations . . . . .	4
8. Security Considerations . . . . .	4
9. Privacy Considerations . . . . .	5
10. Acknowledgments . . . . .	5
11. References . . . . .	5
11.1. Normative References . . . . .	5
11.2. Informative References . . . . .	6
Authors' Addresses . . . . .	6

## 1. Introduction

Today's mobile networks are configured to bundle all flows to and from the Internet into a single "default" EPS bearer whose buffering characteristics are not compatible with low-latency traffic. The established behaviour is partly rooted in the desire to prioritise operators' voice services over competing over-the-top services. Of late, said business consideration seems to have lost momentum and the incentives might now be aligned towards allowing a more suitable treatment of Internet real-time flows. However, a couple of preconditions need to be satisfied before we can move on from the status quo. First, the real-time flows must be efficiently identified so that they can be quickly assigned to the "right" EPS bearer. This is especially important with the rising popularity of encrypted and multiplexed transports, which has the potential of increasing the cost/accuracy ratio of Multi-Field (MF) based classification over the acceptable threshold. Second, the signal must be such that malicious or badly configured nodes can't abuse it. Today's mobile networks take a rather extreme posture in this respect by actively discarding (remarking or bleaching [Custura]) DiffServ signalling coming from an interconnect. Therefore, the signal must

be modelled in a way that the mobile network can either trust it or, even better, avoid the need of trusting it in the first place. The Rate-Delay trade-off [Podlesny] satisfies the above requirements and has been shown [Fossati] to integrate well with a simplified LTE QoS setup that uses one dedicated low-latency bearer in addition to the default.

This document suggests reusing the Non Queue Building (NQB) signalling protocol described in [I-D.white-tsvwg-nqb] as the method employed by endpoints to mark their real-time flows and by the LTE network to classify and route these flows via a suitable (low-latency) bearer through the LTE core network and E-UTRAN.

## 2. Terminology

- o DPI: Deep Packet Inspection
- o EPS bearer: Evolved Packet System bearer, a virtual circuit with a given set of QoS attributes which spans the entire mobile network including the LTE core and E-UTRAN segments;
- o GBR: Guaranteed Bit Rate. EPS bearers can be GBR, in which case they are guaranteed to not drop packets under congestion, or non-GBR, in which case no guarantee of delivery is made by the mobile network;
- o LTE: 3GPP Long Term Evolution, aka 4G;
- o E-UTRAN: LTE Radio Access Network;
- o QCI: QoS Class Identifier. In LTE networks, EPS bearers are partitioned into equivalency classes modulo the QoS treatment they receive. QCI is an integer that labels a specific QoS class. Its semantics is consistently understood by all network elements involved in packet forwarding;
- o UE: User Equipment, any device (e.g., smartphone, laptop, tablet) attached to an LTE network.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

### 3. DiffServ Code

Given the semantic equivalence of a Loss-Latency trade-off with the Non Queue Building (NQB) behaviour, this document reuses the NQB DSCP (Section 4 of [I-D.white-tsvwg-nqb]) as-is.

### 4. Marking

Endpoints SHOULD mark packets that belong to the Best Effort class and are latency sensitive by assigning the NQB DSCP value to the DS field.

The marking could also be added to other BE traffic if the default class could be reclassified by the network to use the NQB DSCP before the packet enters the mobile domain - for example by a classifier in the SGi-LAN or in an LTE router.

### 5. Relationship to a Mobile DiffServ Domain

The Mobile network is configured to give UEs a dedicated, low-latency, non-GBR, EPS bearer with QCI 7 in addition to the default EPS bearer.

A packet carrying the NQB DSCP shall be routed through the dedicated low-latency EPS bearer. A packet that has no associated NQB marking shall be routed through the default EPS bearer.

### 6. On Remarking and Bleaching

NQB markings SHOULD be preserved when forwarding via an interconnect.

The NQB DSCP can have end-to-end semantics and this might benefit any NQB-marked traffic if utilised by other path elements (e.g. allowing DS treatment if a bottleneck link happens to be in the part of the path outside the mobile access network segment).

### 7. IANA Considerations

This document makes no request to IANA.

### 8. Security Considerations

Internet applications cannot benefit from wrongly indicating low-latency as they have to pay the expense of high loss as a trade-off. Hence there is no incentive for Internet applications to set the wrong code-point.

The NQB signal is not integrity protected and could be flipped by an on-path attacker. This might negatively affect the QoS of the tampered flow.

## 9. Privacy Considerations

As described in [Shbair] state of art encrypted traffic analysis based machine learning can successfully identify the type of transported application (e.g., HTTPS, SMTP, P2P, VoIP, SSH, Skype) with good accuracy and without any need to access the clear-text. In this context, despite it being coarse grained, a 1-bit signal such as the one described in this document might be used to improve the precision of the classifier.

## 10. Acknowledgments

We would like to thank the authors of the "Latency Loss Tradeoff PHB Group" draft: Jianjie You, Michael Welzl, Brian Trammell and Kevin Smith. Big thanks to Chris Seal, Dan Druta, Diego Lopez, Shamit Bhat, Georg Mayer, Florin Baboescu, James Gruessing for the help.

This work is partially supported by the European Commission under Horizon 2020 grant agreement no. 688421 Measurement and Architecture for a Middleboxed Internet (MAMI), and by the Swiss State Secretariat for Education, Research, and Innovation under contract no. 15.0268. This support does not imply endorsement.

## 11. References

### 11.1. Normative References

- [I-D.white-tsvwg-nqb]  
White, G., "Identifying and Handling Non Queue Building Flows in a Bottleneck Link", draft-white-tsvwg-nqb-00 (work in progress), October 2018.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

## 11.2. Informative References

- [Custura] Custura, A., Venne, A., and G. Fairhurst, "Exploring DSCP modification pathologies in mobile edge networks", TMA , 2017.
- [Fossati] Fossati, T., Aranda Gutierrez, P., Kuehlewind, M., and D. Lopez, "Loss Latency Tradeoff and the Mobile Network", 2018, <<https://github.com/mami-project/roadshows/blob/master/ietf-103-lola/hotrfc/build/main.pdf>>.
- [Podlesny] Podlesny, M. and S. Gorinsky, "Rd Network Services: Differentiation Through Performance Incentives", SIGCOMM , 2008.
- [Shbair] Shbair, W., Cholez, T., Francois, J., and I. Chrisment, "A Multi-Level Framework to Identify HTTPS Services", NOMS , 2016.

## Authors' Addresses

Thomas Fossati  
Nokia

Email: [thomas.fossati@nokia.com](mailto:thomas.fossati@nokia.com)

Gorry Fairhurst  
University of Aberdeen

Email: [gorry@erg.abdn.ac.uk](mailto:gorry@erg.abdn.ac.uk)

Pedro A. Aranda Gutierrez  
Universidad Carlos III de Madrid

Email: [paranda@it.uc3m.es](mailto:paranda@it.uc3m.es)

Mirja Kuehlewind  
ETH Zurich

Email: [mirja.kuehlewind@tik.ee.ethz.ch](mailto:mirja.kuehlewind@tik.ee.ethz.ch)

INTERNET-DRAFT  
Intended Status: Standard  
Expires: January 2020

T. Herbert  
Intel

July 8, 2019

UDP Surplus Header  
draft-herbert-udp-space-hdr-01

Abstract

This specification defines the UDP Surplus Header that is an extensible and generic format applied to the UDP surplus space. The UDP surplus space comprises the bytes between the end of the UDP datagram, as indicated by the UDP Length field, and the end of the IP packet, as indicated by IP packet or payload length. The UDP Surplus Header can be either a protocol trailer of the UDP datagram, or a protocol header which effectively serves as an extended UDP header.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at  
<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at  
<http://www.ietf.org/shadow.html>

Copyright and License Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1	Introduction . . . . .	3
2	UDP Surplus Header format . . . . .	3
2.1	Protocol trailer format . . . . .	4
2.2	Protocol header format (Extended UDP header) . . . . .	6
3	Operation . . . . .	7
3.1	Sender operation . . . . .	7
3.2	Receiver operation . . . . .	8
3.2.1	Error handling . . . . .	9
4	Motivation . . . . .	9
5	Security Considerations . . . . .	11
6	IANA Considerations . . . . .	11
7	References . . . . .	11
7.1	Normative References . . . . .	11
7.2	Informative References . . . . .	11
	Appendix A: Checksum processing . . . . .	12
A.1	Transmit Checksum processing . . . . .	12
A.1.1	TX checksum for USH trailer . . . . .	12
A.1.2	TX checksum for USH header . . . . .	13
A.2	Receive Checksum handling . . . . .	13
A.2.1	Simultaneous verification . . . . .	13
A.2.2	RX checksum for USH trailer . . . . .	14
A.2.3	RX checksum for USH header . . . . .	14
	Appendix B: Protocol headers versus protocol trailers . . . . .	15
	Appendix C: Protocol field alignment . . . . .	15
	Author's Address . . . . .	16

## 1 Introduction

As defined in [RFC768], the UDP header contains a UDP Length field. The UDP Length is not required to correlate with the IP payload length of a packet such that there may be bytes between the end of the UDP datagram and the end of the IP packet. This space is referred to as the UDP surplus space.

This specification defines the UDP Surplus Header (USH) to provide a common format for the UDP surplus space. The USH is comprised of a four byte base header and some variable amount of data. The base header contains a type field that determines how the header data is interpreted. This allows different formats and uses of the UDP surplus space. UDP options [UDPOPT] are one example of a type where the header data contains a list of options.

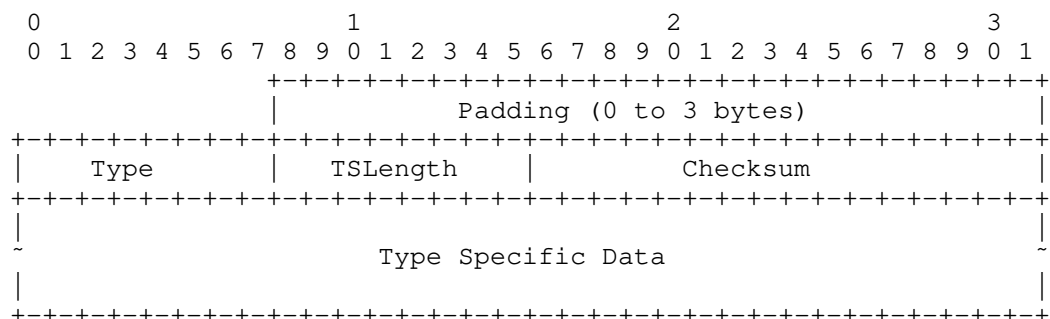
There are two use cases of USH:

- 1) Protocol trailer (section 2.1)
- 2) Protocol header or Extended UDP Header (section 2.2)

The motivations for USH, include the motivations for protocol header format in USH, are described in section 4.

## 2 UDP Surplus Header format

The common format of the UDP Surplus Header (USH) is shown below:



The fields are:

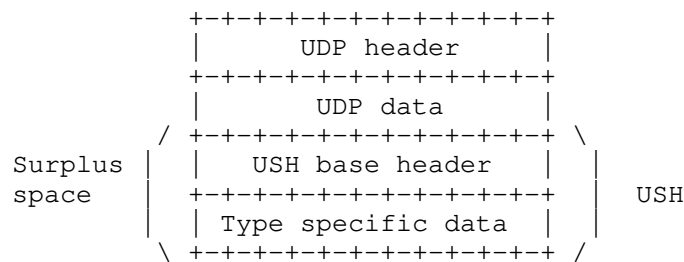
- o Padding: Aligns the UDP Surplus Header to four bytes. The number of padding bytes required is:  $3 - ((\text{udp\_length} - 1) \% 4)$ , where the `udp_length` is the length of the UDP datagram as specified in the UDP Length field. Padding bytes MUST be set to zero on transmission, and MUST be verified to be zero when received.



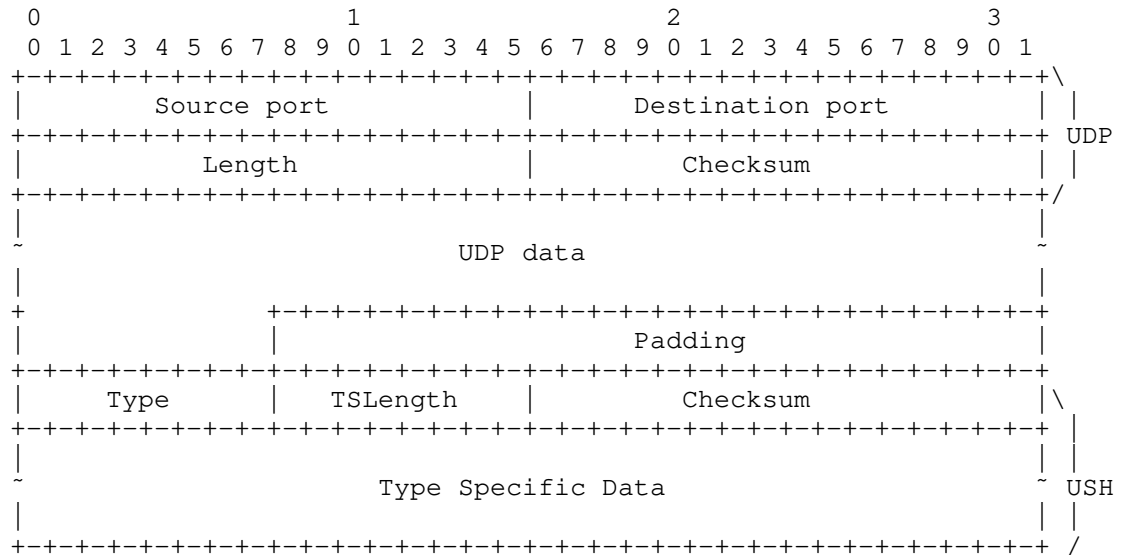
- o Type: Indicates the format of the UDP surplus space and how the Type Specific Data is interpreted. Defined Type values are:
  - o 0: Reserved
  - o 1: UDP options
  - o 2-127: Reserved
  - o 128-255: Available for private use or experimentation
- o TSLength: Length of the type specific data in units of four byte words. The length of the type specific data is thus zero to 1020 bytes.
- o Checksum: The standard one's complement checksum that covers the UDP surplus area. The coverage starts from the first byte of Padding, or the Type field if no padding is present, through the end of the IP packet. If the number of Padding bytes is odd then a zero byte is logically prepended to surplus area for the checksum calculation.
- o Type Specific Data: Variable length data that is considered part of the UDP Surplus Header. This data is interpreted per the value of the Type field.

## 2.1 Protocol trailer format

When used as a protocol trailer, the UDP Surplus Header immediately follows the UDP data. The logical protocol layering is:



The packet format of UDP Surplus Header as a protocol trailer is:

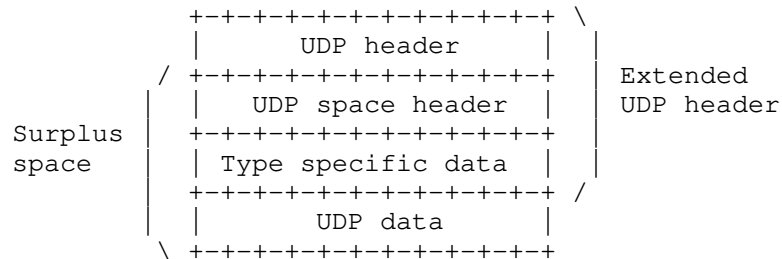


Notes:

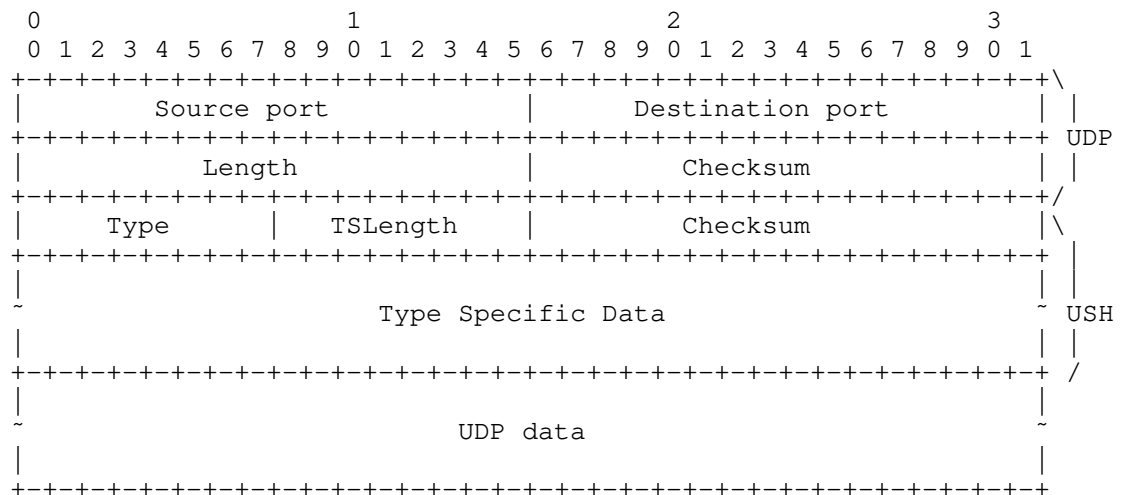
- The offset of the UDP Surplus Header from the start of the UDP header, including possible padding for the USH, is equal the UDP Length.
- The number of padding bytes is  $3 - ((\text{udp\_length} - 1) \% 4)$ , where `udp_length` is equal to the UDP Length field. The offset of the Type field of the USH is  $4 * ((\text{udp\_length} - 1) / 4 + 1)$ .
- If the size of the USH header (four plus four times TSLength) is less than the size of the UDP surplus space in a packet, then the USH is considered to be malformed (see section 3.2).
- The UDP checksum covers the UDP header and UDP data. The USH checksum covers the entire UDP surplus space.
- A legacy receiver, one that does not understand the UDP Surplus Header, will ignore the contents of the UDP surplus space and process the UDP data as normal. Protocol data that cannot correctly be ignored by a receiver, such as the fragmentation option in the [UDPOPT], MUST NOT be in a surplus space trailer.

## 2.2 Protocol header format (Extended UDP header)

The UDP Surplus Header can be used as a protocol header. Effectively, this creates an extended UDP header format. The logical protocol layering is:



The packet format containing an extended UDP header is:



### Notes:

- Since the UDP header is aligned and a multiple of four bytes, no padding for USH is necessary.
- The UDP length is fixed to be eight so that all bytes beyond the UDP header are contained in the surplus space.
- The UDP checksum covers the eight bytes of UDP header and the checksum pseudo header. The USH checksum covers the entire surplus space which includes the UDP Surplus Header and UDP data.

- The UDP data length is the IP payload length minus the size of the UDP header and the size of the UDP Surplus Header. That is:

$$\text{UDP\_data\_length} = \text{IP\_payload\_length} - 12 - (4 * \text{TSLength})$$

- If a legacy receiver, one that does not understand the UDP Surplus Header, receives a packet in protocol header format it will process it as a UDP datagram containing zero length data. Presumably, most applications will ignore such packets, however if an application applies semantics to zero length datagrams then a sender MUST NOT send packets with an extended UDP header to legacy receivers.

### 3 Operation

#### 3.1 Sender operation

A sender sets a UDP Surplus Header in the surplus space when sending an IP packet. The UDP surplus header immediately follows the UDP packet at the offset of UDP Length from the start of the UDP header. The sender MUST insert up to three bytes of padding to align the offset of the Type field in the UDP Surplus Header to four bytes. Padding bytes MUST be set to zero.

If the USH is being used as a protocol trailer then the UDP Surplus Header follows the UDP data. If a protocol header is being set then the UDP Surplus Header follows the eight byte UDP header and the UDP data follows the UDP Surplus Header.

The IP Length field in the IPv4 header or Total Length field in the IPv6 header MUST be set to include the UDP datagram and the UDP surplus space. The UDP Length field MUST be set to size of the UDP header (eight) plus the size of the UDP data in the protocol trailer use case, and MUST be set to the size of the UDP header (eight) in the protocol header use case.

The TSLength field MUST be set to reflect the length of the Type Specific Data. The Type Specific Data MUST be padded if necessary to align its length to four bytes.

The USH Checksum MUST be set. To compute the checksum:

- 1) Set the Checksum field to zero. Compute the standard one's complement two byte checksum starting from the Type field through the end of the IP packet (end of the surplus space). If the length of the surplus space is odd then a zero byte is logically appended for the purposes of the calculation.

- 2) Set the value of the Checksum field to the bitwise "not" of the checksum computed in the previous step.

### 3.2 Receiver operation

The processing for a UDP packet with surplus space is:

- 1) Check for minimum length to contain a UDP Surplus Header. If the UDP surplus space length is less than  $3 - ((\text{udp\_length} - 1) \% 4) + 4$ , then the UDP Surplus Header is considered invalid.
- 2) Check padding bytes. If the UDP Length is not a multiple of four bytes then verify that the padding bytes following the UDP payload are set to zero. The required number of padding bytes is  $3 - ((\text{udp\_length} - 1) \% 4)$ . If the padding bytes are not zero, the UDP Surplus Header is considered invalid.
- 3) Check the TSLength field. If the length determined from the TSLength field plus the starting offset of the Type Specific Data exceeds the length of the IP packet then the UDP Surplus Header is considered invalid.
- 4) Verify the checksum. Compute the one's complement checksum starting from the Type field through the end of the IP packet (the end of the surplus area). If the result of the computation is checksum zero ( $\sim 0$  or  $-0$ ) then the checksum is verified. If the checksum is not verified then the UDP Surplus Header is considered invalid.
- 5) Check the Type. If the Type is unknown to the receiver then the surplus header is considered invalid.
- 6) Process the Type Specific Data per the Type in the UDP Surplus Header. If an error condition is encountered in the course of processing the Type Specific Data then the receiver SHOULD consider that the UDP Surplus Header is invalid.
- 7) In the protocol trailer use case, if there are additional bytes beyond the UDP Surplus Header, a receiver SHOULD ignore those bytes (with the exception that the excess bytes MUST be included in the USH Checksum computation).
- 8) If the UDP Surplus Header is validated and processed, deliver the UDP data to the application.

In the case of a protocol trailer, the surplus area is discarded and the UDP data, which follows the UDP header and has length of UDP Length minus eight, is delivered to the

application.

In the case of protocol header, the UDP data delivered to the application immediately follows the UDP Surplus Header and has length of  $IP\_payload\_length - 12 - (4 * TSLength)$ .

### 3.2.1 Error handling

If an error is encountered when processing the UDP space or UDP Surplus Header such that the UDP Surplus Header is considered invalid, then the following actions should be taken:

- In the protocol trailer case (UDP Length greater than eight), the UDP surplus area SHOULD be ignored per protocol processing convention. An implementation MAY allow configuration that would discard such packets. An implementation MUST either process the surplus space or ignore the whole space. In particular, the UDP Surplus Header MUST NOT be partially processed lest that leads to indeterminate results of processing an accepted packet.
- In the case of a protocol header (a UDP packet having exactly a length of eight), the receiver SHOULD discard packets with malformed UDP surplus space or UDP Surplus Header. A receiver MAY deliver the packet to the application in the unlikely scenario that the application applies semantics to zero length UDP datagrams and there is the possibility that the surplus space is a legacy use case (i.e. the sender set surplus space but doesn't use the UDP Surplus Header format).

## 4 Motivation

This section describes the motivations for the UDP Surplus Header and motivation for protocol headers.

- o While the UDP surplus area was implicitly created by [RFC768], the space was never specifically reserved by IETF action. Prescribing a format enables interoperable and backwards compatible use of this space within the context of defined protocol specifications.
- o A common header allows different uses and extensibility of the UDP surplus space within a common framework. This is achieved by inclusion of a Type field and Type Specific Data in the UDP Surplus Header. For instance, legacy uses of surplus space could be adapted to use the format and brought into conformance.
- o Since the UDP surplus space was never reserved, there is a possibility that the UDP surplus space is already being used by

some implementations. Disambiguating these "legacy" use cases from a newly defined standard format is essential. The required Checksum field, and to a lesser extent the Type and TSLength fields, help disambiguate uses of the surplus area from legacy or accidental uses of the surplus area. Use of the extended UDP header format also reduces the chances of misinterpreting legacy uses.

- o The USH checksum is checksum offload friendly. See appendix A for discussion on checksum offload and USH.
- o The required checksum in the UDP Surplus Header properly compensates for those devices that incorrectly compute UDP checksum over the length of the IP payload as opposed to just the UDP length.
- o A fixed checksum, as opposed to placing a checksum in options, avoids the problem that a checksum can't protect against corruption of the type field for the option containing the checksum.
- o Protocols headers, such as those used in the Extended UDP Header format, are more implementation friendly than protocol trailers. See Appendix B for more discussion.
- o Maintaining four byte alignment, as is common in IP protocols, is beneficial to implementations on several hardware architectures. See Appendix C for more discussion.

## 5 Security Considerations

The UDP Surplus Header does not address nor introduce any new security considerations. The Type Specific Data in a UDP Surplus Header may contain security protocol mechanisms or require additional security considerations. Security considerations for Type Specific Data is out of scope for this document.

## 6 IANA Considerations

IANA is requested to create a registry for the UDP Surplus Header Types.

## 7 References

### 7.1 Normative References

### 7.2 Informative References

[UDPOPT] Touch, J., "Transport Options for UDP", draft-ietf-tsvwg-udp-options-07



## Appendix A: Checksum processing

This appendix is informational and does not constitute a normative part of this document.

Checksum offload is a ubiquitous feature of Network Interface Cards (NICs) that offloads checksum computation to hardware for performance. This section suggests some implementation techniques to best leverage checksum offload when UDP surplus space is being used.

Note that the USH checksum ensures that the checksum computed over the UDP surplus space sums to zero in one's complement arithmetic. This has the intended consequence that the UDP checksum calculation over just the UDP length results in the same value when the UDP checksum is computed over the UDP length and surplus space as well. This property can be exploited for efficient and interoperable processing.

### A.1 Transmit Checksum processing

A UDP packet with a UDP Surplus Header has two checksum that may need to be set on transmission: the UDP checksum and the USH checksum. The UDP checksum is optional for IPv4 and is required for IPv6 except in very narrow circumstances described in [RFC6936]. The USH checksum is always required to be set.

Most devices only offload one checksum on transmit, so a design objective is to offload the checksum that covers the most bytes and hence provides the most benefit to offload. The checksum that is not offloaded is computed by the host CPU. Generally, the checksum that covers the UDP data is the one covers the most data and should be offloaded. That is, when USH is a protocol trailer the UDP checksum should be offloaded, and when the USH is a protocol header (i.e. extended UDP header) the USH checksum should be offloaded.

In generic checksum offload, for each packet the host indicates to the device the starting offset where the checksum calculation begins and the offset of the field to write the resultant checksum. The extent of the checksum coverage is assumed to be the end of the packet. In particular, this means that even if the UDP checksum is being offloaded, the UDP surplus space is included in the device's computation. Ensuring that the surplus space sums to zero in one's complement arithmetic avoids any ambiguity with checksum offload.

#### A.1.1 TX checksum for USH trailer

The recommended procedure for setting checksums when the UDP Surplus Header is a trailer is:

- 1) On the host set the USH checksum using the normal procedures for setting the checksum (section 3.1).
- 2) Arrange for the UDP checksum to be offloaded to the device. This is done by indicating the checksum start offset to be the first byte of UDP header, indicating the checksum field offset to be the offset of the UDP checksum field, and initializing the UDP checksum field to the "bitwise not" of the appropriate IP pseudo header.

Step 1) ensures that the surplus area sums to zero in one's complement arithmetic, so that in step 2) the value that the device sets in the UDP checksum field will be correct regardless of whether the device includes the surplus area in its computation or not.

Note that the USH padding must be set to zero so it does not affect the checksum computed in step 1). The USH checksum on transmission can be correctly computed by starting the checksum computation from the offset of USH Type field.

#### A.1.2 TX checksum for USH header

The recommended procedure for setting checksums when the UDP Surplus Header is a header is:

- 1) Set the UDP checksum on the host. This is normal procedures to set the UDP checksum for a UDP datagram with length of eight.
- 2) Arrange to offload the USH checksum. The USH checksum field is initialized to zero, the offset to start the checksum calculation is set to the offset of the Type field in the USH, and the checksum field offset is set to the offset of the USH checksum field.

#### A.2 Receive Checksum handling

In the most generic form of receive checksum offload, a device performs a running checksum calculation across a packet as it is received. That is, it performs a running ones complement addition over two byte words as they are received. The device then provides the computed value, referred to as the "checksum complete" value, to the host in the meta data (receive descriptor) for the packet. The host can use this value to verify one or more packet checksums contained in the packet.

##### A.2.1 Simultaneous verification

If a device provides a checksum complete value and the UDP checksum

is set, then both the UDP checksum and USH checksum can be simultaneously verified:

- 1) Pull up checksum to start of the UDP header. That is the checksum complete value is computed from the start of the UDP header through the end of the IP packet.
- 2) Verify the UDP checksum taking into account the pseudo header. If the UDP checksum is verified, then the USH checksum is also verified.

If the simultaneous verification fails then further work might be needed if checksum failure of the surplus space does not result in the packet being dropped. For instance, if the surplus space is to be ignored in the trailer use case.

#### A.2.2 RX checksum for USH trailer

The recommended procedure for independently verifying the UDP and USH checksums when the UDP Surplus Header is a protocol trailer is:

- 1) Compute the one's complement checksum across the UDP surplus space. If checksum zero is the result, then the USH checksum is verified.
- 2) Perform one's complement subtraction of the value derived in step 1) from the checksum complete value. The result is the checksum complete value across just the UDP header and UDP data.
- 3) Compute the IP pseudo header for the UDP checksum and one's complement add the result to that of step 2). If the result is checksum zero then the UDP checksum is verified.

If the UDP checksum is zero (unset) then only the USH checksum needs to be verified so steps 2) and 3) can be omitted.

#### A.2.3 RX checksum for USH header

The recommended procedure for independently verifying the UDP and USH checksums when the UDP Surplus Header is a protocol header is:

- 1) Compute the one's complement checksum across the UDP header.
- 2) Compute the IP pseudo header for the UDP checksum and one's complement add the result to that of step 1). If the result is checksum zero then the checksum of the UDP header (zero length datagram) is verified.

- 3) Perform one's complement subtraction of the value derived in step 1) from the checksum complete value. The result is the checksum complete value across just the UDP surplus space. If zero is the result, then the USH checksum is valid.

If the UDP checksum is zero (unset) then only the USH checksum needs to be verified, so step 2) can be omitted.

## Appendix B: Protocol headers versus versus protocol trailers

This appendix is informational and does not constitute a normative part of this document.

Protocol headers by definition are data at the precede the payload of a packet, whereas protocol trailers follow the payload. By nearly universal convention, IP protocols specify protocol headers (e.g. IP, TCP, UDP, Extension headers) and not protocol trailers. A notable exception to this is ESP where the integrity check value is placed after the payload data.

Both software and hardware implementations are designed and optimized for processing protocol headers.

A common technique in software implementations is to "pull up" all the headers in a packet into a contiguous buffer as various protocol layers are processed. To process a protocol trailer, such as a UDP Surplus Header in the trailer use case, an alternate mechanism is needed. This may result in copying data from the end of the packet into a contiguous buffer. Another disadvantage of protocol trailers is that when they are processed a cache miss is almost certain. This will be especially noticeable with hardware techniques that attempt to pre-populate the CPU data cache with some number of header bytes (such as data Direct Data I/O).

High performance hardware devices that perform Deep Packet Inspection (DPI) will be even more sensitive to protocol trailers. Often such devices have a fixed length parsing buffer of X bytes (where X is commonly 64, 128, or 256 bytes). When a device receives a packet, the first X bytes of the packet are preloaded into the parsing buffer before processing commences. Protocol processing is performed on the bytes in the parsing buffer. If the protocol headers extend beyond the parsing buffer then either the device won't process the headers (which may mean they drop the packet) or the packet is relegated to a slow path. Neither behavior is desirable. Given that protocol trailers follow packet payload, it will be common that the protocol trailers for a packet are not contained with parsing buffer.

## Appendix C: Protocol field alignment

This appendix is informational and does not constitute a normative part of this document.

It is often convenient to access multi-byte protocol fields in a protocol header in memory using CPU instructions to access a field as a word (two bytes) or double word (four bytes). When such accesses are done, the data being accessed can be "aligned" or "unaligned". An aligned data access happens when the address of the operation modulo the size of the operand is zero, and conversely an unaligned access occurs when the when the address of the operation modulo the size of the operand is non-zero. On certain CPU architectures including SPARC, older versions of ARM, some cases of RISC-V, and even a corner case in x86, an unaligned access may incur a substantial performance penalty compared to an aligned access. For instance, an unaligned access may result in a software trap and handling the memory access in software.

By convention, most IETF protocols are structured to ensure that multi-byte fields have an offset within the respective protocol header that is properly aligned per their field size. Additionally, most IP protocols are defined to have length that is a multiple of four bytes. These conventions, along with some implementation techniques, have mostly allowed software implementations to be reusable across different architectures without the sustaining performance hit of unaligned accesses.

The Padding field in UDP Surplus Header is important to maintain the benefits of aligned protocol headers.

#### Author's Address

Tom Herbert  
Intel  
Santa Clara, CA  
USA

Email: tom@quantonium.net

Transport Area working group (tsvwg)  
Internet-Draft  
Intended status: Experimental  
Expires: 5 November 2022

K. De Schepper  
Nokia Bell Labs  
B. Briscoe, Ed.  
Independent  
G. White  
CableLabs  
4 May 2022

DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput  
(L4S)  
draft-ietf-tsvwg-aqm-dualq-coupled-23

Abstract

This specification defines a framework for coupling the Active Queue Management (AQM) algorithms in two queues intended for flows with different responses to congestion. This provides a way for the Internet to transition from the scaling problems of standard TCP Reno-friendly ('Classic') congestion controls to the family of 'Scalable' congestion controls. These are designed for consistently very Low queuing Latency, very Low congestion Loss and Scaling of per-flow throughput (L4S) by using Explicit Congestion Notification (ECN) in a modified way. Until the Coupled DualQ, these L4S senders could only be deployed where a clean-slate environment could be arranged, such as in private data centres. The coupling acts like a semi-permeable membrane: isolating the sub-millisecond average queuing delay and zero congestion loss of L4S from Classic latency and loss; but pooling the capacity between any combination of Scalable and Classic flows with roughly equivalent throughput per flow. The DualQ achieves this indirectly, without having to inspect transport layer flow identifiers and without compromising the performance of the Classic traffic, relative to a single queue. The DualQ design has low complexity and requires no configuration for the public Internet.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 5 November 2022.

## Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Outline of the Problem . . . . .	3
1.2. Scope . . . . .	6
1.3. Terminology . . . . .	7
1.4. Features . . . . .	9
2. DualQ Coupled AQM . . . . .	11
2.1. Coupled AQM . . . . .	11
2.2. Dual Queue . . . . .	13
2.3. Traffic Classification . . . . .	13
2.4. Overall DualQ Coupled AQM Structure . . . . .	14
2.5. Normative Requirements for a DualQ Coupled AQM . . . . .	17
2.5.1. Functional Requirements . . . . .	17
2.5.1.1. Requirements in Unexpected Cases . . . . .	18
2.5.2. Management Requirements . . . . .	19
2.5.2.1. Configuration . . . . .	20
2.5.2.2. Monitoring . . . . .	21
2.5.2.3. Anomaly Detection . . . . .	22
2.5.2.4. Deployment, Coexistence and Scaling . . . . .	22
3. IANA Considerations (to be removed by RFC Editor) . . . . .	22
4. Security Considerations . . . . .	22
4.1. Low Delay without Requiring Per-Flow Processing . . . . .	22
4.2. Handling Unresponsive Flows and Overload . . . . .	23
4.2.1. Unresponsive Traffic without Overload . . . . .	24
4.2.2. Avoiding Short-Term Classic Starvation: Sacrifice L4S Throughput or Delay? . . . . .	25

4.2.3. L4S ECN Saturation: Introduce Drop or Delay? . . . .	26
4.2.3.1. Protecting against Overload by Unresponsive ECN-Capable Traffic . . . . .	28
5. Acknowledgements . . . . .	28
6. Contributors . . . . .	29
7. References . . . . .	29
7.1. Normative References . . . . .	29
7.2. Informative References . . . . .	30
Appendix A. Example DualQ Coupled PI2 Algorithm . . . . .	35
A.1. Pass #1: Core Concepts . . . . .	36
A.2. Pass #2: Edge-Case Details . . . . .	47
Appendix B. Example DualQ Coupled Curvy RED Algorithm . . . . .	51
B.1. Curvy RED in Pseudocode . . . . .	51
B.2. Efficient Implementation of Curvy RED . . . . .	57
Appendix C. Choice of Coupling Factor, $k$ . . . . .	59
C.1. RTT-Dependence . . . . .	59
C.2. Guidance on Controlling Throughput Equivalence . . . . .	60
Authors' Addresses . . . . .	64

## 1. Introduction

This document specifies a framework for DualQ Coupled AQMs, which is the network part of the L4S architecture [I-D.ietf-tsvwg-l4s-arch]. L4S enables both very low queuing latency (sub-millisecond on average) and high throughput at the same time, for ad hoc numbers of capacity-seeking applications all sharing the same capacity.

### 1.1. Outline of the Problem

Latency is becoming the critical performance factor for many (most?) applications on the public Internet, e.g. interactive Web, Web services, voice, conversational video, interactive video, interactive remote presence, instant messaging, online gaming, remote desktop, cloud-based applications, and video-assisted remote control of machinery and industrial processes. In the developed world, further increases in access network bit-rate offer diminishing returns, whereas latency is still a multi-faceted problem. In the last decade or so, much has been done to reduce propagation time by placing caches or servers closer to users. However, queuing remains a major intermittent component of latency.



Traditionally very low latency has only been available for a few selected low rate applications, that confine their sending rate within a specially carved-off portion of capacity, which is prioritized over other traffic, e.g. Diffserv EF [RFC3246]. Up to now it has not been possible to allow any number of low latency, high throughput applications to seek to fully utilize available capacity, because the capacity-seeking process itself causes too much queuing delay.

To reduce this queuing delay caused by the capacity seeking process, changes either to the network alone or to end-systems alone are in progress. L4S involves a recognition that both approaches are yielding diminishing returns:

- \* Recent state-of-the-art active queue management (AQM) in the network, e.g. FQ-CoDel [RFC8290], PIE [RFC8033], Adaptive RED [ARED01] ) has reduced queuing delay for all traffic, not just a select few applications. However, no matter how good the AQM, the capacity-seeking (sawtooth) rate of TCP-like congestion controls represents a lower limit that will either cause queuing delay to vary or cause the link to be under-utilized. These AQMs are tuned to allow a typical capacity-seeking Reno-friendly flow to induce an average queue that roughly doubles the base RTT, adding 5-15 ms of queuing on average (cf. 500 microseconds with L4S for the same mix of long-running and web traffic). However, for many applications low delay is not useful unless it is consistently low. With these AQMs, 99th percentile queuing delay is 20-30 ms (cf. 2 ms with the same traffic over L4S).
- \* Similarly, recent research into using e2e congestion control without needing an AQM in the network (e.g. BBR [I-D.cardwell-iccr-g-bbr-congestion-control]) seems to have hit a similar lower limit to queuing delay of about 20ms on average but there are also regular 25ms delay spikes due to bandwidth probes and 60ms spikes due to flow-starts.

L4S learns from the experience of Data Center TCP [RFC8257], which shows the power of complementary changes both in the network and on end-systems. DCTCP teaches us that two small but radical changes to congestion control are needed to cut the two major outstanding causes of queuing delay variability:

1. Far smaller rate variations (sawteeth) than Reno-friendly congestion controls;
2. A shift of smoothing and hence smoothing delay from network to sender.

Without the former, a 'Classic' (e.g. Reno-friendly) flow's round trip time (RTT) varies between roughly 1 and 2 times the base RTT between the machines in question. Without the latter a 'Classic' flow's response to changing events is delayed by a worst-case (transcontinental) RTT, which could be hundreds of times the actual smoothing delay needed for the RTT of typical traffic from localized CDNs.

These changes are the two main features of the family of so-called 'Scalable' congestion controls (which includes DCTCP, TCP Prague and SCReAM). Both these changes only reduce delay in combination with a complementary change in the network and they are both only feasible with ECN, not drop, for the signalling:

1. The smaller sawteeth allow an extremely shallow ECN packet-marking threshold in the queue.
2. And no smoothing in the network means that every fluctuation of the queue is signalled immediately.

Without ECN, either of these would lead to very high loss levels. But, with ECN, the resulting high marking levels are just signals, not impairments. BBRv2 combines the best of both worlds - it works as a scalable congestion control when ECN is available, but also aims to minimize delay when it isn't.

However, until now, Scalable congestion controls (like DCTCP) did not co-exist well in a shared ECN-capable queue with existing ECN-capable TCP Reno [RFC5681] or Cubic [RFC8312] congestion controls -- Scalable controls are so aggressive that these 'Classic' algorithms would drive themselves to a small capacity share. Therefore, until now, L4S controls could only be deployed where a clean-slate environment could be arranged, such as in private data centres (hence the name DCTCP).

This document specifies a 'DualQ Coupled AQM' extension that solves the problem of coexistence between Scalable and Classic flows, without having to inspect flow identifiers. It is not like flow-queuing approaches [RFC8290] that classify packets by flow identifier into separate queues in order to isolate sparse flows from the higher latency in the queues assigned to heavier flows. If a flow needs both low delay and high throughput, having a queue to itself does not isolate it from the harm it causes to itself. In contrast, DualQ Coupled AQMs address the root cause of the latency problem -- they are an enabler for the smooth low latency scalable behaviour of Scalable congestion controls, so that every packet in every flow can potentially enjoy very low latency, then there would be no need to isolate each flow into a separate queue.

## 1.2. Scope

L4S involves complementary changes in the network and on end-systems:

**Network:** A DualQ Coupled AQM (defined in the present document) or a modification to flow-queue AQMs (described in section 4.2.b of the L4S architecture [I-D.ietf-tsvwg-l4s-arch]);

**End-system:** A Scalable congestion control (defined in section 4 of the L4S ECN protocol [I-D.ietf-tsvwg-ecn-l4s-id]).

**Packet identifier:** The network and end-system parts of L4S can be deployed incrementally, because they both identify L4S packets using the experimentally assigned explicit congestion notification (ECN) codepoints in the IP header: ECT(1) and CE [RFC8311] [I-D.ietf-tsvwg-ecn-l4s-id].

Data Center TCP (DCTCP [RFC8257]) is an example of a Scalable congestion control for controlled environments that has been deployed for some time in Linux, Windows and FreeBSD operating systems. During the progress of this document through the IETF a number of other Scalable congestion controls were implemented, e.g. TCP Prague [I-D.briscoe-iccrp-prague-congestion-control] [PragueLinux], BBRv2 [BBRv2], [I-D.cardwell-iccrp-bbr-congestion-control], QUIC Prague and the L4S variant of SCREAM for real-time media [RFC8298].

The focus of this specification is to enable deployment of the network part of the L4S service. Then, without any management intervention, applications can exploit this new network capability as their operating systems migrate to Scalable congestion controls, which can then evolve while their benefits are being enjoyed by everyone on the Internet.

The DualQ Coupled AQM framework can incorporate any AQM designed for a single queue that generates a statistical or deterministic mark/drop probability driven by the queue dynamics. Pseudocode examples of two different DualQ Coupled AQMs are given in the appendices. In many cases the framework simplifies the basic control algorithm, and requires little extra processing. Therefore it is believed the Coupled AQM would be applicable and easy to deploy in all types of buffers; buffers in cost-reduced mass-market residential equipment; buffers in end-system stacks; buffers in carrier-scale equipment including remote access servers, routers, firewalls and Ethernet switches; buffers in network interface cards, buffers in virtualized network appliances, hypervisors, and so on.

For the public Internet, nearly all the benefit will typically be achieved by deploying the Coupled AQM into either end of the access link between a 'site' and the Internet, which is invariably the bottleneck (see section 6.4 of [I-D.ietf-tsvwg-l4s-arch] about deployment, which also defines the term 'site' to mean a home, an office, a campus or mobile user equipment).

Latency is not the only concern of L4S:

- \* The "Low Loss" part of the name denotes that L4S generally achieves zero congestion loss (which would otherwise cause retransmission delays), due to its use of ECN.
- \* The "Scalable throughput" part of the name denotes that the per-flow throughput of Scalable congestion controls should scale indefinitely, avoiding the imminent scaling problems with 'TCP-Friendly' congestion control algorithms [RFC3649].

The former is clearly in scope of this AQM document. However, the latter is an outcome of the end-system behaviour, and therefore outside the scope of this AQM document, even though the AQM is an enabler.

The overall L4S architecture [I-D.ietf-tsvwg-l4s-arch] gives more detail, including on wider deployment aspects such as backwards compatibility of Scalable congestion controls in bottlenecks where a DualQ Coupled AQM has not been deployed. The supporting papers [DualPI2Linux], [PI2], [DCTtH19] and [PI2param] give the full rationale for the AQM's design, both discursively and in more precise mathematical form, as well as the results of performance evaluations. The main results have been validated independently when using the Prague congestion control [Boru20] (experiments are run using Prague and DCTCP, but only the former are relevant for validation, because Prague fixes a number of problems with the Linux DCTCP code that make it unsuitable for the public Internet).

### 1.3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when, and only when, they appear in all capitals, as shown here.

The DualQ Coupled AQM uses two queues for two services. Each of the following terms identifies both the service and the queue that provides the service:

Classic service/queue: The Classic service is intended for all the

congestion control behaviours that co-exist with Reno [RFC5681] (e.g. Reno itself, Cubic [RFC8312], TFRC [RFC5348]).

Low-Latency, Low-Loss Scalable throughput (L4S) service/queue: The 'L4S' service is intended for traffic from scalable congestion control algorithms, such as TCP Prague [I-D.briscoe-iccrp-prague-congestion-control], which was derived from Data Center TCP [RFC8257]. The L4S service is for more general traffic than just TCP Prague -- it allows the set of congestion controls with similar scaling properties to Prague to evolve, such as the examples listed earlier (Relentless, SCReAM, etc.).

Classic Congestion Control: A congestion control behaviour that can co-exist with standard TCP Reno [RFC5681] without causing significantly negative impact on its flow rate [RFC5033]. With Classic congestion controls, such as Reno or Cubic, because flow rate has scaled since TCP congestion control was first designed in 1988, it now takes hundreds of round trips (and growing) to recover after a congestion signal (whether a loss or an ECN mark) as shown in the examples in section 5.1 of the L4S architecture [I-D.ietf-tsvwg-l4s-arch] and in [RFC3649]. Therefore control of queuing and utilization becomes very slack, and the slightest disturbances (e.g. from new flows starting) prevent a high rate from being attained.

Scalable Congestion Control: A congestion control where the average time from one congestion signal to the next (the recovery time) remains invariant as the flow rate scales, all other factors being equal. This maintains the same degree of control over queueing and utilization whatever the flow rate, as well as ensuring that high throughput is robust to disturbances. For instance, DCTCP averages 2 congestion signals per round-trip whatever the flow rate, as do other recently developed scalable congestion controls, e.g. Relentless TCP [Mathis09], TCP Prague [I-D.briscoe-iccrp-prague-congestion-control], [PragueLinux], BBRv2 [BBRv2], [I-D.cardwell-iccrp-bbr-congestion-control] and the L4S variant of SCREAM for real-time media [SCReAM], [RFC8298]). For the public Internet a Scalable transport has to comply with the requirements in Section 4 of [I-D.ietf-tsvwg-ecn-l4s-id] (aka. the 'Prague L4S requirements').

C: Abbreviation for Classic, e.g. when used as a subscript.

L: Abbreviation for L4S, e.g. when used as a subscript.

The terms Classic or L4S can also qualify other nouns, such as 'codepoint', 'identifier', 'classification', 'packet', 'flow'. For example: an L4S packet means a packet with an L4S identifier sent from an L4S congestion control.

Both Classic and L4S services can cope with a proportion of unresponsive or less-responsive traffic as well, but in the L4S case its rate has to be smooth enough or low enough not to build a queue (e.g. DNS, VoIP, game sync datagrams, etc). The DualQ Coupled AQM behaviour is defined to be similar to a single FIFO queue with respect to unresponsive and overload traffic.

Reno-friendly: The subset of Classic traffic that is friendly to the standard Reno congestion control defined for TCP in [RFC5681]. Reno-friendly is used in place of 'TCP-friendly', given the latter has become imprecise, because the TCP protocol is now used with so many different congestion control behaviours, and Reno is used in non-TCP transports such as QUIC.

Classic ECN: The original Explicit Congestion Notification (ECN) protocol [RFC3168], which requires ECN signals to be treated the same as drops, both when generated in the network and when responded to by the sender.

For L4S, the names used for the four codepoints of the 2-bit IP-ECN field are unchanged from those defined in [RFC3168]: Not ECT, ECT(0), ECT(1) and CE, where ECT stands for ECN-Capable Transport and CE stands for Congestion Experienced. A packet marked with the CE codepoint is termed 'ECN-marked' or sometimes just 'marked' where the context makes ECN obvious.

#### 1.4. Features

The AQM couples marking and/or dropping from the Classic queue to the L4S queue in such a way that a flow will get roughly the same throughput whichever it uses. Therefore both queues can feed into the full capacity of a link and no rates need to be configured for the queues. The L4S queue enables Scalable congestion controls like DCTCP or TCP Prague to give very low and predictably low latency, without compromising the performance of competing 'Classic' Internet traffic.

Thousands of tests have been conducted in a typical fixed residential broadband setting. Experiments used a range of base round trip delays up to 100ms and link rates up to 200 Mb/s between the data centre and home network, with varying amounts of background traffic in both queues. For every L4S packet, the AQM kept the average queuing delay below 1ms (or 2 packets where serialization delay

exceeded 1ms on slower links), with 99th percentile no worse than 2ms. No losses at all were introduced by the L4S AQM. Details of the extensive experiments are available [DualPI2Linux], [PI2], [DCTtH19].

In all these experiments, the host was connected to the home network by fixed Ethernet, in order to quantify the queuing delay that can be achieved by a user who cares about delay. It should be emphasized that L4S support at the bottleneck link cannot 'undelay' bursts introduced by another link on the path, for instance by legacy WiFi equipment. However, if L4S support is added to the queue feeding the `_outgoing_` WAN link of a home gateway, it would be counterproductive not to also reduce the burstiness of the `_incoming_` WiFi. Also, trials of WiFi equipment with an L4S DualQ Coupled AQM on the `_outgoing_` WiFi interface are in progress, and early results of an L4S DualQ Coupled AQM in a 5G radio access network testbed with emulated outdoor cell edge radio fading are given in [L4S\_5G].

Subjective testing has also been conducted by multiple people all simultaneously using very demanding high bandwidth low latency applications over a single shared access link [L4Sdemo16]. In one application, each user could use finger gestures to pan or zoom their own high definition (HD) sub-window of a larger video scene generated on the fly in 'the cloud' from a football match. Another user wearing VR goggles was remotely receiving a feed from a 360-degree camera in a racing car, again with the sub-window in their field of vision generated on the fly in 'the cloud' dependent on their head movements. Even though other users were also downloading large amounts of L4S and Classic data, playing a gaming benchmark and watchings videos over the same 40Mb/s downstream broadband link, latency was so low that the football picture appeared to stick to the user's finger on the touch pad and the experience fed from the remote camera did not noticeably lag head movements. All the L4S data (even including the downloads) achieved the same very low latency. With an alternative AQM, the video noticeably lagged behind the finger gestures and head movements.

Unlike Diffserv Expedited Forwarding, the L4S queue does not have to be limited to a small proportion of the link capacity in order to achieve low delay. The L4S queue can be filled with a heavy load of capacity-seeking flows (TCP Prague etc.) and still achieve low delay. The L4S queue does not rely on the presence of other traffic in the Classic queue that can be 'overtaken'. It gives low latency to L4S traffic whether or not there is Classic traffic. The tail latency of traffic served by the Classic AQM is sometimes a little better sometimes a little worse, when a proportion of the traffic is L4S.

The two queues are only necessary because:

- \* the large variations (sawteeth) of Classic flows need roughly a base RTT of queuing delay to ensure full utilization
- \* Scalable flows do not need a queue to keep utilization high, but they cannot keep latency predictably low if they are mixed with Classic traffic,

The L4S queue has latency priority within sub-round trip timescales, but over longer periods the coupling from the Classic to the L4S AQM (explained below) ensures that it does not have bandwidth priority over the Classic queue.

## 2. DualQ Coupled AQM

There are two main aspects to the approach:

- \* The Coupled AQM that addresses throughput equivalence between Classic (e.g. Reno, Cubic) flows and L4S flows (that satisfy the Prague L4S requirements).
- \* The Dual Queue structure that provides latency separation for L4S flows to isolate them from the typically large Classic queue.

### 2.1. Coupled AQM

In the 1990s, the 'TCP formula' was derived for the relationship between the steady-state congestion window,  $cwnd$ , and the drop probability,  $p$  of standard Reno congestion control [RFC5681]. To a first order approximation, the steady-state  $cwnd$  of Reno is inversely proportional to the square root of  $p$ .

The design focuses on Reno as the worst case, because if it does no harm to Reno, it will not harm Cubic or any traffic designed to be friendly to Reno. TCP Cubic implements a Reno-compatibility mode, which is relevant for typical RTTs under 20ms as long as the throughput of a single flow is less than about 350Mb/s. In such cases it can be assumed that Cubic traffic behaves similarly to Reno. The term 'Classic' will be used for the collection of Reno-friendly traffic including Cubic and potentially other experimental congestion controls intended not to significantly impact the flow rate of Reno.



A supporting paper [PI2] includes the derivation of the equivalent rate equation for DCTCP, for which  $cwnd$  is inversely proportional to  $p$  (not the square root), where in this case  $p$  is the ECN marking probability. DCTCP is not the only congestion control that behaves like this, so the term 'Scalable' will be used for all similar congestion control behaviours (see examples in Section 1.2). The term 'L4S' is used for traffic driven by a Scalable congestion control that also complies with the additional 'Prague L4S' requirements [I-D.ietf-tsvwg-ecn-l4s-id].

For safe co-existence, under stationary conditions, a Scalable flow has to run at roughly the same rate as a Reno TCP flow (all other factors being equal). So the drop or marking probability for Classic traffic,  $p_C$  has to be distinct from the marking probability for L4S traffic,  $p_L$ . The original ECN specification [RFC3168] required these probabilities to be the same, but [RFC8311] updates RFC 3168 to enable experiments in which these probabilities are different.

Also, to remain stable, Classic sources need the network to smooth  $p_C$  so it changes relatively slowly. It is hard for a network node to know the RTTs of all the flows, so a Classic AQM adds a `_worst-case_` RTT of smoothing delay (about 100-200 ms). In contrast, L4S shifts responsibility for smoothing ECN feedback to the sender, which only delays its response by its `_own_` RTT, as well as allowing a more immediate response if necessary.

The Coupled AQM achieves safe coexistence by making the Classic drop probability  $p_C$  proportional to the square of the coupled L4S probability  $p_{CL}$ .  $p_{CL}$  is an input to the instantaneous L4S marking probability  $p_L$  but it changes as slowly as  $p_C$ . This makes the Reno flow rate roughly equal the DCTCP flow rate, because the squaring of  $p_{CL}$  counterbalances the square root of  $p_C$  in the 'TCP formula' of Classic Reno congestion control.

Stating this as a formula, the relation between Classic drop probability,  $p_C$ , and the coupled L4S probability  $p_{CL}$  needs to take the form:

$$p_C = ( p_{CL} / k )^2 \quad (1)$$

where  $k$  is the constant of proportionality, which is termed the coupling factor.

## 2.2. Dual Queue

Classic traffic needs to build a large queue to prevent under-utilization. Therefore a separate queue is provided for L4S traffic, and it is scheduled with priority over the Classic queue. Priority is conditional to prevent starvation of Classic traffic in certain conditions (see Section 2.4).

Nonetheless, coupled marking ensures that giving priority to L4S traffic still leaves the right amount of spare scheduling time for Classic flows to each get equivalent throughput to DCTCP flows (all other factors such as RTT being equal).

## 2.3. Traffic Classification

Both the Coupled AQM and DualQ mechanisms need an identifier to distinguish L4S (L) and Classic (C) packets. Then the coupling algorithm can achieve coexistence without having to inspect flow identifiers, because it can apply the appropriate marking or dropping probability to all flows of each type. A separate specification [I-D.ietf-tsvwg-ecn-l4s-id] requires the network to treat the ECT(1) and CE codepoints of the ECN field as this identifier. An additional process document has proved necessary to make the ECT(1) codepoint available for experimentation [RFC8311].

For policy reasons, an operator might choose to steer certain packets (e.g. from certain flows or with certain addresses) out of the L queue, even though they identify themselves as L4S by their ECN codepoints. In such cases, the L4S ECN protocol [I-D.ietf-tsvwg-ecn-l4s-id] says that the device "MUST NOT alter the end-to-end L4S ECN identifier", so that it is preserved end-to-end. The aim is that each operator can choose how it treats L4S traffic locally, but an individual operator does not alter the identification of L4S packets, which would prevent other operators downstream from making their own choices on how to treat L4S traffic.

In addition, an operator could use other identifiers to classify certain additional packet types into the L queue that it deems will not risk harm to the L4S service. For instance addresses of specific applications or hosts; specific Diffserv codepoints such as EF (Expedited Forwarding), Voice-Admit or the Non-Queue-Building (NQB) per-hop behaviour; or certain protocols (e.g. ARP, DNS) (see Section 5.4.1 of [I-D.ietf-tsvwg-ecn-l4s-id]). Note that the mechanism only reads these identifiers. [I-D.ietf-tsvwg-ecn-l4s-id] says it "MUST NOT alter these non-ECN identifiers". Thus, the L queue is not solely an L4S queue, it can be considered more generally as a low latency queue.

#### 2.4. Overall DualQ Coupled AQM Structure

Figure 1 shows the overall structure that any DualQ Coupled AQM is likely to have. This schematic is intended to aid understanding of the current designs of DualQ Coupled AQMs. However, it is not intended to preclude other innovative ways of satisfying the normative requirements in Section 2.5 that minimally define a DualQ Coupled AQM. Also, the schematic only illustrates operation under normally expected circumstances; behaviour under overload or with operator-specific classifiers is deferred to Section 2.5.1.1.

The classifier on the left separates incoming traffic between the two queues (L and C). Each queue has its own AQM that determines the likelihood of marking or dropping ( $p_L$  and  $p_C$ ). It has been proved [PI2] that it is preferable to control load with a linear controller, then square the output before applying it as a drop probability to Reno-friendly traffic (because Reno congestion control decreases its load proportional to the square-root of the increase in drop). So, the AQM for Classic traffic needs to be implemented in two stages: i) a base stage that outputs an internal probability  $p'$  (pronounced p-prime); and ii) a squaring stage that outputs  $p_C$ , where

$$p_C = (p')^2. \quad (2)$$

Substituting for  $p_C$  in Eqn (1) gives:

$$p' = p_{CL} / k$$

So the slow-moving input to ECN marking in the L queue (the coupled L4S probability) is:

$$p_{CL} = k * p'. \quad (3)$$

The actual ECN marking probability  $p_L$  that is applied to the L queue needs to track the immediate L queue delay under L-only congestion conditions, as well as track  $p_{CL}$  under coupled congestion conditions. So the L queue uses a native AQM that calculates a probability  $p'_L$  as a function of the instantaneous L queue delay. And, given the L queue has conditional priority over the C queue, whenever the L queue grows, the AQM ought to apply marking probability  $p'_L$ , but  $p_L$  ought not to fall below  $p_{CL}$ . This suggests:

$$p_L = \max(p'_L, p_{CL}), \quad (4)$$

which has also been found to work very well in practice.

The two transformations of  $p'$  in equations (2) and (3) implement the required coupling given in equation (1) earlier.

The constant of proportionality or coupling factor,  $k$ , in equation (1) determines the ratio between the congestion probabilities (loss or marking) experienced by L4S and Classic traffic. Thus  $k$  indirectly determines the ratio between L4S and Classic flow rates, because flows (assuming they are responsive) adjust their rate in response to congestion probability. Appendix C.2 gives guidance on the choice of  $k$  and its effect on relative flow rates.

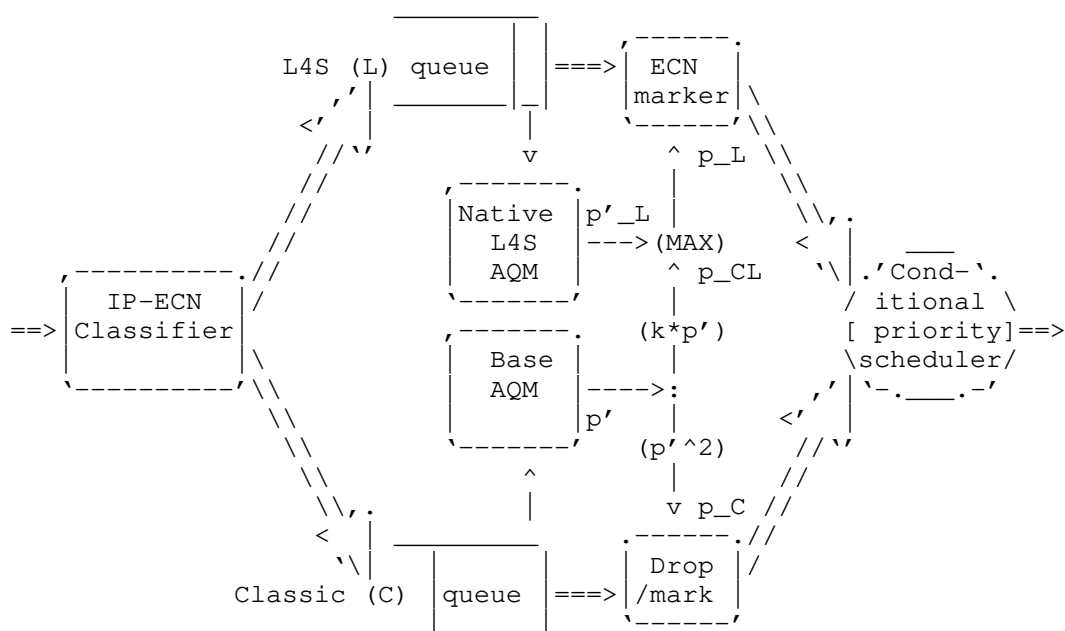


Figure 1: DualQ Coupled AQM Schematic

Legend: ==> traffic flow; ---> control dependency.

After the AQMs have applied their dropping or marking, the scheduler forwards their packets to the link. Even though the scheduler gives priority to the L queue, it is not as strong as the coupling from the C queue. This is because, as the C queue grows, the base AQM applies more congestion signals to L traffic (as well as C). As L flows reduce their rate in response, they use less than the scheduling share for L traffic. So, because the scheduler is work preserving, it schedules any C traffic in the gaps.

Giving priority to the L queue has the benefit of very low L queue delay, because the L queue is kept empty whenever L traffic is controlled by the coupling. Also there only has to be a coupling in one direction - from Classic to L4S. Priority has to be conditional in some way to prevent the C queue being starved in the short-term (see Section 4.2.2) to give C traffic a means to push in, as explained next. With normal responsive L traffic, the coupled ECN marking gives C traffic the ability to push back against even strict priority, by congestion marking the L traffic to make it yield some space. However, if there is just a small finite set of C packets (e.g. a DNS request or an initial window of data) some Classic AQMs will not induce enough ECN marking in the L queue, no matter how long the small set of C packets waits. Then, if the L queue happens to remain busy, the C traffic would never get a scheduling opportunity from a strict priority scheduler. Ideally the Classic AQM would be designed to increase the coupled marking the longer that C packets have been waiting, but this is not always practical - hence the need for L priority to be conditional. Giving a small weight or limited waiting time for C traffic improves response times for short Classic messages, such as DNS requests, and improves Classic flow startup because immediate capacity is available.

Example DualQ Coupled AQM algorithms called DualPI2 and Curvy RED are given in Appendix A and Appendix B. Either example AQM can be used to couple packet marking and dropping across a dual Q.

DualPI2 uses a Proportional-Integral (PI) controller as the Base AQM. Indeed, this Base AQM with just the squared output and no L4S queue can be used as a drop-in replacement for PIE [RFC8033], in which case it is just called PI2 [PI2]. PI2 is a principled simplification of PIE that is both more responsive and more stable in the face of dynamically varying load.

Curvy RED is derived from RED [RFC2309], except its configuration parameters are delay-based to make them insensitive to link rate and it requires less operations per packet than RED. However, DualPI2 is more responsive and stable over a wider range of RTTs than Curvy RED. As a consequence, at the time of writing, DualPI2 has attracted more development and evaluation attention than Curvy RED, leaving the Curvy RED design not so fully evaluated.

Both AQMs regulate their queue in units of time rather than bytes. As already explained, this ensures configuration can be invariant for different drain rates. With AQMs in a dualQ structure this is particularly important because the drain rate of each queue can vary rapidly as flows for the two queues arrive and depart, even if the combined link rate is constant.

It would be possible to control the queues with other alternative AQMs, as long as the normative requirements (those expressed in capitals) in Section 2.5 are observed.

The two queues could optionally be part of a larger queuing hierarchy, such as the initial example ideas in [I-D.briscoe-tsvwg-l4s-diffserv].

## 2.5. Normative Requirements for a DualQ Coupled AQM

The following requirements are intended to capture only the essential aspects of a DualQ Coupled AQM. They are intended to be independent of the particular AQMs used for each queue.

### 2.5.1. Functional Requirements

A Dual Queue Coupled AQM implementation **MUST** comply with the prerequisite L4S behaviours for any L4S network node (not just a DualQ) as specified in section 5 of [I-D.ietf-tsvwg-ecn-l4s-id]. These primarily concern classification and remarking as briefly summarized in Section 2.3 earlier. But there is also a subsection (5.5) giving guidance on reducing the burstiness of the link technology underlying any L4S AQM.

A Dual Queue Coupled AQM implementation **MUST** utilize two queues, each with an AQM algorithm.

The AQM algorithm for the low latency (L) queue **MUST** be able to apply ECN marking to ECN-capable packets.

The scheduler draining the two queues **MUST** give L4S packets priority over Classic, although priority **MUST** be bounded in order not to starve Classic traffic (see Section 4.2.2). The scheduler **SHOULD** be work-conserving, or otherwise close to work-conserving. This is because Classic traffic needs to be able to efficiently fill any space left by L4S traffic even though the scheduler would otherwise allocate it to L4S.

[I-D.ietf-tsvwg-ecn-l4s-id] defines the meaning of an ECN marking on L4S traffic, relative to drop of Classic traffic. In order to ensure coexistence of Classic and Scalable L4S traffic, it says, "The likelihood that an AQM drops a Not-ECT Classic packet (p\_C) **MUST** be roughly proportional to the square of the likelihood that it would have marked it if it had been an L4S packet (p\_L)." The term 'likelihood' is used to allow for marking and dropping to be either probabilistic or deterministic.

For the current specification, this translates into the following requirement. A DualQ Coupled AQM MUST apply ECN marking to traffic in the L queue that is no lower than that derived from the likelihood of drop (or ECN marking) in the Classic queue using Eqn. (1).

The constant of proportionality,  $k$ , in Eqn (1) determines the relative flow rates of Classic and L4S flows when the AQM concerned is the bottleneck (all other factors being equal). The L4S ECN protocol [I-D.ietf-tsvwg-ecn-l4s-id] says, "The constant of proportionality ( $k$ ) does not have to be standardised for interoperability, but a value of 2 is RECOMMENDED."

Assuming Scalable congestion controls for the Internet will be as aggressive as DCTCP, this will ensure their congestion window will be roughly the same as that of a standards track TCP Reno congestion control (Reno) [RFC5681] and other Reno-friendly controls, such as TCP Cubic in its Reno-compatibility mode.

The choice of  $k$  is a matter of operator policy, and operators MAY choose a different value using the guidelines in Appendix C.2.

If multiple customers or users share capacity at a bottleneck (e.g. in the Internet access link of a campus network), the operator's choice of  $k$  will determine capacity sharing between the flows of different customers. However, on the public Internet, access network operators typically isolate customers from each other with some form of layer-2 multiplexing (OFDM(A) in DOCSIS3.1, CDMA in 3G, SC-FDMA in LTE) or L3 scheduling (WRR in DSL), rather than relying on host congestion controls to share capacity between customers [RFC0970]. In such cases, the choice of  $k$  will solely affect relative flow rates within each customer's access capacity, not between customers. Also,  $k$  will not affect relative flow rates at any times when all flows are Classic or all flows are L4S, and it will not affect the relative throughput of small flows.

#### 2.5.1.1. Requirements in Unexpected Cases

The flexibility to allow operator-specific classifiers (Section 2.3) leads to the need to specify what the AQM in each queue ought to do with packets that do not carry the ECN field expected for that queue. It is expected that the AQM in each queue will inspect the ECN field to determine what sort of congestion notification to signal, then it will decide whether to apply congestion notification to this particular packet, as follows:

- \* If a packet that does not carry an ECT(1) or CE codepoint is classified into the L queue:

- if the packet is ECT(0), the L AQM SHOULD apply CE-marking using a probability appropriate to Classic congestion control and appropriate to the target delay in the L queue
- if the packet is Not-ECT, the appropriate action depends on whether some other function is protecting the L queue from misbehaving flows (e.g. per-flow queue protection [I-D.briscoe-docsis-q-protection] or latency policing):
  - o If separate queue protection is provided, the L AQM SHOULD ignore the packet and forward it unchanged, meaning it should not calculate whether to apply congestion notification and it should neither drop nor CE-mark the packet (for instance, the operator might classify EF traffic that is unresponsive to drop into the L queue, alongside responsive L4S-ECN traffic)
  - o if separate queue protection is not provided, the L AQM SHOULD apply drop using a drop probability appropriate to Classic congestion control and appropriate to the target delay in the L queue
- \* If a packet that carries an ECT(1) codepoint is classified into the C queue:
  - the C AQM SHOULD apply CE-marking using the coupled AQM probability  $p_{CL}$  ( $= k \cdot p'$ ).

The above requirements are worded as "SHOULDs", because operator-specific classifiers are for flexibility, by definition. Therefore, alternative actions might be appropriate in the operator's specific circumstances. An example would be where the operator knows that certain legacy traffic marked with one codepoint actually has a congestion response associated with another codepoint.

If the DualQ Coupled AQM has detected overload, it MUST introduce Classic drop to both types of ECN-capable traffic until the overload episode has subsided. Introducing drop if ECN marking is persistently high is recommended by Section 7 of the ECN specification [RFC3168] and Section 4.2.1 of the AQM Recommendations [RFC7567].

#### 2.5.2. Management Requirements



### 2.5.2.1. Configuration

By default, a DualQ Coupled AQM SHOULD NOT need any configuration for use at a bottleneck on the public Internet [RFC7567]. The following parameters MAY be operator-configurable, e.g. to tune for non-Internet settings:

- \* Optional packet classifier(s) to use in addition to the ECN field (see Section 2.3);
- \* Expected typical RTT, which can be used to determine the queuing delay of the Classic AQM at its operating point, in order to prevent typical lone flows from under-utilizing capacity. For example:
  - for the PI2 algorithm (Appendix A) the queuing delay target is dependent on the typical RTT;
  - for the Curvy RED algorithm (Appendix B) the queuing delay at the desired operating point of the curvy ramp is configured to encompass a typical RTT;
  - if another Classic AQM was used, it would be likely to need an operating point for the queue based on the typical RTT, and if so it SHOULD be expressed in units of time.

An operating point that is manually calculated might be directly configurable instead, e.g. for links with large numbers of flows where under-utilization by a single flow would be unlikely.

- \* Expected maximum RTT, which can be used to set the stability parameter(s) of the Classic AQM. For example:
  - for the PI2 algorithm (Appendix A), the gain parameters of the PI algorithm depend on the maximum RTT.
  - for the Curvy RED algorithm (Appendix B) the smoothing parameter is chosen to filter out transients in the queue within a maximum RTT.

Stability parameter(s) that are manually calculated assuming a maximum RTT might be directly configurable instead.

- \* Coupling factor,  $k$  (see Appendix C.2);
- \* A limit to the conditional priority of L4S. This is scheduler-dependent, but it SHOULD be expressed as a relation between the max delay of a C packet and an L packet. For example:

- for a WRR scheduler a weight ratio between L and C of  $w:1$  means that the maximum delay to a C packet is  $w$  times that of an L packet.
  - for a time-shifted FIFO (TS-FIFO) scheduler (see Section 4.2.2) a time-shift of  $tshift$  means that the maximum delay to a C packet is  $tshift$  greater than that of an L packet.  $tshift$  could be expressed as a multiple of the typical RTT rather than as an absolute delay.
- \* The maximum Classic ECN marking probability,  $p_{Cmax}$ , before introducing drop.

#### 2.5.2.2. Monitoring

An experimental DualQ Coupled AQM SHOULD allow the operator to monitor each of the following operational statistics on demand, per queue and per configurable sample interval, for performance monitoring and perhaps also for accounting in some cases:

- \* Bits forwarded, from which utilization can be calculated;
- \* Total packets in the three categories: arrived, presented to the AQM, and forwarded. The difference between the first two will measure any non-AQM tail discard. The difference between the last two will measure proactive AQM discard;
- \* ECN packets marked, non-ECN packets dropped, ECN packets dropped, which can be combined with the three total packet counts above to calculate marking and dropping probabilities;
- \* Queue delay (not including serialization delay of the head packet or medium acquisition delay) - see further notes below.

Unlike the other statistics, queue delay cannot be captured in a simple accumulating counter. Therefore the type of queue delay statistics produced (mean, percentiles, etc.) will depend on implementation constraints. To facilitate comparative evaluation of different implementations and approaches, an implementation SHOULD allow mean and 99th percentile queue delay to be derived (per queue per sample interval). A relatively simple way to do this would be to store a coarse-grained histogram of queue delay. This could be done with a small number of bins with configurable edges that represent contiguous ranges of queue delay. Then, over a sample interval, each bin would accumulate a count of the number of packets that had fallen within each range. The maximum queue delay per queue per interval MAY also be recorded, to aid diagnosis of faults and anomalous events.

#### 2.5.2.3. Anomaly Detection

An experimental DualQ Coupled AQM SHOULD asynchronously report the following data about anomalous conditions:

- \* Start-time and duration of overload state.

A hysteresis mechanism SHOULD be used to prevent flapping in and out of overload causing an event storm. For instance, exit from overload state could trigger one report, but also latch a timer. Then, during that time, if the AQM enters and exits overload state any number of times, the duration in overload state is accumulated but no new report is generated until the first time the AQM is out of overload once the timer has expired.

#### 2.5.2.4. Deployment, Coexistence and Scaling

[RFC5706] suggests that deployment, coexistence and scaling should also be covered as management requirements. The raison d'être of the DualQ Coupled AQM is to enable deployment and coexistence of Scalable congestion controls - as incremental replacements for today's Reno-friendly controls that do not scale with bandwidth-delay product. Therefore there is no need to repeat these motivating issues here given they are already explained in the Introduction and detailed in the L4S architecture [I-D.ietf-tsvwg-l4s-arch].

The descriptions of specific DualQ Coupled AQM algorithms in the appendices cover scaling of their configuration parameters, e.g. with respect to RTT and sampling frequency.

### 3. IANA Considerations (to be removed by RFC Editor)

This specification contains no IANA considerations.

### 4. Security Considerations

#### 4.1. Low Delay without Requiring Per-Flow Processing

The L4S architecture [I-D.ietf-tsvwg-l4s-arch] compares the DualQ and per-flow-queuing (FQ) approaches to L4S. The privacy considerations section in that document motivates the DualQ on the grounds that users who want to encrypt application flow identifiers, e.g. in IPSec or other encrypted VPN tunnels, don't have to sacrifice low delay ([RFC8404] encourages avoidance of such privacy compromises).

The security considerations section of the L4S architecture also includes subsections on policing of relative flow-rates (section 8.1) and on policing of flows that cause excessive queuing delay (section 8.2). It explains that the interests of users do not collide in the same way for delay as they do for bandwidth. For someone to get more of the bandwidth of a shared link, someone else necessarily gets less (a 'zero-sum game'), whereas queuing delay can be reduced for everyone, without any need for someone else to lose out. It also explains that, on the current Internet, scheduling usually enforces separation between 'sites' (e.g. households, businesses or mobile users), but it is not common to need to schedule or police individual application flows.

By the above arguments, per-flow policing might not be necessary and in trusted environments it is certainly unlikely to be needed. Therefore, because it is hard to avoid complexity and unintended side-effects with per-flow policing, it needs to be separable from a basic AQM, as an option, under policy control. On this basis, the DualQ Coupled AQM provides low delay without prejudging the question of per-flow policing.

Nonetheless, the interests of users or flows might conflict, e.g. in case of accident or malice. Then per-flow control could be necessary. If flow-rate control is needed, it can be provided as a modular addition to a DualQ. And similarly, if protection against excessive queue delay is needed, a per-flow queue protection option can be added to a DualQ (e.g. [I-D.briscoe-docsis-q-protection]).

#### 4.2. Handling Unresponsive Flows and Overload

In the absence of any per-flow control, it is important that the basic DualQ Coupled AQM gives unresponsive flows no more throughput advantage than a single-queue AQM would, and that it at least handles overload situations. Overload means that incoming load significantly or persistently exceeds output capacity, but it is not intended to be a precise term -- significant and persistent are matters of degree.

A trade-off needs to be made between complexity and the risk of either traffic class harming the other. In overloaded conditions the higher priority L4S service will have to sacrifice some aspect of its performance. Depending on the degree of overload, alternative solutions may relax a different factor: e.g. throughput, delay, drop. These choices need to be made either by the developer or by operator policy, rather than by the IETF. Subsequent subsections discuss aspects relating to handling of different degrees of overload:

- \* Unresponsive flows (L and/or C) but not overloaded, i.e. the sum of unresponsive load before adding any responsive traffic is below capacity;

This case is handled by the regular Coupled DualQ (Section 2.1) but not discussed there. So below, Section 4.2.1 explains the design goal, and how it is achieved in practice;

- \* Unresponsive flows (L and/or C) causing persistent overload, i.e. the sum of unresponsive load even before adding any responsive traffic persistently exceeds capacity;

This case is not covered by the regular Coupled DualQ mechanism (Section 2.1) but the last para in Section 2.5.1.1 sets out a requirement to handle the case where ECN-capable traffic could starve non-ECN-capable traffic. Section 4.2.3 below discusses the general options and gives specific examples.

- \* Short-term overload that lies between the 'not overloaded' and 'persistently overloaded' cases.

For the period before overload is deemed persistent, Section 4.2.2 discusses options for more immediate mechanisms at the scheduler timescale. These prevent short-term starvation of the C queue by making the priority of the L queue conditional, as required in Section 2.5.1.

#### 4.2.1. Unresponsive Traffic without Overload

When one or more L flows and/or C flows are unresponsive, but their total load is within the link capacity so that they do not saturate the coupled marking (below 100%), the goal of a DualQ AQM is to behave no worse than a single-queue AQM.

Tests have shown that this is indeed the case with no additional mechanism beyond the regular Coupled DualQ of Section 2.1 (see the results of 'overload experiments' in [DCttH19]). Perhaps counter-intuitively, whether the unresponsive flow classifies itself into the L or the C queue, the DualQ system behaves as if it has subtracted from the overall link capacity. Then, the coupling shares out the remaining capacity between any competing responsive flows (in either queue). See also Section 4.2.2, which discusses scheduler-specific details.

#### 4.2.2. Avoiding Short-Term Classic Starvation: Sacrifice L4S Throughput or Delay?

Priority of L4S is required to be conditional (see Section 2.4 & Section 2.5.1) to avoid short-term starvation of Classic. Otherwise, as explained in Section 2.4, even a lone responsive L4S flow could temporarily block a small finite set of C packets (e.g. an initial window or DNS request). The blockage would only be brief, but it could be longer for certain AQM implementations that can only increase the congestion signal coupled from the C queue when C packets are actually being dequeued. There is then the question of whether to sacrifice L4S throughput or L4S delay (or some other policy) to make the priority conditional:

**Sacrifice L4S throughput:** By using weighted round robin as the conditional priority scheduler, the L4S service can sacrifice some throughput during overload. This can either be thought of as guaranteeing a minimum throughput service for Classic traffic, or as guaranteeing a maximum delay for a packet at the head of the Classic queue.

Cautionary note: a WRR scheduler can only guarantee Classic throughput if Classic sources are sending enough to use it -- congestion signals can undermine scheduling because they determine how much responsive traffic of each class arrives for scheduling in the first place. This is why scheduling is only relied on to handle short-term starvation; until congestion signals build up and the sources react. Even during long-term overload (discussed more fully in Section 4.2.3), it's pragmatic to discard packets from both queues, which again thins the traffic before it reaches the scheduler. This is because a scheduler cannot be relied on to handle long-term overload since the right scheduler weight cannot be known for every scenario.

The scheduling weight of the Classic queue should be small (e.g. 1/16). In most traffic scenarios the scheduler will not interfere and it will not need to, because the coupling mechanism and the end-systems will determine the share of capacity across both queues as if it were a single pool. However, if L4S traffic is over-aggressive or unresponsive, the scheduler weight for Classic traffic will at least be large enough to ensure it does not starve in the short-term.

Although WRR scheduling is only expected to address short-term overload, there are (somewhat rare) cases when WRR has an effect on capacity shares over longer time-scales. But its effect is minor, and it certainly does no harm. Specifically, in cases where the ratio of L4S to Classic flows (e.g. 19:1) is greater

than the ratio of their scheduler weights (e.g. 15:1), the L4S flows will get less than an equal share of the capacity, but only slightly. For instance, with the example numbers given, each L4S flow will get  $(15/16)/19 = 4.9\%$  when ideally each would get  $1/20=5\%$ . In the rather specific case of an unresponsive flow taking up just less than the capacity set aside for L4S (e.g. 14/16 in the above example), using WRR could significantly reduce the capacity left for any responsive L4S flows.

The scheduling weight of the Classic queue should not be too small, otherwise a C packet at the head of the queue could be excessively delayed by a continually busy L queue. For instance if the Classic weight is 1/16, the maximum that a Classic packet at the head of the queue can be delayed by L traffic is the serialization delay of 15 MTU-sized packets.

**Sacrifice L4S Delay:** The operator could choose to control overload of the Classic queue by allowing some delay to 'leak' across to the L4S queue. The scheduler can be made to behave like a single First-In First-Out (FIFO) queue with different service times by implementing a very simple conditional priority scheduler that could be called a "time-shifted FIFO" (see the Modifier Earliest Deadline First (MEDF) scheduler [MEDF]). This scheduler adds  $t_{\text{shift}}$  to the queue delay of the next L4S packet, before comparing it with the queue delay of the next Classic packet, then it selects the packet with the greater adjusted queue delay.

Under regular conditions, this time-shifted FIFO scheduler behaves just like a strict priority scheduler. But under moderate or high overload it prevents starvation of the Classic queue, because the time-shift ( $t_{\text{shift}}$ ) defines the maximum extra queuing delay of Classic packets relative to L4S. This would control milder overload of responsive traffic by introducing delay to defer invoking the overload mechanisms in Section 4.2.3, particularly when close to the maximum congestion signal.

The example implementations in Appendix A and Appendix B could both be implemented with either policy.

#### 4.2.3. L4S ECN Saturation: Introduce Drop or Delay?

This section concerns persistent overload caused by unresponsive L and/or C flows. To keep the throughput of both L4S and Classic flows roughly equal over the full load range, a different control strategy needs to be defined above the point where the L4S AQM persistently saturates to an ECN marking probability of 100% leaving no room to push back the load any harder. L4S ECN marking will saturate first (assuming the coupling factor  $k>1$ ), even though saturation could be

caused by the sum of unresponsive traffic in either or both queues exceeding the link capacity.

The term 'unresponsive' includes cases where a flow becomes temporarily unresponsive, for instance, a real-time flow that takes a while to adapt its rate in response to congestion, or a standard Reno flow that is normally responsive, but above a certain congestion level it will not be able to reduce its congestion window below the allowed minimum of 2 segments [RFC5681], effectively becoming unresponsive. (Note that L4S traffic ought to remain responsive below a window of 2 segments (see the L4S requirements [I-D.ietf-tsvwg-ecn-l4s-id])).

Saturation raises the question of whether to relieve congestion by introducing some drop into the L4S queue or by allowing delay to grow in both queues (which could eventually lead to drop due to buffer exhaustion anyway):

Drop on Saturation: Persistent saturation can be defined by a maximum threshold for coupled L4S ECN marking (assuming  $k > 1$ ) before saturation starts to make the flow rates of the different traffic types diverge. Above that, the drop probability of Classic traffic is applied to all packets of all traffic types. Then experiments have shown that queueing delay can be kept at the target in any overload situation, including with unresponsive traffic, and no further measures are required (Section 4.2.3.1).

Delay on Saturation: When L4S marking saturates, instead of introducing L4S drop, the drop and marking probabilities of both queues could be capped. Beyond that, delay will grow either solely in the queue with unresponsive traffic (if WRR is used), or in both queues (if time-shifted FIFO is used). In either case, the higher delay ought to control temporary high congestion. If the overload is more persistent, eventually the combined DualQ will overflow and tail drop will control congestion.

The example implementation in Appendix A solely applies the "drop on saturation" policy. The DOCSIS specification of a DualQ Coupled AQM [DOCSIS3.1] also implements the 'drop on saturation' policy with a very shallow L buffer. However, the addition of DOCSIS per-flow Queue Protection [I-D.briscoe-docsis-q-protection] turns this into 'delay on saturation' by redirecting some packets of the flow(s) most responsible for L queue overload into the C queue, which has a higher delay target. If overload continues, this again becomes 'drop on saturation' as the level of drop in the C queue rises to maintain the target delay of the C queue.



#### 4.2.3.1. Protecting against Overload by Unresponsive ECN-Capable Traffic

Without a specific overload mechanism, unresponsive traffic would have a greater advantage if it were also ECN-capable. The advantage is undetectable at normal low levels of marking. However, it would become significant with the higher levels of marking typical during overload, when it could evade a significant degree of drop. This is an issue whether the ECN-capable traffic is L4S or Classic.

This raises the question of whether and when to introduce drop of ECN-capable traffic, as required by both Section 7 of the ECN spec [RFC3168] and Section 4.2.1 of the AQM recommendations [RFC7567].

As an example, experiments with the DualPI2 AQM (Appendix A) have shown that introducing 'drop on saturation' at 100% coupled L4S marking addresses this problem with unresponsive ECN as well as addressing the saturation problem. At saturation, DualPI2 switches into overload mode, where the base AQM is driven by the max delay of both queues and it introduces probabilistic drop to both queues equally. It leaves only a small range of congestion levels just below saturation where unresponsive traffic gains any advantage from using the ECN capability (relative to being unresponsive without ECN), and the advantage is hardly detectable (see [DualQ-Test] and section IV-E of [DCttH19]). Also overload with an unresponsive ECT(1) flow gets no more bandwidth advantage than with ECT(0).

### 5. Acknowledgements

Thanks to Anil Agarwal, Sowmini Varadhan's, Gabi Bracha, Nicolas Kuhn, Greg Skinner, Tom Henderson, David Pullen, Mirja Kuehlewind, Gorrry Fairhurst, Pete Heist and Ermin Sakic for detailed review comments particularly of the appendices and suggestions on how to make the explanations clearer. Thanks also to Tom Henderson for insights on the choice of schedulers and queue delay measurement techniques.

The early contributions of Koen De Schepper, Bob Briscoe, Olga Bondarenko and Inton Tsang were part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). Contributions of Koen De Schepper and Olivier Tilmans were also part-funded by the 5Growth and DAEMON EU H2020 projects. Bob Briscoe's contribution was also part-funded by the Comcast Innovation Fund and the Research Council of Norway through the TimeIn project. The views expressed here are solely those of the authors.

## 6. Contributors

The following contributed implementations and evaluations that validated and helped to improve this specification:

Olga Albisser <olga@albisser.org> of Simula Research Lab, Norway (Olga Bondarenko during early drafts) implemented the prototype DualPI2 AQM for Linux with Koen De Schepper and conducted extensive evaluations as well as implementing the live performance visualization GUI [L4Sdemo16].

Olivier Tilmans <olivier.tilmans@nokia-bell-labs.com> of Nokia Bell Labs, Belgium prepared and maintains the Linux implementation of DualPI2 for upstreaming.

Shravya K.S. wrote a model for the ns-3 simulator based on the -01 version of this Internet-Draft. Based on this initial work, Tom Henderson <tomh@tomh.org> updated that earlier model and created a model for the DualQ variant specified as part of the Low Latency DOCSIS specification, as well as conducting extensive evaluations.

Ing Jyh (Inton) Tsang of Nokia, Belgium built the End-to-End Data Centre to the Home broadband testbed on which DualQ Coupled AQM implementations were tested.

## 7. References

### 7.1. Normative References

- [I-D.ietf-tsvwg-ecn-l4s-id]  
Schepper, K. D. and B. Briscoe, "Explicit Congestion Notification (ECN) Protocol for Very Low Queuing Delay (L4S)", Work in Progress, Internet-Draft, draft-ietf-tsvwg-ecn-l4s-id-25, 4 March 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-ecn-l4s-id-25>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.

- [RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <<https://www.rfc-editor.org/info/rfc8311>>.

## 7.2. Informative References

- [Alizadeh-stability] Alizadeh, M., Javanmard, A., and B. Prabhakar, "Analysis of DCTCP: Stability, Convergence, and Fairness", ACM SIGMETRICS 2011, June 2011, <<https://dl.acm.org/citation.cfm?id=1993753>>.
- [AQMetrics] Kwon, M. and S. Fahmy, "A Comparison of Load-based and Queue-based Active Queue Management Algorithms", Proc. Int'l Soc. for Optical Engineering (SPIE) 4866:35--46 DOI: 10.1117/12.473021, 2002, <<https://www.cs.purdue.edu/homes/fahmy/papers/ldc.pdf>>.
- [ARED01] Floyd, S., Gummadi, R., and S. Shenker, "Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management", ACIRI Technical Report, August 2001, <<http://www.icir.org/floyd/red.html>>.
- [BBRv2] Cardwell, N., "BRTCP BBR v2 Alpha/Preview Release", github repository; Linux congestion control module, <<https://github.com/google/bbr/blob/v2alpha/README.md>>.
- [Boru20] Boru Oljira, D., Grinnemo, K-J., Brunstrom, A., and J. Taheri, "Validating the Sharing Behavior and Latency Characteristics of the L4S Architecture", ACM CCR 50(2):37--44, May 2020, <<https://dl.acm.org/doi/abs/10.1145/3402413.3402419>>.
- [CCcensus19] Mishra, A., Sun, X., Jain, A., Pande, S., Joshi, R., and B. Leong, "The Great Internet TCP Congestion Control Census", Proc. ACM on Measurement and Analysis of Computing Systems 3(3), December 2019, <<https://doi.org/10.1145/3366693>>.
- [CoDel] Nichols, K. and V. Jacobson, "Controlling Queue Delay", ACM Queue 10(5), May 2012, <<http://queue.acm.org/issuedetail.cfm?issue=2208917>>.

- [CRED\_Insights] Briscoe, B., "Insights from Curvy RED (Random Early Detection)", BT Technical Report TR-TUB8-2015-003 arXiv:1904.07339 [cs.NI], July 2015, <<https://arxiv.org/abs/1904.07339>>.
- [DCttH19] De Schepper, K., Bondarenko, O., Tilmans, O., and B. Briscoe, "'Data Centre to the Home': Ultra-Low Latency for All", Updated RITE project Technical Report , July 2019, <[https://bobbbriscoe.net/pubs.html#DCttH\\_TR](https://bobbbriscoe.net/pubs.html#DCttH_TR)>.
- [DOCSIS3.1] CableLabs, "MAC and Upper Layer Protocols Interface (MULPI) Specification, CM-SP-MULPIv3.1", Data-Over-Cable Service Interface Specifications DOCSIS® 3.1 Version i17 or later, 21 January 2019, <<https://specification-search.cablelabs.com/CM-SP-MULPIv3.1>>.
- [DualPI2Linux] Albisser, O., De Schepper, K., Briscoe, B., Tilmans, O., and H. Steen, "DUALPI2 - Low Latency, Low Loss and Scalable (L4S) AQM", Proc. Linux Netdev 0x13 , March 2019, <<https://www.netdevconf.org/0x13/session.html?talk-DUALPI2-AQM>>.
- [DualQ-Test] Steen, H., "Destruction Testing: Ultra-Low Delay using Dual Queue Coupled Active Queue Management", Masters Thesis, Dept of Informatics, Uni Oslo , May 2017, <<https://www.duo.uio.no/bitstream/handle/10852/57424/thesis-henrste.pdf?sequence=1>>.
- [Heist21] Heist, P. and J. Morton, "L4S Tests", github README, August 2021, <<https://github.com/heistp/l4s-tests/#underutilization-with-bursty-traffic>>.
- [I-D.briscoe-docsis-q-protection] Briscoe, B. and G. White, "The DOCSIS(r) Queue Protection Algorithm to Preserve Low Latency", Work in Progress, Internet-Draft, draft-briscoe-docsis-q-protection-03, 7 March 2022, <<https://datatracker.ietf.org/doc/html/draft-briscoe-docsis-q-protection-03>>.

- [I-D.briscoe-iccrp-prague-congestion-control]  
Schepper, K. D., Tilmans, O., and B. Briscoe, "Prague Congestion Control", Work in Progress, Internet-Draft, draft-briscoe-iccrp-prague-congestion-control-00, 9 March 2021, <<https://datatracker.ietf.org/doc/html/draft-briscoe-iccrp-prague-congestion-control-00>>.
- [I-D.briscoe-tsvwg-l4s-diffserv]  
Briscoe, B., "Interactions between Low Latency, Low Loss, Scalable Throughput (L4S) and Differentiated Services", Work in Progress, Internet-Draft, draft-briscoe-tsvwg-l4s-diffserv-02, 4 November 2018, <<https://datatracker.ietf.org/doc/html/draft-briscoe-tsvwg-l4s-diffserv-02>>.
- [I-D.cardwell-iccrp-bbr-congestion-control]  
Cardwell, N., Cheng, Y., Yeganeh, S. H., Swett, I., and V. Jacobson, "BBR Congestion Control", Work in Progress, Internet-Draft, draft-cardwell-iccrp-bbr-congestion-control-02, 7 March 2022, <<https://datatracker.ietf.org/doc/html/draft-cardwell-iccrp-bbr-congestion-control-02>>.
- [I-D.ietf-tsvwg-l4s-arch]  
Briscoe, B., Schepper, K. D., Bagnulo, M., and G. White, "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture", Work in Progress, Internet-Draft, draft-ietf-tsvwg-l4s-arch-17, 4 March 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-l4s-arch-17>>.
- [L4Sdemo16]  
Bondarenko, O., De Schepper, K., Tsang, I., and B. Briscoe, "Ultra-Low Delay for All: Live Experience, Live Analysis", Proc. MMSYS'16 pp33:1--33:4, May 2016, <<http://dl.acm.org/citation.cfm?doid=2910017.2910633>> (videos of demos: <https://riteproject.eu/dctth/#1511dispatchwg> )>.
- [L4S\_5G]  
Willars, P., Wittenmark, E., Ronkainen, H., Östberg, C., Johansson, I., Strand, J., Lédl, P., and D. Schnieders, "Enabling time-critical applications over 5G with rate adaptation", Ericsson - Deutsche Telekom White Paper BNEW-21:025455 Uen, May 2021, <<https://www.ericsson.com/en/reports-and-papers/white-papers/enabling-time-critical-applications-over-5g-with-rate-adaptation>>.

- [Labovitz10] Labovitz, C., Iekel-Johnson, S., McPherson, D., Oberheide, J., and F. Jahanian, "Internet Inter-Domain Traffic", Proc ACM SIGCOMM; ACM CCR 40(4):75--86, August 2010, <<https://doi.org/10.1145/1851275.1851194>>.
- [LLD] White, G., Sundaresan, K., and B. Briscoe, "Low Latency DOCSIS: Technology Overview", CableLabs White Paper , February 2019, <<https://cablelabs.com/low-latency-docsis-technology-overview-february-2019>>.
- [Mathis09] Mathis, M., "Relentless Congestion Control", PFLDNet'09 , May 2009, <[http://www.hpcc.jp/pfldnet2009/Program\\_files/1569198525.pdf](http://www.hpcc.jp/pfldnet2009/Program_files/1569198525.pdf)>.
- [MEDF] Menth, M., Schmid, M., Heiss, H., and T. Reim, "MEDF - a simple scheduling algorithm for two real-time transport service classes with application in the UTRAN", Proc. IEEE Conference on Computer Communications (INFOCOM'03) Vol.2 pp.1116-1122, March 2003, <[http://infocom2003.ieee-infocom.org/papers/27\\_04.PDF](http://infocom2003.ieee-infocom.org/papers/27_04.PDF)>.
- [PI2] De Schepper, K., Bondarenko, O., Briscoe, B., and I. Tsang, "PI2: A Linearized AQM for both Classic and Scalable TCP", ACM CoNEXT'16 , December 2016, <[https://riteproject.files.wordpress.com/2015/10/pi2\\_conext.pdf](https://riteproject.files.wordpress.com/2015/10/pi2_conext.pdf)>.
- [PI2param] Briscoe, B., "PI2 Parameters", Technical Report TR-BB-2021-001 arXiv:2107.01003 [cs.NI], July 2021, <<https://arxiv.org/abs/2107.01003>>.
- [PragueLinux] Briscoe, B., De Schepper, K., Albisser, O., Misund, J., Tilmans, O., Kühlewind, M., and A.S. Ahmed, "Implementing the 'TCP Prague' Requirements for Low Latency Low Loss Scalable Throughput (L4S)", Proc. Linux Netdev 0x13 , March 2019, <<https://www.netdevconf.org/0x13/session.html?talk-tcp-prague-l4s>>.
- [RFC0970] Nagle, J., "On Packet Switches With Infinite Storage", RFC 970, DOI 10.17487/RFC0970, December 1985, <<https://www.rfc-editor.org/info/rfc970>>.
- [RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and L. Zhang, "Recommendations on

- Queue Management and Congestion Avoidance in the Internet", RFC 2309, DOI 10.17487/RFC2309, April 1998, <<https://www.rfc-editor.org/info/rfc2309>>.
- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, DOI 10.17487/RFC2914, September 2000, <<https://www.rfc-editor.org/info/rfc2914>>.
- [RFC3246] Davie, B., Charny, A., Bennet, J.C.R., Benson, K., Le Boudec, J.Y., Courtney, W., Davari, S., Firoiu, V., and D. Stiliadis, "An Expedited Forwarding PHB (Per-Hop Behavior)", RFC 3246, DOI 10.17487/RFC3246, March 2002, <<https://www.rfc-editor.org/info/rfc3246>>.
- [RFC3649] Floyd, S., "HighSpeed TCP for Large Congestion Windows", RFC 3649, DOI 10.17487/RFC3649, December 2003, <<https://www.rfc-editor.org/info/rfc3649>>.
- [RFC5033] Floyd, S. and M. Allman, "Specifying New Congestion Control Algorithms", BCP 133, RFC 5033, DOI 10.17487/RFC5033, August 2007, <<https://www.rfc-editor.org/info/rfc5033>>.
- [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <<https://www.rfc-editor.org/info/rfc5348>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/info/rfc5681>>.
- [RFC5706] Harrington, D., "Guidelines for Considering Operations and Management of New Protocols and Protocol Extensions", RFC 5706, DOI 10.17487/RFC5706, November 2009, <<https://www.rfc-editor.org/info/rfc5706>>.
- [RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <<https://www.rfc-editor.org/info/rfc7567>>.
- [RFC8033] Pan, R., Natarajan, P., Baker, F., and G. White, "Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem", RFC 8033, DOI 10.17487/RFC8033, February 2017, <<https://www.rfc-editor.org/info/rfc8033>>.

- [RFC8034] White, G. and R. Pan, "Active Queue Management (AQM) Based on Proportional Integral Controller Enhanced PIE) for Data-Over-Cable Service Interface Specifications (DOCSIS) Cable Modems", RFC 8034, DOI 10.17487/RFC8034, February 2017, <<https://www.rfc-editor.org/info/rfc8034>>.
- [RFC8257] Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <<https://www.rfc-editor.org/info/rfc8257>>.
- [RFC8290] Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm", RFC 8290, DOI 10.17487/RFC8290, January 2018, <<https://www.rfc-editor.org/info/rfc8290>>.
- [RFC8298] Johansson, I. and Z. Sarker, "Self-Clocked Rate Adaptation for Multimedia", RFC 8298, DOI 10.17487/RFC8298, December 2017, <<https://www.rfc-editor.org/info/rfc8298>>.
- [RFC8312] Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", RFC 8312, DOI 10.17487/RFC8312, February 2018, <<https://www.rfc-editor.org/info/rfc8312>>.
- [RFC8404] Moriarty, K., Ed. and A. Morton, Ed., "Effects of Pervasive Encryption on Operators", RFC 8404, DOI 10.17487/RFC8404, July 2018, <<https://www.rfc-editor.org/info/rfc8404>>.
- [SCReAM] Johansson, I., "SCReAM", github repository; , <<https://github.com/EricssonResearch/scream/blob/master/README.md>>.
- [SigQ-Dyn] Briscoe, B., "Rapid Signalling of Queue Dynamics", Technical Report TR-BB-2017-001 arXiv:1904.07044 [cs.NI], September 2017, <<https://arxiv.org/abs/1904.07044>>.

#### Appendix A. Example DualQ Coupled PI2 Algorithm

As a first concrete example, the pseudocode below gives the DualPI2 algorithm. DualPI2 follows the structure of the DualQ Coupled AQM framework in Figure 1. A simple ramp function (configured in units of queuing time) with unsmoothed ECN marking is used for the Native L4S AQM. The ramp can also be configured as a step function. The PI2 algorithm [PI2] is used for the Classic AQM. PI2 is an improved variant of the PIE AQM [RFC8033].



The pseudocode will be introduced in two passes. The first pass explains the core concepts, deferring handling of edge-cases like overload to the second pass. To aid comparison, line numbers are kept in step between the two passes by using letter suffixes where the longer code needs extra lines.

All variables are assumed to be floating point in their basic units (size in bytes, time in seconds, rates in bytes/second, alpha and beta in Hz, and probabilities from 0 to 1. Constants expressed in k (kilo), M (mega), G (giga), u (micro), m (milli) , %, ... are assumed to be converted to their appropriate multiple or fraction to represent the basic units. A real implementation that wants to use integer values needs to handle appropriate scaling factors and allow accordingly appropriate resolution of its integer types (including temporary internal values during calculations).

A full open source implementation for Linux is available at: [https://github.com/L4STeam/sch\\_dualpi2\\_upstream](https://github.com/L4STeam/sch_dualpi2_upstream) and explained in [DualPI2Linux]. The specification of the DualQ Coupled AQM for DOCSIS cable modems and CMTSS is available in [DOCSIS3.1] and explained in [LLD].

#### A.1. Pass #1: Core Concepts

The pseudocode manipulates three main structures of variables: the packet (pkt), the L4S queue (lq) and the Classic queue (cq). The pseudocode consists of the following six functions:

- \* The initialization function `dualpi2_params_init(...)` (Figure 2) that sets parameter defaults (the API for setting non-default values is omitted for brevity)
- \* The enqueue function `dualpi2_enqueue(lq, cq, pkt)` (Figure 3)
- \* The dequeue function `dualpi2_dequeue(lq, cq, pkt)` (Figure 4)
- \* The recurrence function `recur(q, likelihood)` for de-randomized ECN marking (shown at the end of Figure 4).
- \* The L4S AQM function `laqm(qdelay)` (Figure 5) used to calculate the ECN-marking probability for the L4S queue
- \* The base AQM function that implements the PI algorithm `dualpi2_update(lq, cq)` (Figure 6) used to regularly update the base probability ( $p'$ ), which is squared for the Classic AQM as well as being coupled across to the L4S queue.

It also uses the following functions that are not shown in full here:

- \* `scheduler()`, which selects between the head packets of the two queues; the choice of scheduler technology is discussed later;
- \* `cq.bytest()` or `lq.bytest()` returns the current length (aka. backlog) of the relevant queue in bytes;
- \* `cq.len()` or `lq.len()` returns the current length of the relevant queue in packets;
- \* `cq.time()` or `lq.time()` returns the current queuing delay (aka. sojourn time or service time) of the relevant queue in units of time (see Note a);
- \* `mark(pkt)` and `drop(pkt)` for ECN-marking and dropping a packet;

In experiments so far (building on experiments with PIE) on broadband access links ranging from 4 Mb/s to 200 Mb/s with base RTTs from 5 ms to 100 ms, DualPI2 achieves good results with the default parameters in Figure 2. The parameters are categorised by whether they relate to the Base PI2 AQM, the L4S AQM or the framework coupling them together. Constants and variables derived from these parameters are also included at the end of each category. Each parameter is explained as it is encountered in the walk-through of the pseudocode below, and the rationale for the chosen defaults are given so that sensible values can be used in scenarios other than the regular public Internet.

```

1:  dualpi2_params_init(...) {           % Set input parameter defaults
2:      % DualQ Coupled framework parameters
5:      limit = MAX_LINK_RATE * 250 ms    % Dual buffer size
3:      k = 2                             % Coupling factor
4:      % NOT SHOWN % scheduler-dependent weight or equival't parameter
6:
7:      % PI2 Classic AQM parameters
8:      target = 15 ms                     % Queue delay target
9:      RTT_max = 100 ms                   % Worst case RTT expected
10:     % PI2 constants derived from above PI2 parameters
11:     p_Cmax = min(1/k^2, 1)              % Max Classic drop/mark prob
12:     Tupdate = min(target, RTT_max/3)    % PI sampling interval
13:     alpha = 0.1 * Tupdate / RTT_max^2   % PI integral gain in Hz
14:     beta = 0.3 / RTT_max                % PI proportional gain in Hz
15:
16:     % L4S ramp AQM parameters
17:     minTh = 800 us                      % L4S min marking threshold in time units
18:     range = 400 us                      % Range of L4S ramp in time units
19:     Th_len = 1 pkt                      % Min L4S marking threshold in packets
20:     % L4S constants
21:     p_Lmax = 1                          % Max L4S marking prob
22: }
```

Figure 2: Example Header Pseudocode for DualQ Coupled PI2 AQM

The overall goal of the code is to apply the marking and dropping probabilities for L4S and Classic traffic ( $p_L$  and  $p_C$ ). These are derived from the underlying base probabilities  $p'_L$  and  $p'$  driven respectively by the traffic in the L and C queues. The marking probability for the L queue ( $p_L$ ) depends on both the base probability in its own queue ( $p'_L$ ) and a probability called  $p_{CL}$ , which is coupled across from  $p'$  in the C queue (see Section 2.4 for the derivation of the specific equations and dependencies).

The probabilities  $p_{CL}$  and  $p_C$  are derived in lines 4 and 5 of the `dualpi2_update()` function (Figure 6) then used in the `dualpi2_dequeue()` function where  $p_L$  is also derived from  $p_{CL}$  at line 6 (Figure 4). The code walk-through below builds up to explaining that part of the code eventually, but it starts from packet arrival.

```

1: dualpi2_enqueue(lq, cq, pkt) { % Test limit and classify lq or cq
2:   if ( lq.bytest() + cq.bytest() + MTU > limit)
3:     drop(pkt) % drop packet if buffer is full
4:   timestamp(pkt) % attach arrival time to packet
5:   % Packet classifier
6:   if ( ecn(pkt) modulo 2 == 1 ) % ECN bits = ECT(1) or CE
7:     lq.enqueue(pkt)
8:   else % ECN bits = not-ECT or ECT(0)
9:     cq.enqueue(pkt)
10: }

```

Figure 3: Example Enqueue Pseudocode for DualQ Coupled PI2 AQM

```

1: dualpi2_dequeue(lq, cq, pkt) { % Couples L4S & Classic queues
2:   while ( lq.bytest() + cq.bytest() > 0 ) {
3:     if ( scheduler() == lq ) {
4:       lq.dequeue(pkt) % Scheduler chooses lq
5:       p'_L = laqm(lq.time()) % Native LAQM
6:       p_L = max(p'_L, p_CL) % Combining function
7:       if ( recur(lq, p_L) ) % Linear marking
8:         mark(pkt)
9:     } else {
10:      cq.dequeue(pkt) % Scheduler chooses cq
11:      if ( recur(cq, p_C) ) { % probability p_C = p'^2
12:        if ( ecn(pkt) == 0 ) { % if ECN field = not-ECT
13:          drop(pkt) % squared drop
14:          continue % continue to the top of the while loop
15:        }
16:        mark(pkt) % squared mark
17:      }
18:    }
19:    return(pkt) % return the packet and stop
20:  }
21:  return(NULL) % no packet to dequeue
22: }

23: recur(q, likelihood) { % Returns TRUE with a certain likelihood
24:   q.count += likelihood
25:   if (q.count > 1) {
26:     q.count -= 1
27:     return TRUE
28:   }
29:   return FALSE
30: }

```

Figure 4: Example Dequeue Pseudocode for DualQ Coupled PI2 AQM

When packets arrive, first a common queue limit is checked as shown in line 2 of the enqueueing pseudocode in Figure 3. This assumes a shared buffer for the two queues (Note b discusses the merits of separate buffers). In order to avoid any bias against larger packets, 1 MTU of space is always allowed and the limit is deliberately tested before enqueue.

If limit is not exceeded, the packet is timestamped in line 4. This assumes that queue delay is measured using the sojourn time technique (see Note a for alternatives).

At lines 5-9, the packet is classified and enqueued to the Classic or L4S queue dependent on the least significant bit of the ECN field in the IP header (line 6). Packets with a codepoint having an LSB of 0 (Not-ECT and ECT(0)) will be enqueued in the Classic queue. Otherwise, ECT(1) and CE packets will be enqueued in the L4S queue. Optional additional packet classification flexibility is omitted for brevity (see the L4S ECN protocol [I-D.ietf-tsvwg-ecn-l4s-id]).

The dequeue pseudocode (Figure 4) is repeatedly called whenever the lower layer is ready to forward a packet. It schedules one packet for dequeuing (or zero if the queue is empty) then returns control to the caller, so that it does not block while that packet is being forwarded. While making this dequeue decision, it also makes the necessary AQM decisions on dropping or marking. The alternative of applying the AQMs at enqueue would shift some processing from the critical time when each packet is dequeued. However, it would also add a whole queue of delay to the control signals, making the control loop sloppier (for a typical RTT it would double the Classic queue's feedback delay).

All the dequeue code is contained within a large while loop so that if it decides to drop a packet, it will continue until it selects a packet to schedule. Line 3 of the dequeue pseudocode is where the scheduler chooses between the L4S queue (lq) and the Classic queue (cq). Detailed implementation of the scheduler is not shown (see discussion later).

- \* If an L4S packet is scheduled, in lines 7 and 8 the packet is ECN-marked with likelihood  $p_L$ . The `recur()` function at the end of Figure 4 is used, which is preferred over random marking because it avoids delay due to randomization when interpreting congestion signals, but it still desynchronizes the saw-teeth of the flows. Line 6 calculates  $p_L$  as the maximum of the coupled L4S probability  $p_{CL}$  and the probability from the native L4S AQM  $p'_L$ . This implements the `max()` function shown in Figure 1 to couple the outputs of the two AQMs together. Of the two probabilities input to  $p_L$  in line 6:

- $p'_L$  is calculated per packet in line 5 by the `laqm()` function (see Figure 5),
  - Whereas  $p_{CL}$  is maintained by the `dualpi2_update()` function which runs every `Tupdate` (`Tupdate` is set in line 12 of Figure 2).
- \* If a Classic packet is scheduled, lines 10 to 17 drop or mark the packet with probability  $p_C$ .

The Native L4S AQM algorithm (Figure 5) is a ramp function, similar to the RED algorithm, but simplified as follows:

- \* The extent of the ramp is defined in units of queuing delay, not bytes, so that configuration remains invariant as the queue departure rate varies.
- \* It uses instantaneous queueing delay, which avoids the complexity of smoothing, but also avoids embedding a worst-case RTT of smoothing delay in the network (see Section 2.1).
- \* The ramp rises linearly directly from 0 to 1, not to an intermediate value of  $p'_L$  as RED would, because there is no need to keep ECN marking probability low.
- \* Marking does not have to be randomized. Determinism is used instead of randomness; to reduce the delay necessary to smooth out the noise of randomness from the signal.

The ramp function requires two configuration parameters, the minimum threshold (`minTh`) and the width of the ramp (`range`), both in units of queuing time, as shown in lines 17 & 18 of the initialization function in Figure 2. The ramp function can be configured as a step (see Note c).

Although the DCTCP paper [Alizadeh-stability] recommends an ECN marking threshold of  $0.17 \cdot \text{RTT}_{\text{typ}}$ , it also shows that the threshold can be much shallower with hardly any worse under-utilization of the link (because the amplitude of DCTCP's sawteeth is so small). Based on extensive experiments, for the public Internet the default minimum ECN marking threshold (`target`) in Figure 2 is considered a good compromise, even though it is significantly smaller fraction of  $\text{RTT}_{\text{typ}}$ .

```

1: laqm(qdelay) {                                % Returns native L4S AQM probability
2:   if (qdelay >= maxTh)
3:     return 1
4:   else if (qdelay > minTh)
5:     return (qdelay - minTh)/range % Divide could use a bit-shift
6:   else
7:     return 0
8: }

```

Figure 5: Example Pseudocode for the Native L4S AQM

```

1: dualpi2_update(lq, cq) {                      % Update p' every Tupdate
2:   curq = cq.time() % use queuing time of first-in Classic packet
3:   p' = p' + alpha * (curq - target) + beta * (curq - prevq)
4:   p_CL = k * p' % Coupled L4S prob = base prob * coupling factor
5:   p_C = p'^2 % Classic prob = (base prob)^2
6:   prevq = curq
7: }

```

Figure 6: Example PI-Update Pseudocode for DualQ Coupled PI2 AQM

(Clamping  $p'$  within the range  $[0,1]$  omitted for clarity - see text)

The coupled marking probability,  $p_{CL}$  depends on the base probability ( $p'$ ), which is kept up to date by the core PI algorithm in Figure 6 executed every Tupdate.

Note that  $p'$  solely depends on the queuing time in the Classic queue. In line 2, the current queuing delay ( $curq$ ) is evaluated from how long the head packet was in the Classic queue ( $cq$ ). The function  $cq.time()$  (not shown) subtracts the time stamped at enqueue from the current time (see Note a) and implicitly takes the current queuing delay as 0 if the queue is empty.

The algorithm centres on line 3, which is a classical Proportional-Integral (PI) controller that alters  $p'$  dependent on: a) the error between the current queuing delay ( $curq$ ) and the target queuing delay, 'target'; and b) the change in queuing delay since the last sample. The name 'PI' represents the fact that the second factor (how fast the queue is growing) is  $\_P\_roportional$  to load while the first is the  $\_I\_ntegral$  of the load (so it removes any standing queue in excess of the target).

The target parameter can be set based on local knowledge, but the aim is for the default to be a good compromise for anywhere in the intended deployment environment -- the public Internet. According to [PI2param], the target queuing delay on line 9 of Figure 2 is related

to the typical base RTT worldwide,  $RTT_{typ}$ , by two factors:  $target = RTT_{typ} * g * f$ . Below we summarize the rationale behind these factors and introduce a further adjustment. The two factors ensure that, in a large proportion of cases (say 90%), the sawtooth variations in RTT of a single flow will fit within the buffer without underutilizing the link. Frankly, these factors are educated guesses, but with the emphasis closer to 'educated' than to 'guess' (see [PI2param] for full background):

- \*  $RTT_{typ}$  is taken as 25 ms. This is based on an average CDN latency measured in each country weighted by the number of Internet users in that country to produce an overall weighted average for the Internet [PI2param]. Countries were ranked by number of Internet users, and once 90% of Internet users were covered, smaller countries were excluded to avoid unrepresentatively small sample sizes. Also, importantly, the data for the average CDN latency in China (with the largest number of Internet users) has been removed, because the CDN latency was a significant outlier and, on reflection, the experimental technique seemed inappropriate to the CDN market in China.
- \*  $g$  is taken as 0.38. The factor  $g$  is a geometry factor that characterizes the shape of the sawteeth of prevalent Classic congestion controllers. The geometry factor is the fraction of the amplitude of the sawtooth variability in queue delay that lies below the AQM's target. For instance, at low bit rate, the geometry factor of standard Reno is 0.5, but at higher rates it tends to just under 1. According to the census of congestion controllers conducted by Mishra *et al.* in Jul-Oct 2019 [CCcensus19], most Classic TCP traffic uses Cubic. And, according to the analysis in [PI2param], if running over a PI2 AQM, a large proportion of this Cubic traffic would be in its Reno-Friendly mode, which has a geometry factor of  $\sim 0.39$  (all known implementations). The rest of the Cubic traffic would be in true Cubic mode, which has a geometry factor of  $\sim 0.36$ . Without modelling the sawtooth profiles from all the other less prevalent congestion controllers, we estimate a 7:3 weighted average of these two, resulting in an average geometry factor of 0.38.
- \*  $f$  is taken as 2. The factor  $f$  is a safety factor that increases the target queue to allow for the distribution of  $RTT_{typ}$  around its mean. Otherwise the target queue would only avoid underutilization for those users below the mean. It also provides a safety margin for the proportion of paths in use that span beyond the distance between a user and their local CDN. Currently no data is available on the variance of queue delay around the mean in each region, so there is plenty of room for this guess to become more educated.



- \* [PI2param] recommends  $\text{target} = \text{RTT\_typ} * g * f = 25\text{ms} * 0.38 * 2 = 19\text{ ms}$ . However a further adjustment is warranted, because target is moving year on year. The paper is based on data collected in 2019, and it mentions evidence from speedtest.net that suggests RTT\_typ reduced by 17% (fixed) or 12% (mobile) between 2020 and 2021. Therefore we recommend a default of  $\text{target} = 15\text{ ms}$  at the time of writing (2021).

Operators can always use the data and discussion in [PI2param] to configure a more appropriate target for their environment. For instance, an operator might wish to question the assumptions called out in that paper, such as the goal of no underutilization for a large majority of single flow transfers (given many large transfers use multiple flows to avoid the scaling limitations of Classic flows).

The two 'gain factors' in line 3 of Figure 6, alpha and beta, respectively weight how strongly each of the two elements (Integral and Proportional) alters  $p'$ . They are in units of 'per second of delay' or Hz, because they transform differences in queueing delay into changes in probability (assuming probability has a value from 0 to 1).

Alpha and beta determine how much  $p'$  ought to change after each update interval (Tupdate). For smaller Tupdate,  $p'$  should change by the same amount per second, but in finer more frequent steps. So alpha depends on Tupdate (see line 13 of the initialization function in Figure 2). It is best to update  $p'$  as frequently as possible, but Tupdate will probably be constrained by hardware performance. As shown in line 13, the update interval should be frequent enough to update at least once in the time taken for the target queue to drain ('target') as long as it updates at least three times per maximum RTT. Tupdate defaults to 16 ms in the reference Linux implementation because it has to be rounded to a multiple of 4 ms. For link rates from 4 to 200 Mb/s and a maximum RTT of 100ms, it has been verified through extensive testing that Tupdate=16ms (as also recommended in the PIE spec [RFC8033]) is sufficient.

The choice of alpha and beta also determines the AQM's stable operating range. The AQM ought to change  $p'$  as fast as possible in response to changes in load without over-compensating and therefore causing oscillations in the queue. Therefore, the values of alpha and beta also depend on the RTT of the expected worst-case flow (RTT\_max).

The maximum RTT of a PI controller (RTT\_max in line 10 of Figure 2) is not an absolute maximum, but more instability (more queue variability) sets in for long-running flows with an RTT above this

value. The propagation delay half way round the planet and back in glass fibre is 200 ms. However, hardly any traffic traverses such extreme paths and, since the significant consolidation of Internet traffic between 2007 and 2009 [Labovitz10], a high and growing proportion of all Internet traffic (roughly two-thirds at the time of writing) has been served from content distribution networks (CDNs) or 'cloud' services distributed close to end-users. The Internet might change again, but for now, designing for a maximum RTT of 100ms is a good compromise between faster queue control at low RTT and some instability on the occasions when a longer path is necessary.

Recommended derivations of the gain constants alpha and beta can be approximated for Reno over a PI2 AQM as:  $\alpha = 0.1 * \text{Tupdate} / \text{RTT\_max}^2$ ;  $\beta = 0.3 / \text{RTT\_max}$ , as shown in lines 14 & 15 of Figure 2. These are derived from the stability analysis in [PI2]. For the default values of Tupdate=16 ms and RTT\_max = 100 ms, they result in  $\alpha = 0.16$ ;  $\beta = 3.2$  (discrepancies are due to rounding). These defaults have been verified with a wide range of link rates, target delays and a range of traffic models with mixed and similar RTTs, short and long flows, etc.

In corner cases,  $p'$  can overflow the range [0,1] so the resulting value of  $p'$  has to be bounded (omitted from the pseudocode). Then, as already explained, the coupled and Classic probabilities are derived from the new  $p'$  in lines 4 and 5 of Figure 6 as  $p_{\text{CL}} = k * p'$  and  $p_{\text{C}} = p'^2$ .

Because the coupled L4S marking probability ( $p_{\text{CL}}$ ) is factored up by  $k$ , the dynamic gain parameters alpha and beta are also inherently factored up by  $k$  for the L4S queue. So, the effective gain factor for the L4S queue is  $k * \alpha$  (with defaults  $\alpha = 0.16 \text{ Hz}$  and  $k=2$ , effective L4S  $\alpha = 0.32 \text{ Hz}$ ).

Unlike in PIE [RFC8033], alpha and beta do not need to be tuned every Tupdate dependent on  $p'$ . Instead, in PI2, alpha and beta are independent of  $p'$  because the squaring applied to Classic traffic tunes them inherently. This is explained in [PI2], which also explains why this more principled approach removes the need for most of the heuristics that had to be added to PIE.

Nonetheless, an implementer might wish to add selected details to either AQM. For instance the Linux reference DualPI2 implementation includes the following (not shown in the pseudocode above):

- \* Classic and coupled marking or dropping (i.e. based on p\_C and p\_CL from the PI controller) is not applied to a packet if the aggregate queue length in bytes is < 2 MTU (prior to enqueueing the packet or dequeuing it, depending on whether the AQM is configured to be applied at enqueue or dequeue);
- \* In the WRR scheduler, the 'credit' indicating which queue should transmit is only changed if there are packets in both queues (i.e. if there is actual resource contention). This means that a properly paced L flow might never be delayed by the WRR. The WRR credit is reset in favour of the L queue when the link is idle.

An implementer might also wish to add other heuristics, e.g. burst protection [RFC8033] or enhanced burst protection [RFC8034].

Notes:

- a. The drain rate of the queue can vary if it is scheduled relative to other queues, or to cater for fluctuations in a wireless medium. To auto-adjust to changes in drain rate, the queue needs to be measured in time, not bytes or packets [AQMetrics], [CoDel]. Queuing delay could be measured directly by storing a per-packet time-stamp as each packet is enqueued, and subtracting this from the system time when the packet is dequeued. If time-stamping is not easy to introduce with certain hardware, queuing delay could be predicted indirectly by dividing the size of the queue by the predicted departure rate, which might be known precisely for some link technologies (see for example in DOCSIS PIE [RFC8034]).
- b. Line 2 of the `dualpi2_enqueue()` function (Figure 3) assumes an implementation where lq and cq share common buffer memory. An alternative implementation could use separate buffers for each queue, in which case the arriving packet would have to be classified first to determine which buffer to check for available space. The choice is a trade off; a shared buffer can use less memory whereas separate buffers isolate the L4S queue from tail-drop due to large bursts of Classic traffic (e.g. a Classic Reno TCP during slow-start over a long RTT).
- c. There has been some concern that using the step function of DCTCP for the Native L4S AQM requires end-systems to smooth the signal for an unnecessarily large number of round trips to ensure sufficient fidelity. A ramp is no worse than a step in initial experiments with existing DCTCP. Therefore, it is recommended that a ramp is configured in place of a step, which will allow congestion control algorithms to investigate faster smoothing algorithms.

A ramp is more general than a step, because an operator can effectively turn the ramp into a step function, as used by DCTCP, by setting the range to zero. There will not be a divide by zero problem at line 5 of Figure 5 because, if minTh is equal to maxTh, the condition for this ramp calculation cannot arise.

#### A.2. Pass #2: Edge-Case Details

This section takes a second pass through the pseudocode adding details of two edge-cases: low link rate and overload. Figure 7 repeats the dequeue function of Figure 4, but with details of both edge-cases added. Similarly Figure 8 repeats the core PI algorithm of Figure 6, but with overload details added. The initialization, enqueue, L4S AQM and recur functions are unchanged.

The link rate can be so low that it takes a single packet queue longer to serialize than the threshold delay at which ECN marking starts to be applied in the L queue. Therefore, a minimum marking threshold parameter in units of packets rather than time is necessary (Th\_len, default 1 packet in line 19 of Figure 2) to ensure that the ramp does not trigger excessive marking on slow links. Where an implementation knows the link rate, it can set up this minimum at the time it is configured. For instance, it would divide 1 MTU by the link rate to convert it into a serialization time, then if the lower threshold of the Native L AQM ramp was lower than this serialization time, it could increase the thresholds to shift the bottom of the ramp to 2 MTU. This is the approach used in DOCSIS [DOCSIS3.1], because the configured link rate is dedicated to the DualQ.

The pseudocode given here applies where the link rate is unknown, which is more common for software implementations that might be deployed in scenarios where the link is shared with other queues. In lines 5a to 5d in Figure 7 the native L4S marking probability,  $p'_L$ , is zeroed if the queue is only 1 packet (in the default configuration).

Linux implementation note:

- \* In Linux, the check that the queue exceeds Th\_len before marking with the native L4S AQM is actually at enqueue, not dequeue, otherwise it would exempt the last packet of a burst from being marked. The result of the check is conveyed from enqueue to the dequeue function via a boolean in the packet metadata.

Persistent overload is deemed to have occurred when Classic drop/marking probability reaches  $p_{Cmax}$ . Above this point, the Classic drop probability is applied to both L and C queues, irrespective of whether any packet is ECN-capable. ECT packets that are not dropped can still be ECN-marked.

In line 10 of the initialization function (Figure 2), the maximum Classic drop probability  $p_{Cmax} = \min(1/k^2, 1)$  or  $1/4$  for the default coupling factor  $k=2$ . In practice, 25% has been found to be a good threshold to preserve fairness between ECN capable and non ECN capable traffic. This protects the queues against both temporary overload from responsive flows and more persistent overload from any unresponsive traffic that falsely claims to be responsive to ECN.

When the Classic ECN marking probability reaches the  $p_{Cmax}$  threshold ( $1/k^2$ ), the marking probability coupled to the L4S queue,  $p_{CL}$  will always be 100% for any  $k$  (by equation (1) in Section 2). So, for readability, the constant  $p_{Lmax}$  is defined as 1 in line 22 of the initialization function (Figure 2). This is intended to ensure that the L4S queue starts to introduce dropping once ECN-marking saturates at 100% and can rise no further. The 'Prague L4S' requirements [I-D.ietf-tsvwg-ecn-l4s-id] state that, when an L4S congestion control detects a drop, it falls back to a response that coexists with 'Classic' Reno congestion control. So it is correct that, when the L4S queue drops packets, it drops them proportional to  $p'^2$ , as if they are Classic packets.

The two queues each test for overload in lines 4b and 12b of the dequeue function (Figure 7). Lines 8c to 8g drop L4S packets with probability  $p'^2$ . Lines 8h to 8i mark the remaining packets with probability  $p_{CL}$ . Given  $p_{Lmax} = 1$ , all remaining packets will be marked because, to have reached the else block at line 8b,  $p_{CL} \geq 1$ .

Line 2a in the core PI algorithm (Figure 8) deals with overload of the L4S queue when there is little or no Classic traffic. This is necessary, because the core PI algorithm maintains the appropriate drop probability to regulate overload, but it depends on the length of the Classic queue. If there is little or no Classic queue the naive PI update function in Figure 6 would drop nothing, even if the L4S queue were overloaded - so tail drop would have to take over (lines 2 and 3 of Figure 3).

Instead, line 2a of the full PI update function in Figure 8 ensures that the base PI AQM in line 3 is driven by whichever of the two queue delays is greater, but line 3 still always uses the same Classic target (default 15 ms). If L queue delay is greater just because there is little or no Classic traffic, normally it will still be well below the base AQM target. This is because L4S traffic is

also governed by the shallow threshold of its own native AQM (lines 5 and 6 of the dequeue algorithm in Figure 7). So the base AQM will be driven to zero and not contribute. However, if the L queue is overloaded by traffic that is unresponsive to its marking, the `max()` in line 2 enables the L queue to smoothly take over driving the base AQM into overload mode even if there is little or no Classic traffic. Then the base AQM will keep the L queue to the Classic target (default 15 ms) by shedding L packets.

```

1:  dualpi2_dequeue(lq, cq, pkt) {      % Couples L4S & Classic queues
2:    while ( lq.by() + cq.by() > 0 ) {
3:      if ( scheduler() == lq ) {
4a:        lq.dequeue(pkt)                % L4S scheduled
4b:        if ( p_CL < p_Lmax ) {          % Check for overload saturation
5a:          if (lq.len()>Th_len)          % >1 packet queued
5b:            p'_L = laqm(lq.time())      % Native LAQM
5c:          else
5d:            p'_L = 0                    % Suppress marking 1 pkt queue
6:            p_L = max(p'_L, p_CL)        % Combining function
7:            if ( recur(lq, p_L)          % Linear marking
8a:              mark(pkt)
8b:            } else {                    % overload saturation
8c:              if ( recur(lq, p_C) ) {    % probability p_C = p'^2
8e:                drop(pkt)               % revert to Classic drop due to overload
8f:                continue                 % continue to the top of the while loop
8g:              }
8h:              if ( recur(lq, p_CL) )    % probability p_CL = k * p'
8i:                mark(pkt)               % linear marking of remaining packets
8j:            }
9:          } else {
10:           cq.dequeue(pkt)               % Classic scheduled
11:           if ( recur(cq, p_C) ) {        % probability p_C = p'^2
12a:            if ( (ecn(pkt) == 0)         % ECN field = not-ECT
12b:              OR (p_C >= p_Cmax) ) {     % Overload disables ECN
13:              drop(pkt)                  % squared drop, redo loop
14:              continue                   % continue to the top of the while loop
15:            }
16:            mark(pkt)                    % squared mark
17:          }
18:        }
19:      return(pkt)                        % return the packet and stop
20:    }
21:    return(NULL)                         % no packet to dequeue
22:  }

```

Figure 7: Example Dequeue Pseudocode for DualQ Coupled PI2 AQM  
(Including Code for Edge-Cases)

```

1: dualpi2_update(lq, cq) {                                % Update p' every Tupdate
2a:   curq = max(cq.time(), lq.time())                    % use greatest queuing time
3:   p' = p' + alpha * (curq - target) + beta * (curq - prevq)
4:   p_CL = p' * k    % Coupled L4S prob = base prob * coupling factor
5:   p_C = p'^2        % Classic prob = (base prob)^2
6:   prevq = curq
7: }
```

Figure 8: Example PI-Update Pseudocode for DualQ Coupled PI2 AQM  
(Including Overload Code)

The choice of scheduler technology is critical to overload protection (see Section 4.2.2).

- \* A well-understood weighted scheduler such as weighted round robin (WRR) is recommended. As long as the scheduler weight for Classic is small (e.g. 1/16), its exact value is unimportant because it does not normally determine capacity shares. The weight is only important to prevent unresponsive L4S traffic starving Classic traffic in the short term (see Section 4.2.2). This is because capacity sharing between the queues is normally determined by the coupled congestion signal, which overrides the scheduler, by making L4S sources leave roughly equal per-flow capacity available for Classic flows.
- \* Alternatively, a time-shifted FIFO (TS-FIFO) could be used. It works by selecting the head packet that has waited the longest, biased against the Classic traffic by a time-shift of *tshift*. To implement time-shifted FIFO, the scheduler() function in line 3 of the dequeue code would simply be implemented as the scheduler() function at the bottom of Figure 10 in Appendix B. For the public Internet a good value for *tshift* is 50ms. For private networks with smaller diameter, about 4\*target would be reasonable. TS-FIFO is a very simple scheduler, but complexity might need to be added to address some deficiencies (which is why it is not recommended over WRR):
  - TS-FIFO does not fully isolate latency in the L4S queue from uncontrolled bursts in the Classic queue;
  - TS-FIFO is only appropriate if time-stamping of packets is feasible;
  - Even if time-stamping is supported, the sojourn time of the head packet is always stale. For instance, if a burst arrives at an empty queue, the sojourn time only fully measures the burst's delay when its last packet is dequeued, even though the

queue knew about the burst from the start - so it could have signalled congestion earlier. To remedy this, each head packet can be marked when it is dequeued based on the expected delay of the tail packet behind it, as explained below, rather than based on the head packet's own delay due to the packets in front of it. [Heist21] identifies a specific scenario where bursty traffic significantly hits utilization of the L queue. If this effect proves to be more widely applicable, it is believed that using the delay behind the head would improve performance.

The delay behind the head can be implemented by dividing the backlog at dequeue by the link rate or equivalently multiplying the backlog by the delay per unit of backlog. The implementation details will depend on whether the link rate is known; if it is not, a moving average of the delay per unit backlog can be maintained. This delay consists of serialization as well as media acquisition for shared media. So the details will depend strongly on the specific link technology. This approach should be less sensitive to timing errors and cost less in operations and memory than the otherwise equivalent 'scaled sojourn time' metric, which is the sojourn time of a packet scaled by the ratio of the queue sizes when the packet departed and arrived [SigQ-Dyn].

- \* A strict priority scheduler would be inappropriate as discussed in Section 4.2.2.

## Appendix B. Example DualQ Coupled Curvy RED Algorithm

As another example of a DualQ Coupled AQM algorithm, the pseudocode below gives the Curvy RED based algorithm. Although the AQM was designed to be efficient in integer arithmetic, to aid understanding it is first given using floating point arithmetic (Figure 10). Then, one possible optimization for integer arithmetic is given, also in pseudocode (Figure 11). To aid comparison, the line numbers are kept in step between the two by using letter suffixes where the longer code needs extra lines.

### B.1. Curvy RED in Pseudocode

The pseudocode manipulates three main structures of variables: the packet (pkt), the L4S queue (lq) and the Classic queue (cq) and consists of the following five functions:

- \* The initialization function `cred_params_init(...)` (Figure 2) that sets parameter defaults (the API for setting non-default values is omitted for brevity);



- \* The dequeue function `cred_dequeue(lq, cq, pkt)` (Figure 4);
- \* The scheduling function `scheduler()`, which selects between the head packets of the two queues.

It also uses the following functions that are either shown elsewhere, or not shown in full here:

- \* The enqueue function, which is identical to that used for DualPI2, `dualpi2_enqueue(lq, cq, pkt)` in Figure 3;
- \* `mark(pkt)` and `drop(pkt)` for ECN-marking and dropping a packet;
- \* `cq.bytest()` or `lq.bytest()` returns the current length (aka. backlog) of the relevant queue in bytes;
- \* `cq.time()` or `lq.time()` returns the current queuing delay (aka. sojourn time or service time) of the relevant queue in units of time (see Note a in Appendix A.1).

Because Curvy RED was evaluated before DualPI2, certain improvements introduced for DualPI2 were not evaluated for Curvy RED. In the pseudocode below, the straightforward improvements have been added on the assumption they will provide similar benefits, but that has not been proven experimentally. They are: i) a conditional priority scheduler instead of strict priority ii) a time-based threshold for the native L4S AQM; iii) ECN support for the Classic AQM. A recent evaluation has proved that a minimum ECN-marking threshold (`minTh`) greatly improves performance, so this is also included in the pseudocode.

Overload protection has not been added to the Curvy RED pseudocode below so as not to detract from the main features. It would be added in exactly the same way as in Appendix A.2 for the DualPI2 pseudocode. The native L4S AQM uses a step threshold, but a ramp like that described for DualPI2 could be used instead. The scheduler uses the simple TS-FIFO algorithm, but it could be replaced with WRR.

The Curvy RED algorithm has not been maintained or evaluated to the same degree as the DualPI2 algorithm. In initial experiments on broadband access links ranging from 4 Mb/s to 200 Mb/s with base RTTs from 5 ms to 100 ms, Curvy RED achieved good results with the default parameters in Figure 9.

The parameters are categorised by whether they relate to the Classic AQM, the L4S AQM or the framework coupling them together. Constants and variables derived from these parameters are also included at the end of each category. These are the raw input parameters for the

algorithm. A configuration front-end could accept more meaningful parameters (e.g. `RTT_max` and `RTT_typ`) and convert them into these raw parameters, as has been done for DualPI2 in Appendix A. Where necessary, parameters are explained further in the walk-through of the pseudocode below.

```

1: cred_params_init(...) {           % Set input parameter defaults
2:   % DualQ Coupled framework parameters
3:   limit = MAX_LINK_RATE * 250 ms   % Dual buffer size
4:   k' = 1                           % Coupling factor as a power of 2
5:   tshift = 50 ms                   % Time shift of TS-FIFO scheduler
6:   % Constants derived from Classic AQM parameters
7:   k = 2^k'                         % Coupling factor from Equation (1)
6:
7:   % Classic AQM parameters
8:   g_C = 5                         % EWMA smoothing parameter as a power of 1/2
9:   S_C = -1                        % Classic ramp scaling factor as a power of 2
10:  minTh = 500 ms                  % No Classic drop/mark below this queue delay
11:  % Constants derived from Classic AQM parameters
12:  gamma = 2^(-g_C)                % EWMA smoothing parameter
13:  range_C = 2^S_C                  % Range of Classic ramp
14:
15:  % L4S AQM parameters
16:  T = 1 ms                        % Queue delay threshold for native L4S AQM
17:  % Constants derived from above parameters
18:  S_L = S_C - k'                  % L4S ramp scaling factor as a power of 2
19:  range_L = 2^S_L                 % Range of L4S ramp
20: }
```

Figure 9: Example Header Pseudocode for DualQ Coupled Curvy RED AQM

```

1: cred_dequeue(lq, cq, pkt) {           % Couples L4S & Classic queues
2:   while ( lq.bytt() + cq.bytt() > 0 ) {
3:     if ( scheduler() == lq ) {
4:       lq.dequeue(pkt)                  % L4S scheduled
5a:      p_CL = (Q_C - minTh) / range_L
5b:      if ( ( lq.time() > T )
5c:          OR ( p_CL > maxrand(U) ) )
6:        mark(pkt)
7:      } else {
8:        cq.dequeue(pkt)                  % Classic scheduled
9a:        Q_C = gamma * cq.time() + (1-gamma) * Q_C % Classic Q EWMA
10a:       sqrt_p_C = (Q_C - minTh) / range_C
10b:       if ( sqrt_p_C > maxrand(2*U) ) {
11:         if ( (ecn(pkt) == 0) ) {        % ECN field = not-ECT
12:           drop(pkt)                    % Squared drop, redo loop
13:           continue                    % continue to the top of the while loop
14:         }
15:         mark(pkt)
16:       }
17:     }
18:     return(pkt)                        % return the packet and stop here
19:   }
20:   return(NULL)                         % no packet to dequeue
21: }

22: maxrand(u) {                          % return the max of u random numbers
23:   maxr=0
24:   while (u-- > 0)
25:     maxr = max(maxr, rand())            % 0 <= rand() < 1
26:   return(maxr)
27: }

28: scheduler() {
29:   if ( lq.time() + tshift >= cq.time() )
30:     return lq;
31:   else
32:     return cq;
33: }

```

Figure 10: Example Dequeue Pseudocode for DualQ Coupled Curvy RED AQM

The dequeue pseudocode (Figure 10) is repeatedly called whenever the lower layer is ready to forward a packet. It schedules one packet for dequeuing (or zero if the queue is empty) then returns control to the caller, so that it does not block while that packet is being forwarded. While making this dequeue decision, it also makes the necessary AQM decisions on dropping or marking. The alternative of applying the AQMs at enqueue would shift some processing from the

critical time when each packet is dequeued. However, it would also add a whole queue of delay to the control signals, making the control loop very sloppy.

The code is written assuming the AQMs are applied on dequeue (Note 1). All the dequeue code is contained within a large while loop so that if it decides to drop a packet, it will continue until it selects a packet to schedule. If both queues are empty, the routine returns NULL at line 20. Line 3 of the dequeue pseudocode is where the conditional priority scheduler chooses between the L4S queue (lq) and the Classic queue (cq). The time-shifted FIFO scheduler is shown at lines 28-33, which would be suitable if simplicity is paramount (see Note 2).

Within each queue, the decision whether to forward, drop or mark is taken as follows (to simplify the explanation, it is assumed that  $U=1$ ):

L4S: If the test at line 3 determines there is an L4S packet to dequeue, the tests at lines 5b and 5c determine whether to mark it. The first is a simple test of whether the L4S queue delay (lq.time()) is greater than a step threshold  $T$  (Note 3). The second test is similar to the random ECN marking in RED, but with the following differences: i) marking depends on queuing time, not bytes, in order to scale for any link rate without being reconfigured; ii) marking of the L4S queue depends on a logical OR of two tests; one against its own queuing time and one against the queuing time of the `_other_` (Classic) queue; iii) the tests are against the instantaneous queuing time of the L4S queue, but a smoothed average of the other (Classic) queue; iv) the queue is compared with the maximum of  $U$  random numbers (but if  $U=1$ , this is the same as the single random number used in RED).

Specifically, in line 5a the coupled marking probability  $p_{CL}$  is set to the amount by which the averaged Classic queueing delay  $Q_C$  exceeds the minimum queuing delay threshold (minTh) all divided by the L4S scaling parameter range\_L. range\_L represents the queuing delay (in seconds) added to minTh at which marking probability would hit 100%. Then in line 5c (if  $U=1$ ) the result is compared with a uniformly distributed random number between 0 and 1, which ensures that, over range\_L, marking probability will linearly increase with queueing time.

Classic: If the scheduler at line 3 chooses to dequeue a Classic packet and jumps to line 7, the test at line 10b determines whether to drop or mark it. But before that, line 9a updates  $Q_C$ , which is an exponentially weighted moving average (Note 4) of the queuing time of the Classic queue, where cq.time() is the current

instantaneous queueing time of the packet at the head of the Classic queue (zero if empty) and gamma is the EWMA constant (default 1/32, see line 12 of the initialization function).

Lines 10a and 10b implement the Classic AQM. In line 10a the averaged queueing time  $Q_C$  is divided by the Classic scaling parameter  $range\_C$ , in the same way that queueing time was scaled for L4S marking. This scaled queueing time will be squared to compute Classic drop probability so, before it is squared, it is effectively the square root of the drop probability, hence it is given the variable name  $sqrt\_p\_C$ . The squaring is done by comparing it with the maximum out of two random numbers (assuming  $U=1$ ). Comparing it with the maximum out of two is the same as the logical 'AND' of two tests, which ensures drop probability rises with the square of queueing time.

The AQM functions in each queue (lines 5c & 10b) are two cases of a new generalization of RED called Curvy RED, motivated as follows. When the performance of this AQM was compared with FQ-CoDel and PIE, their goal of holding queueing delay to a fixed target seemed misguided [CRED\_Insights]. As the number of flows increases, if the AQM does not allow host congestion controllers to increase queueing delay, it has to introduce abnormally high levels of loss. Then loss rather than queueing becomes the dominant cause of delay for short flows, due to timeouts and tail losses.

Curvy RED constrains delay with a softened target that allows some increase in delay as load increases. This is achieved by increasing drop probability on a convex curve relative to queue growth (the square curve in the Classic queue, if  $U=1$ ). Like RED, the curve hugs the zero axis while the queue is shallow. Then, as load increases, it introduces a growing barrier to higher delay. But, unlike RED, it requires only two parameters, not three. The disadvantage of Curvy RED (compared to a PI controller for example) is that it is not adapted to a wide range of RTTs. Curvy RED can be used as is when the RTT range to be supported is limited, otherwise an adaptation mechanism is needed.

From our limited experiments with Curvy RED so far, recommended values of these parameters are:  $S_C = -1$ ;  $g_C = 5$ ;  $T = 5 * MTU$  at the link rate (about 1ms at 60Mb/s) for the range of base RTTs typical on the public Internet. [CRED\_Insights] explains why these parameters are applicable whatever rate link this AQM implementation is deployed on and how the parameters would need to be adjusted for a scenario with a different range of RTTs (e.g. a data centre). The setting of  $k$  depends on policy (see Section 2.5 and Appendix C.2 respectively for its recommended setting and guidance on alternatives).

There is also a cUrviness parameter,  $U$ , which is a small positive integer. It is likely to take the same hard-coded value for all implementations, once experiments have determined a good value. Only  $U=1$  has been used in experiments so far, but results might be even better with  $U=2$  or higher.

Notes:

1. The alternative of applying the AQMs at enqueue would shift some processing from the critical time when each packet is dequeued. However, it would also add a whole queue of delay to the control signals, making the control loop sloppier (for a typical RTT it would double the Classic queue's feedback delay). On a platform where packet timestamping is feasible, e.g. Linux, it is also easiest to apply the AQMs at dequeue because that is where queuing time is also measured.
2. WRR better isolates the L4S queue from large delay bursts in the Classic queue, but it is slightly less simple than TS-FIFO. If WRR were used, a low default Classic weight (e.g. 1/16) would need to be configured in place of the time shift in line 5 of the initialization function (Figure 9).
3. A step function is shown for simplicity. A ramp function (see Figure 5 and the discussion around it in Appendix A.1) is recommended, because it is more general than a step and has the potential to enable L4S congestion controls to converge more rapidly.
4. An EWMA is only one possible way to filter bursts; other more adaptive smoothing methods could be valid and it might be appropriate to decrease the EWMA faster than it increases, e.g. by using the minimum of the smoothed and instantaneous queue delays, `min(Q_C, qc.time())`.

## B.2. Efficient Implementation of Curvy RED

Although code optimization depends on the platform, the following notes explain where the design of Curvy RED was particularly motivated by efficient implementation.

The Classic AQM at line 10b calls `maxrand(2*U)`, which gives twice as much curviness as the call to `maxrand(U)` in the marking function at line 5c. This is the trick that implements the square rule in equation (1) (Section 2.1). This is based on the fact that, given a number  $X$  from 1 to 6, the probability that two dice throws will both be less than  $X$  is the square of the probability that one throw will be less than  $X$ . So, when  $U=1$ , the L4S marking function is linear and the Classic dropping function is squared. If  $U=2$ , L4S would be a square function and Classic would be quartic. And so on.

The `maxrand(u)` function in lines 16-21 simply generates  $u$  random numbers and returns the maximum. Typically, `maxrand(u)` could be run in parallel out of band. For instance, if  $U=1$ , the Classic queue would require the maximum of two random numbers. So, instead of calling `maxrand(2*U)` in-band, the maximum of every pair of values from a pseudorandom number generator could be generated out-of-band, and held in a buffer ready for the Classic queue to consume.

```

1: cred_dequeue(lq, cq, pkt) {           % Couples L4S & Classic queues
2:   while ( lq.bytt() + cq.bytt() > 0 ) {
3:     if ( scheduler() == lq ) {
4:       lq.dequeue(pkt)                  % L4S scheduled
5:       if ((lq.time() > T) OR (Q_C >> (S_L-2) > maxrand(U)))
6:         mark(pkt)
7:     } else {
8:       cq.dequeue(pkt)                  % Classic scheduled
9:       Q_C += (qc.ns() - Q_C) >> g_C    % Classic Q EWMA
10:      if ( (Q_C >> (S_C-2) ) > maxrand(2*U) ) {
11:        if ( (ecn(pkt) == 0) ) {        % ECN field = not-ECT
12:          drop(pkt)                    % Squared drop, redo loop
13:          continue                    % continue to the top of the while loop
14:        }
15:        mark(pkt)
16:      }
17:    }
18:    return(pkt)                        % return the packet and stop here
19:  }
20:  return(NULL)                        % no packet to dequeue
21: }
```

Figure 11: Optimised Example Dequeue Pseudocode for Coupled DualQ AQM using Integer Arithmetic

The two ranges, `range_L` and `range_C` are expressed as powers of 2 so that division can be implemented as a right bit-shift (`>>`) in lines 5 and 10 of the integer variant of the pseudocode (Figure 11).

For the integer variant of the pseudocode, an integer version of the `rand()` function used at line 25 of the `maxrand(function)` in Figure 10 would be arranged to return an integer in the range  $0 \leq \text{maxrand}() < 2^{32}$  (not shown). This would scale up all the floating point probabilities in the range  $[0,1]$  by  $2^{32}$ .

Queuing delays are also scaled up by  $2^{32}$ , but in two stages: i) In line 9 queuing time `qc.ns()` is returned in integer nanoseconds, making the value about  $2^{30}$  times larger than when the units were seconds, ii) then in lines 5 and 10 an adjustment of  $-2$  to the right bit-shift multiplies the result by  $2^2$ , to complete the scaling by  $2^{32}$ .

In line 8 of the initialization function, the EWMA constant `gamma` is represented as an integer power of 2, `g_C`, so that in line 9 of the integer code the division needed to weight the moving average can be implemented by a right bit-shift (`>> g_C`).

## Appendix C. Choice of Coupling Factor, $k$

### C.1. RTT-Dependence

Where Classic flows compete for the same capacity, their relative flow rates depend not only on the congestion probability, but also on their end-to-end RTT (= base RTT + queue delay). The rates of Reno [RFC5681] flows competing over an AQM are roughly inversely proportional to their RTTs. Cubic exhibits similar RTT-dependence when in Reno-compatibility mode, but it is less RTT-dependent otherwise.

Until the early experiments with the DualQ Coupled AQM, the importance of the reasonably large Classic queue in mitigating RTT-dependence when the base RTT is low had not been appreciated. Appendix A.1.6 of the L4S ECN protocol [I-D.ietf-tsvwg-ecn-l4s-id] uses numerical examples to explain why bloated buffers had concealed the RTT-dependence of Classic congestion controls before that time. Then it explains why, the more that queuing delays have reduced, the more that RTT-dependence has surfaced as a potential starvation problem for long RTT flows, when competing against very short RTT flows.

Given that congestion control on end-systems is voluntary, there is no reason why it has to be voluntarily RTT-dependent. The RTT-dependence of existing Classic traffic cannot be 'undeployed'. Therefore, [I-D.ietf-tsvwg-ecn-l4s-id] requires L4S congestion controls to be significantly less RTT-dependent than the standard Reno congestion control [RFC5681], at least at low RTT. Then RTT-



dependence ought to be no worse than it is with appropriately sized Classic buffers. Following this approach means there is no need for network devices to address RTT-dependence, although there would be no harm if they did, which per-flow queuing inherently does.

## C.2. Guidance on Controlling Throughput Equivalence

The coupling factor,  $k$ , determines the balance between L4S and Classic flow rates (see Section 2.5.2.1 and equation (1)).

For the public Internet, a coupling factor of  $k=2$  is recommended, and justified below. For scenarios other than the public Internet, a good coupling factor can be derived by plugging the appropriate numbers into the same working.

To summarize the maths below, from equation (7) it can be seen that choosing  $k=1.64$  would theoretically make L4S throughput roughly the same as Classic, if their actual end-to-end RTTs were the same. However, even if the base RTTs are the same, the actual RTTs are unlikely to be the same, because Classic traffic needs a fairly large queue to avoid under-utilization and excess drop. Whereas L4S does not.

Therefore, to determine the appropriate coupling factor policy, the operator needs to decide at what base RTT it wants L4S and Classic flows to have roughly equal throughput, once the effect of the additional Classic queue on Classic throughput has been taken into account. With this approach, a network operator can determine a good coupling factor without knowing the precise L4S algorithm for reducing RTT-dependence - or even in the absence of any algorithm.

The following additional terminology will be used, with appropriate subscripts:

$r$ : Packet rate [pkt/s]

$R$ : RTT [s/round]

$p$ : ECN marking probability []

On the Classic side, we consider Reno as the most sensitive and therefore worst-case Classic congestion control. We will also consider Cubic in its Reno-friendly mode ('CReno'), as the most prevalent congestion control, according to the references and analysis in [PI2param]. In either case, the Classic packet rate in steady state is given by the well-known square root formula for Reno congestion control:

$$r_C = 1.22 / (R_C * p_C^{0.5}) \quad (5)$$

On the L4S side, we consider the Prague congestion control [I-D.briscoe-iccrp-prague-congestion-control] as the reference for steady-state dependence on congestion. Prague conforms to the same equation as DCTCP, but we do not use the equation derived in the DCTCP paper, which is only appropriate for step marking. The coupled marking,  $p_{CL}$ , is the appropriate one when considering throughput equivalence with Classic flows. Unlike step marking, coupled markings are inherently spaced out, so we use the formula for DCTCP packet rate with probabilistic marking derived in Appendix A of [PI2]. We use the equation without RTT-independence enabled, which will be explained later.

$$r_L = 2 / (R_L * p_{CL}) \quad (6)$$

For packet rate equivalence, we equate the two packet rates and rearrange into the same form as Equation (1), so the two can be equated and simplified to produce a formula for a theoretical coupling factor, which we shall call  $k^*$ :

$$\begin{aligned} r_C &= r_L \\ \Rightarrow p_C &= (p_{CL}/1.64 * R_L/R_C)^2 \\ p_C &= (p_{CL} / k)^2 \quad (1) \\ k^* &= 1.64 * (R_C / R_L) \quad (7) \end{aligned}$$

We say that this coupling factor is theoretical, because it is in terms of two RTTs, which raises two practical questions: i) for multiple flows with different RTTs, the RTT for each traffic class would have to be derived from the RTTs of all the flows in that class (actually the harmonic mean would be needed); ii) a network node cannot easily know the RTT of any of the flows anyway.

RTT-dependence is caused by window-based congestion control, so it ought to be reversed there, not in the network. Therefore, we use a fixed coupling factor in the network, and reduce RTT-dependence in L4S senders. We cannot expect Classic senders to all be updated to reduce their RTT-dependence. But solely addressing the problem in L4S senders at least makes RTT-dependence no worse – not just between L4S senders, but also between L4S and Classic senders.

Traditionally, throughput equivalence has been defined for flows under comparable conditions, including with the same base RTT [RFC2914]. So if we assume the same base RTT,  $R_b$ , for comparable flows, we can put both  $R_C$  and  $R_L$  in terms of  $R_b$ .

We can approximate the L4S RTT to be hardly greater than the base RTT, i.e.  $R_L \approx R_b$ . And we can replace  $R_C$  with  $(R_b + q_C)$ , where the Classic queue,  $q_C$ , depends on the target queue delay that the operator has configured for the Classic AQM.

Taking PI2 as an example Classic AQM, it seems that we could just take  $R_C = R_b + \text{target}$  (recommended 15 ms by default in Appendix A.1). However, target is roughly the queue depth reached by the tips of the sawteeth of a congestion control, not the average [PI2param]. That is  $R_{\text{max}} = R_b + \text{target}$ .

The position of the average in relation to the max depends on the amplitude and geometry of the sawteeth. We consider two examples: Reno [RFC5681], as the most sensitive worst-case, and Cubic [RFC8312] in its Reno-friendly mode ('Creno') as the most prevalent congestion control algorithm on the Internet according to the references in [PI2param]. Both are AIMD, so we will generalize using  $b$  as the multiplicative decrease factor ( $b_r = 0.5$  for Reno,  $b_c = 0.7$  for Creno). Then:

$$\begin{aligned} R_C &= (R_{\text{max}} + b \cdot R_{\text{max}}) / 2 \\ &= R_{\text{max}} * (1+b)/2 \end{aligned}$$

$$R_{\text{reno}} = 0.75 * (R_b + \text{target}); \quad R_{\text{creno}} = 0.85 * (R_b + \text{target}). \quad (8)$$

Plugging all this into equation (7) we get a fixed coupling factor for each:

$$\begin{aligned} k_{\text{reno}} &= 1.64 * 0.75 * (R_b + \text{target}) / R_b \\ &= 1.23 * (1 + \text{target} / R_b); \quad k_{\text{creno}} = 1.39 * (1 + \text{target} / R_b) \end{aligned}$$

An operator can then choose the base RTT at which it wants throughput to be equivalent. For instance, if we recommend that the operator chooses  $R_b = 25$  ms, as a typical base RTT between Internet users and CDNs [PI2param], then these coupling factors become:

$$\begin{aligned} k_{\text{reno}} &= 1.23 * (1 + 15/25) & k_{\text{creno}} &= 1.39 * (1 + 15/25) \\ &= 1.97 & &= 2.22 \\ &\approx 2 & &\approx 2 \end{aligned} \quad (9)$$

The approximation is relevant to any of the above example DualQ Coupled algorithms, which use a coupling factor that is an integer power of 2 to aid efficient implementation. It also fits best to the worst case (Reno).

To check the outcome of this coupling factor, we can express the ratio of L4S to Classic throughput by substituting from their rate equations (5) and (6), then also substituting for  $p_C$  in terms of  $p_{CL}$ , using equation (1) with  $k=2$  as just determined for the Internet:

$$\begin{aligned} r_L / r_C &= 2 (R_C * p_C^{0.5}) / 1.22 (R_L * p_{CL}) \\ &= (R_C * p_{CL}) / (1.22 * R_L * p_{CL}) \\ &= R_C / (1.22 * R_L) \end{aligned} \tag{10}$$

As an example, we can then consider single competing CReNO and Prague flows, by expressing both their RTTs in (10) in terms of their base RTTs,  $R_{bC}$  and  $R_{bL}$ . So  $R_C$  is replaced by equation (8) for CReNO. And  $R_L$  is replaced by the  $\max()$  function below, which represents the effective RTT of the current Prague congestion control [I-D.briscoe-iccrp-prague-congestion-control] in its (default) RTT-independent mode, because it sets a floor to the effective RTT that it uses for additive increase:

$$\begin{aligned} \tilde{R} &= 0.85 * (R_{bC} + \text{target}) / (1.22 * \max(R_{bL}, R_{\text{typ}})) \\ \tilde{R} &= (R_{bC} + \text{target}) / (1.4 * \max(R_{bL}, R_{\text{typ}})) \end{aligned}$$

It can be seen that, for base RTTs below target (15 ms), both the numerator and the denominator plateau, which has the desired effect of limiting RTT-dependence.

At the start of the above derivations, an explanation was promised for why the L4S throughput equation in equation (6) did not need to model RTT-independence. This is because we only use one point - at the the typical base RTT where the operator chooses to calculate the coupling factor. Then, throughput equivalence will at least hold at that chosen point. Nonetheless, assuming Prague senders implement RTT-independence over a range of RTTs below this, the throughput equivalence will then extend over that range as well.

Congestion control designers can choose different ways to reduce RTT-dependence. And each operator can make a policy choice to decide on a different base RTT, and therefore a different  $k$ , at which it wants throughput equivalence. Nonetheless, for the Internet, it makes sense to choose what is believed to be the typical RTT most users experience, because a Classic AQM's target queuing delay is also derived from a typical RTT for the Internet.

As a non-Internet example, for localized traffic from a particular ISP's data centre, using the measured RTTs, it was calculated that a value of  $k = 8$  would achieve throughput equivalence, and experiments verified the formula very closely.

But, for a typical mix of RTTs across the general Internet, a value of  $k=2$  is recommended as a good workable compromise.

#### Authors' Addresses

Koen De Schepper  
Nokia Bell Labs  
Antwerp  
Belgium  
Email: [koen.de\\_schepper@nokia.com](mailto:koen.de_schepper@nokia.com)  
URI: [https://www.bell-labs.com/usr/koen.de\\_schepper](https://www.bell-labs.com/usr/koen.de_schepper)

Bob Briscoe (editor)  
Independent  
United Kingdom  
Email: [ietf@bobbriscoe.net](mailto:ietf@bobbriscoe.net)  
URI: <http://bobbriscoe.net/>

Greg White  
CableLabs  
Louisville, CO,  
United States of America  
Email: [G.White@CableLabs.com](mailto:G.White@CableLabs.com)

Internet Engineering Task Force	G. Fairhurst
Internet-Draft	T. Jones
Updates: 4821, 4960, 6951, 8085, 8261 (if approved)	University of Aberdeen
Intended status: Standards Track	M. Tuexen
Expires: 12 December 2020	I. Ruengeler
	T. Voelker
	Muenster University of Applied Sciences
	10 June 2020

Packetization Layer Path MTU Discovery for Datagram Transports  
draft-ietf-tsvwg-datagram-plpmtud-22

Abstract

This document describes a robust method for Path MTU Discovery (PMTUD) for datagram Packetization Layers (PLs). It describes an extension to RFC 1191 and RFC 8201, which specifies ICMP-based Path MTU Discovery for IPv4 and IPv6. The method allows a PL, or a datagram application that uses a PL, to discover whether a network path can support the current size of datagram. This can be used to detect and reduce the message size when a sender encounters a packet black hole (where packets are discarded). The method can probe a network path with progressively larger packets to discover whether the maximum packet size can be increased. This allows a sender to determine an appropriate packet size, providing functionality for datagram transports that is equivalent to the Packetization Layer PMTUD specification for TCP, specified in RFC 4821.

This document updates RFC 4821 to specify the PLPMTUD method for datagram PLs. It also updates RFC 8085 to refer to the method specified in this document instead of the method in RFC 4821 for use with UDP datagrams. Section 7.3 of RFC 4960 recommends an endpoint apply the techniques in RFC 4821 on a per-destination-address basis. RFC 4960, RFC 6951, and RFC 8261 are updated to recommend that SCTP, SCTP encapsulated in UDP and SCTP encapsulated in DTLS use the method specified in this document instead of the method in RFC 4821.

The document also provides implementation notes for incorporating Datagram PMTUD into IETF datagram transports or applications that use datagram transports.

When published, this specification updates RFC 4960, RFC 4821, RFC 8085 and RFC 8261.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 12 December 2020.

## Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	4
1.1. Classical Path MTU Discovery . . . . .	4
1.2. Packetization Layer Path MTU Discovery . . . . .	6
1.3. Path MTU Discovery for Datagram Services . . . . .	7
2. Terminology . . . . .	8
3. Features Required to Provide Datagram PLPMTUD . . . . .	11
4. DPLPMTUD Mechanisms . . . . .	14
4.1. PLPMTU Probe Packets . . . . .	14
4.2. Confirmation of Probed Packet Size . . . . .	15
4.3. Black Hole Detection and Reducing the PLPMTU . . . . .	15
4.4. The Maximum Packet Size (MPS) . . . . .	17
4.5. Disabling the Effect of PMTUD . . . . .	18
4.6. Response to PTB Messages . . . . .	18
4.6.1. Validation of PTB Messages . . . . .	18
4.6.2. Use of PTB Messages . . . . .	19

5.	Datagram Packetization Layer PMTUD . . . . .	20
5.1.	DPLPMTUD Components . . . . .	21
5.1.1.	Timers . . . . .	21
5.1.2.	Constants . . . . .	22
5.1.3.	Variables . . . . .	23
5.1.4.	Overview of DPLPMTUD Phases . . . . .	24
5.2.	State Machine . . . . .	26
5.3.	Search to Increase the PLPMTU . . . . .	29
5.3.1.	Probing for a larger PLPMTU . . . . .	29
5.3.2.	Selection of Probe Sizes . . . . .	30
5.3.3.	Resilience to Inconsistent Path Information . . . . .	30
5.4.	Robustness to Inconsistent Paths . . . . .	31
6.	Specification of Protocol-Specific Methods . . . . .	31
6.1.	Application support for DPLPMTUD with UDP or UDP-Lite . . . . .	31
6.1.1.	Application Request . . . . .	32
6.1.2.	Application Response . . . . .	32
6.1.3.	Sending Application Probe Packets . . . . .	32
6.1.4.	Initial Connectivity . . . . .	32
6.1.5.	Validating the Path . . . . .	32
6.1.6.	Handling of PTB Messages . . . . .	32
6.2.	DPLPMTUD for SCTP . . . . .	33
6.2.1.	SCTP/IPv4 and SCTP/IPv6 . . . . .	33
6.2.1.1.	Initial Connectivity . . . . .	33
6.2.1.2.	Sending SCTP Probe Packets . . . . .	33
6.2.1.3.	Validating the Path with SCTP . . . . .	34
6.2.1.4.	PTB Message Handling by SCTP . . . . .	34
6.2.2.	DPLPMTUD for SCTP/UDP . . . . .	34
6.2.2.1.	Initial Connectivity . . . . .	35
6.2.2.2.	Sending SCTP/UDP Probe Packets . . . . .	35
6.2.2.3.	Validating the Path with SCTP/UDP . . . . .	35
6.2.2.4.	Handling of PTB Messages by SCTP/UDP . . . . .	35
6.2.3.	DPLPMTUD for SCTP/DTLS . . . . .	35
6.2.3.1.	Initial Connectivity . . . . .	35
6.2.3.2.	Sending SCTP/DTLS Probe Packets . . . . .	36
6.2.3.3.	Validating the Path with SCTP/DTLS . . . . .	36
6.2.3.4.	Handling of PTB Messages by SCTP/DTLS . . . . .	36
6.3.	DPLPMTUD for QUIC . . . . .	36
7.	Acknowledgments . . . . .	36
8.	IANA Considerations . . . . .	36
9.	Security Considerations . . . . .	37
10.	References . . . . .	38
10.1.	Normative References . . . . .	38
10.2.	Informative References . . . . .	39
	Appendix A. Revision Notes . . . . .	41
	Authors' Addresses . . . . .	46



## 1. Introduction

The IETF has specified datagram transport using UDP, SCTP, and DCCP, as well as protocols layered on top of these transports (e.g., SCTP/UDP, DCCP/UDP, QUIC/UDP), and direct datagram transport over the IP network layer. This document describes a robust method for Path MTU Discovery (PMTUD) that can be used with these transport protocols (or the applications that use their transport service) to discover an appropriate size of packet to use across an Internet path.

### 1.1. Classical Path MTU Discovery

Classical Path Maximum Transmission Unit Discovery (PMTUD) can be used with any transport that is able to process ICMP Packet Too Big (PTB) messages (e.g., [RFC1191] and [RFC8201]). In this document, the term PTB message is applied to both IPv4 ICMP Unreachable messages (type 3) that carry the error Fragmentation Needed (Type 3, Code 4) [RFC0792] and ICMPv6 Packet Too Big messages (Type 2) [RFC4443]. When a sender receives a PTB message, it reduces the effective MTU to the value reported as the Link MTU in the PTB message. A method from time-to-time increases the packet size in attempt to discover an increase in the supported PMTU. The packets sent with a size larger than the current effective PMTU are known as probe packets.

Packets not intended as probe packets are either fragmented to the current effective PMTU, or the attempt to send fails with an error code. Applications can be provided with a primitive to let them read the Maximum Packet Size (MPS), derived from the current effective PMTU.

Classical PMTUD is subject to protocol failures. One failure arises when traffic using a packet size larger than the actual PMTU is black-holed (all datagrams larger than the actual PMTU, are discarded). This could arise when the PTB messages are not delivered back to the sender for some reason (see for example [RFC2923]).

Examples where PTB messages are not delivered include:

- \* The generation of ICMP messages is usually rate limited. This could result in no PTB messages being generated to the sender (see section 2.4 of [RFC4443])
- \* ICMP messages can be filtered by middleboxes (including firewalls) [RFC4890]. A firewall could be configured with a policy to block incoming ICMP messages, which would prevent reception of PTB messages to a sending endpoint behind this firewall.

- \* When the router issuing the ICMP message drops a tunneled packet, the resulting ICMP message will be directed to the tunnel ingress. This tunnel endpoint is responsible for forwarding the ICMP message and also processing the quoted packet within the payload field to remove the effect of the tunnel, and return a correctly formatted ICMP message to the sender [I-D.ietf-intarea-tunnels]. Failure to do this prevents the PTB message reaching the original sender.
- \* Asymmetry in forwarding can result in there being no return route to the original sender, which would prevent an ICMP message being delivered to the sender. This issue can also arise when policy-based routing is used, Equal Cost Multipath (ECMP) routing is used, or a middlebox acts as an application load balancer. An example is where the path towards the server is chosen by ECMP routing depending on bytes in the IP payload. In this case, when a packet sent by the server encounters a problem after the ECMP router, then any resulting ICMP message also needs to be directed by the ECMP router towards the original sender.
- \* There are additional cases where the next hop destination fails to receive a packet because of its size. This could be due to misconfiguration of the layer 2 path between nodes, for instance the MTU configured in a layer 2 switch, or misconfiguration of the Maximum Receive Unit (MRU). If a packet is dropped by the link, this will not cause a PTB message to be sent to the original sender.

Another failure could result if a node that is not on the network path sends a PTB message that attempts to force a sender to change the effective PMTU [RFC8201]. A sender can protect itself from reacting to such messages by utilizing the quoted packet within a PTB message payload to validate that the received PTB message was generated in response to a packet that had actually originated from the sender. However, there are situations where a sender would be unable to provide this validation. Examples where validation of the PTB message is not possible include:

- \* When a router issuing the ICMP message implements RFC792 [RFC0792], it is only required to include the first 64 bits of the IP payload of the packet within the quoted payload. There could be insufficient bytes remaining for the sender to interpret the quoted transport information.

Note: The recommendation in RFC1812 [RFC1812] is that IPv4 routers return a quoted packet with as much of the original datagram as possible without the length of the ICMP datagram exceeding 576

bytes. IPv6 routers include as much of the invoking packet as possible without the ICMPv6 packet exceeding 1280 bytes [RFC4443].

- \* The use of tunnels/encryption can reduce the size of the quoted packet returned to the original source address, increasing the risk that there could be insufficient bytes remaining for the sender to interpret the quoted transport information.
- \* Even when the PTB message includes sufficient bytes of the quoted packet, the network layer could lack sufficient context to validate the message, because validation depends on information about the active transport flows at an endpoint node (e.g., the socket/address pairs being used, and other protocol header information).
- \* When a packet is encapsulated/tunneled over an encrypted transport, the tunnel/encapsulation ingress might have insufficient context, or computational power, to reconstruct the transport header that would be needed to perform validation.
- \* When an ICMP message is generated by a router in a network segment that has inserted a header into a packet, the quoted packet could contain additional protocol header information that was not included in the original sent packet, and which the PL sender does not process or may not know how to process. This could disrupt the ability of the sender to validate this PTB message.
- \* A Network Address Translation (NAT) device that translates a packet header, ought to also translate ICMP messages and update the ICMP quoted packet [RFC5508] in that message. If this is not correctly translated then the sender would not be able to associate the message with the PL that originated the packet, and hence this ICMP message cannot be validated.

## 1.2. Packetization Layer Path MTU Discovery

The term Packetization Layer (PL) has been introduced to describe the layer that is responsible for placing data blocks into the payload of IP packets and selecting an appropriate MPS. This function is often performed by a transport protocol (e.g., DCCP, RTP, SCTP, QUIC), but can also be performed by other encapsulation methods working above the transport layer.

In contrast to PMTUD, Packetization Layer Path MTU Discovery (PLPMTUD) [RFC4821] introduced a method that does not rely upon reception and validation of PTB messages. It is therefore more robust than Classical PMTUD. This has become the recommended approach for implementing discovery of the PMTU [BCP145].

It uses a general strategy where the PL sends probe packets to search for the largest size of unfragmented datagram that can be sent over a network path. Probe packets are sent to explore using a larger packet size. If a probe packet is successfully delivered (as determined by the PL), then the PLPMTU is raised to the size of the successful probe. If a black hole is detected (e.g., where packets of size PLPMTU are consistently not received), the method reduces the PLPMTU.

Datagram PLPMTUD introduces flexibility in implementation. At one extreme, it can be configured to only perform Black Hole Detection and recovery with increased robustness compared to Classical PMTUD. At the other extreme, all PTB processing can be disabled, and PLPMTUD replaces Classical PMTUD.

PLPMTUD can also include additional consistency checks without increasing the risk that data is lost when probing to discover the Path MTU. For example, information available at the PL, or higher layers, enables received PTB messages to be validated before being utilized.

### 1.3. Path MTU Discovery for Datagram Services

Section 5 of this document presents a set of algorithms for datagram protocols to discover the largest size of unfragmented datagram that can be sent over a network path. The method relies upon features of the PL described in Section 3 and applies to transport protocols operating over IPv4 and IPv6. It does not require cooperation from the lower layers, although it can utilize PTB messages when these received messages are made available to the PL.

The message size guidelines in section 3.2 of the UDP Usage Guidelines [BCP145] state "an application SHOULD either use the Path MTU information provided by the IP layer or implement Path MTU Discovery (PMTUD)", but does not provide a mechanism for discovering the largest size of unfragmented datagram that can be used on a network path. The present document updates RFC 8085 to specify this method in place of PLPMTUD [RFC4821] and provides a mechanism for sharing the discovered largest size as the MPS (see Section 4.4).

Section 10.2 of [RFC4821] recommended a PLPMTUD probing method for the Stream Control Transport Protocol (SCTP). SCTP utilizes probe packets consisting of a minimal sized HEARTBEAT chunk bundled with a PAD chunk as defined in [RFC4820]. However, RFC 4821 did not provide a complete specification. The present document replaces that description by providing a complete specification.

The Datagram Congestion Control Protocol (DCCP) [RFC4340] requires implementations to support Classical PMTUD and states that a DCCP sender "MUST maintain the MPS allowed for each active DCCP session". It also defines the current congestion control MPS (CCMPS) supported by a network path. This recommends use of PMTUD, and suggests use of control packets (DCCP-Sync) as path probe packets, because they do not risk application data loss. The method defined in this specification can be used with DCCP.

Section 4 and Section 5 define the protocol mechanisms and specification for Datagram Packetization Layer Path MTU Discovery (DPLPMTUD).

Section 6 specifies the method for datagram transports and provides information to enable the implementation of PLPMTUD with other datagram transports and applications that use datagram transports.

Section 6 also provides updated recommendations for [RFC6951] and [RFC8261].

## 2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terminology is defined. Relevant terms are directly copied from [RFC4821], and the definitions in [RFC1122].

**Acknowledged PL:** A PL that includes a mechanism that can confirm successful delivery of datagrams to the remote PL endpoint (e.g., SCTP). Typically, the PL receiver returns acknowledgments corresponding to the received datagrams, which can be utilised to detect black-holing of packets (c.f., Unacknowledged PL).

**Actual PMTU:** The Actual PMTU is the PMTU of a network path between a sender PL and a destination PL, which the DPLPMTUD algorithm seeks to determine.

**Black Hole:** A Black Hole is encountered when a sender is unaware that packets are not being delivered to the destination end point. Two types of Black Hole are relevant to DPLPMTUD:

- \* Packets encounter a packet Black Hole when packets are not delivered to the destination endpoint (e.g., when the sender transmits packets of a particular size with a previously known

effective PMTU and they are discarded by the network).

- \* An ICMP Black Hole is encountered when the sender is unaware that packets are not delivered to the destination endpoint because PTB messages are not received by the originating PL sender.

**Classical Path MTU Discovery:** Classical PMTUD is a process described in [RFC1191] and [RFC8201], in which nodes rely on PTB messages to learn the largest size of unfragmented packet that can be used across a network path.

**Datagram:** A datagram is a transport-layer protocol data unit, transmitted in the payload of an IP packet.

**Effective PMTU:** The Effective PMTU is the current estimated value for PMTU that is used by a PMTUD. This is equivalent to the PLPMTU derived by PLPMTUD plus the size of any headers added below the PL, including the IP layer headers.

**EMTU\_S:** The Effective MTU for sending (EMTU\_S) is defined in [RFC1122] as "the maximum IP datagram size that may be sent, for a particular combination of IP source and destination addresses...".

**EMTU\_R:** The Effective MTU for receiving (EMTU\_R) is designated in [RFC1122] as "the largest datagram size that can be reassembled".

**Link:** A Link is a communication facility or medium over which nodes can communicate at the link layer, i.e., a layer below the IP layer. Examples are Ethernet LANs and Internet (or higher) layer tunnels.

**Link MTU:** The Link Maximum Transmission Unit (MTU) is the size in bytes of the largest IP packet, including the IP header and payload, that can be transmitted over a link. Note that this could more properly be called the IP MTU, to be consistent with how other standards organizations use the acronym. This includes the IP header, but excludes link layer headers and other framing that is not part of IP or the IP payload. Other standards organizations generally define the link MTU to include the link layer headers. This specification continues the requirement in [RFC4821], that states "All links MUST enforce their MTU: links that might non-deterministically deliver packets that are larger than their rated MTU MUST consistently discard such packets."

**MAX\_PLPMTU:** The MAX\_PLPMTU is the largest size of PLPMTU that DPLPMTUD will attempt to use (see the constants defined in Section 5.1.2).

**MIN\_PLPMTU:** The MIN\_PLPMTU is the smallest size of PLPMTU that DPLPMTUD will attempt to use (see the constants defined in Section 5.1.2).

**MPS:** The Maximum Packet Size (MPS) is the largest size of application data block that can be sent across a network path by a PL using a single Datagram (see Section 4.4).

**MSL:** Maximum Segment Lifetime (MSL) The maximum delay a packet is expected to experience across a path, taken as 2 minutes [BCP145].

**Packet:** A Packet is the IP header(s) and any extension headers/options plus the IP payload.

**Packetization Layer (PL):** The PL is a layer of the network stack that places data into packets and performs transport protocol functions. Examples of a PL include: TCP, SCTP, SCTP over UDP, SCTP over DTLS, or QUIC.

**Path:** The Path is the set of links and routers traversed by a packet between a source node and a destination node by a particular flow.

**Path MTU (PMTU):** The Path MTU (PMTU) is the minimum of the Link MTU of all the links forming a network path between a source node and a destination node, as used by PMTUD.

**PTB:** In this document, the term PTB message is applied to both IPv4 ICMP Unreachable messages (type 3) that carry the error Fragmentation Needed (Type 3, Code 4) [RFC0792] and ICMPv6 Packet Too Big messages (Type 2) [RFC4443].

**PTB\_SIZE:** The PTB\_SIZE is a value reported in a validated PTB message that indicates next hop link MTU of a router along the path.

**PL\_PTBSIZE:** The size reported in a validated PTB message, reduced by the size of all headers added by layers below the PL.

**PLPMTU:** The Packetization Layer PMTU is an estimate of the largest size of PL datagram that can be sent by a path, controlled by PLPMTUD.

**PLPMTUD:** Packetization Layer Path MTU Discovery (PLPMTUD), the method described in this document for datagram PLs, which is an extension to Classical PMTU Discovery.

**Probe packet:** A probe packet is a datagram sent with a purposely chosen size (typically the current PLPMTU or larger) to detect if

packets of this size can be successfully sent end-to-end across the network path.

Unacknowledged PL: A PL that does not itself provide a mechanism to confirm delivery of datagrams to the remote PL endpoint (e.g., UDP), and therefore requires DPLPMTUD to provide a mechanism to detect black-holing of packets (c.f., Acknowledged PL).

### 3. Features Required to Provide Datagram PLPMTUD

The principles expressed in [RFC4821] apply to the use of the technique with any PL. TCP PLPMTUD has been defined using standard TCP protocol mechanisms. Unlike TCP, a datagram PL requires additional mechanisms and considerations to implement PLPMTUD.

The requirements for datagram PLPMTUD are:

1. Managing the PLPMTU: For datagram PLs, the PLPMTU is managed by DPLPMTUD. A PL MUST NOT send a datagram (other than a probe packet) with a size at the PL that is larger than the current PLPMTU.
2. Probe packets: The network interface below PL is REQUIRED to provide a way to transmit a probe packet that is larger than the PLPMTU. In IPv4, a probe packet MUST be sent with the Don't Fragment (DF) bit set in the IP header, and without network layer endpoint fragmentation. In IPv6, a probe packet is always sent without source fragmentation (as specified in section 5.4 of [RFC8201]).
3. Reception feedback: The destination PL endpoint is REQUIRED to provide a feedback method that indicates to the DPLPMTUD sender when a probe packet has been received by the destination PL endpoint. Section 6 provides examples of how a PL can provide this acknowledgment of received probe packets.
4. Probe loss recovery: It is RECOMMENDED to use probe packets that do not carry any user data that would require retransmission if lost. Most datagram transports permit this. If a probe packet contains user data requiring retransmission in case of loss, the PL (or layers above) are REQUIRED to arrange any retransmission/repair of any resulting loss. The PL is REQUIRED to be robust in the case where probe packets are lost due to other reasons (including link transmission error, congestion).
5. PMTU parameters: A DPLPMTUD sender is RECOMMENDED to utilize information about the maximum size of packet that can be transmitted by the sender on the local link (e.g., the local Link



MTU). A PL sender MAY utilize similar information about the maximum size of network layer packet that a receiver can accept when this is supplied (note this could be less than EMTU\_R). This avoids implementations trying to send probe packets that can not be transferred by the local link. Too high of a value could reduce the efficiency of the search algorithm. Some applications also have a maximum transport protocol data unit (PDU) size, in which case there is no benefit from probing for a size larger than this (unless a transport allows multiplexing multiple applications PDUs into the same datagram).

6. Processing PTB messages: A DPLPMTUD sender MAY optionally utilize PTB messages received from the network layer to help identify when a network path does not support the current size of probe packet. Any received PTB message MUST be validated before it is used to update the PLPMTU discovery information [RFC8201]. This validation confirms that the PTB message was sent in response to a packet originating by the sender, and needs to be performed before the PLPMTU discovery method reacts to the PTB message. A PTB message MUST NOT be used to increase the PLPMTU [RFC8201], but could trigger a probe to test for a larger PLPMTU. A valid PTB\_SIZE is converted to a PL\_PTBSIZE before it is to be used in the DPLPMTUD state machine. A PL\_PTBSIZE that is greater than that currently probed SHOULD be ignored. (This PTB message ought to be discarded without further processing, but could be utilized as an input that enables a resilience mode).
7. Probing and congestion control: A PL MAY use a congestion controller to decide when to send a probe packet. If transmission of probe packets is limited by the congestion controller, this could result in transmission of probe packets being delayed or suspended during congestion. When the transmission of probe packets is not controlled by the congestion controller, the interval between probe packets MUST be at least one RTT. Loss of a probe packet SHOULD NOT be treated as an indication of congestion and SHOULD NOT trigger a congestion control reaction [RFC4821], because this could result in unnecessary reduction of the sending rate. An update to the PLPMTU (or MPS) MUST NOT increase the congestion window measured in bytes [RFC4821]. Therefore, an increase in the packet size does not cause an increase in the data rate in bytes per second. A PL that maintains the congestion window in terms of a limit to the number of outstanding fixed size packets SHOULD adapt this limit to compensate for the size of the actual packets. The transmission of probe packets can interact with the operation of a PL that performs burst mitigation or pacing and could need transmission of probe packets to be regulated by these methods.

8. Probing and flow control: Flow control at the PL concerns the end-to-end flow of data using the PL service. Flow control SHOULD NOT apply to DPLPMTU when probe packets use a design that does not carry user data to the remote application.
9. Shared PLPMTU state: The PMTU value calculated from the PLPMTU MAY also be stored with the corresponding entry associated with the destination in the IP layer cache, and used by other PL instances. The specification of PLPMTUD [RFC4821] states: "If PLPMTUD updates the MTU for a particular path, all Packetization Layer sessions that share the path representation (as described in Section 5.2 of [RFC4821]) SHOULD be notified to make use of the new MTU". Such methods MUST be robust to the wide variety of underlying network forwarding behaviors. Section 5.2 of [RFC8201] provides guidance on the caching of PMTU information and also the relation to IPv6 flow labels.

In addition, the following principles are stated for design of a DPLPMTUD method:

- \* A PL MAY be designed to segment data blocks larger than the MPS into multiple datagrams. However, not all datagram PLs support segmentation of data blocks. It is RECOMMENDED that methods avoid forcing an application to use an arbitrary small MPS for transmission while the method is searching for the currently supported PLPMTU. A reduced MPS can adversely impact the performance of an application.
- \* To assist applications in choosing a suitable data block size, the PL is RECOMMENDED to provide a primitive that returns the MPS derived from the PLPMTU to the higher layer using the PL. The value of the MPS can change following a change in the path, or loss of probe packets.
- \* Path validation: It is RECOMMENDED that methods are robust to path changes that could have occurred since the path characteristics were last confirmed, and to the possibility of inconsistent path information being received.
- \* Datagram reordering: A method is REQUIRED to be robust to the possibility that a flow encounters reordering, or the traffic (including probe packets) is divided over more than one network path.
- \* Datagram delay and duplication: The feedback mechanism is REQUIRED to be robust to the possibility that packets could be significantly delayed or duplicated along a network path.

- \* When to probe: It is RECOMMENDED that methods determine whether the path has changed since it last measured the path. This can help determine when to probe the path again.

#### 4. DPLPMTUD Mechanisms

This section lists the protocol mechanisms used in this specification.

##### 4.1. PLPMTU Probe Packets

The DPLPMTUD method relies upon the PL sender being able to generate probe packets with a specific size. TCP is able to generate these probe packets by choosing to appropriately segment data being sent [RFC4821]. In contrast, a datagram PL that constructs a probe packet has to either request an application to send a data block that is larger than that generated by an application, or to utilize padding functions to extend a datagram beyond the size of the application data block. Protocols that permit exchange of control messages (without an application data block) can generate a probe packet by extending a control message with padding data. The total size of a probe packet includes all headers and padding added to the payload data being sent (e.g., including protocol option fields, security-related fields such as an Authenticated Encryption with Associated Data (AEAD) tag and TLS record layer padding).

A receiver is REQUIRED to be able to distinguish an in-band data block from any added padding. This is needed to ensure that any added padding is not passed on to an application at the receiver.

This results in three possible ways that a sender can create a probe packet:

Probing using padding data: A probe packet that contains only control information together with any padding, which is needed to be inflated to the size of the probe packet. Since these probe packets do not carry an application-supplied data block, they do not typically require retransmission, although they do still consume network capacity and incur endpoint processing.

Probing using application data and padding data: A probe packet that contains a data block supplied by an application that is combined with padding to inflate the length of the datagram to the size of the probe packet.

Probing using application data: A probe packet that contains a data block supplied by an application that matches the size of the

probe packet. This method requests the application to issue a data block of the desired probe size.

A PL that uses a probe packet carrying application data and needs protection from the loss of this probe packet could perform transport-layer retransmission/repair of the data block (e.g., by retransmission after loss is detected or by duplicating the data block in a datagram without the padding data). This retransmitted data block might possibly need to be sent using a smaller PLPMTU, which could force the PL to use a smaller packet size to traverse the end-to-end path. (This could utilize endpoint network-layer fragmentation or a PL that can re-segment the data block into multiple datagrams).

DPLPMTUD MAY choose to use only one of these methods to simplify the implementation.

Probe messages sent by a PL MUST contain enough information to uniquely identify the probe within Maximum Segment Lifetime (e.g., including a unique identifier from the PL or the DPLPMTUD implementation), while being robust to reordering and replay of probe response and PTB messages.

#### 4.2. Confirmation of Probed Packet Size

The PL needs a method to determine (confirm) when probe packets have been successfully received end-to-end across a network path.

Transport protocols can include end-to-end methods that detect and report reception of specific datagrams that they send (e.g., DCCP, SCTP, and QUIC provide keep-alive/heartbeat features). When supported, this mechanism MAY also be used by DPLPMTUD to acknowledge reception of a probe packet.

A PL that does not acknowledge data reception (e.g., UDP and UDP-Lite) is unable itself to detect when the packets that it sends are discarded because their size is greater than the actual PMTU. These PLs need to rely on an application protocol to detect this loss.

Section 6 specifies this function for a set of IETF-specified protocols.

#### 4.3. Black Hole Detection and Reducing the PLPMTU

The description that follows uses the set of constants defined in Section 5.1.2 and variables defined in Section 5.1.3.

Black Hole Detection is triggered by an indication that the network path could be unable to support the current PLPMTU size.

There are three indicators that can detect black holes:

- \* A validated PTB message can be received that indicates a PL\_PTB\_SIZE less than the current PLPMTU. A DPLPMTUD method MUST NOT rely solely on this method.
- \* A PL can use the DPLPMTUD probing mechanism to periodically generate probe packets of the size of the current PLPMTU (e.g., using the confirmation timer Section 5.1.1). A timer tracks whether acknowledgments are received. Successive loss of probes is an indication that the current path no longer supports the PLPMTU (e.g., when the number of probe packets sent without receiving an acknowledgment, PROBE\_COUNT, becomes greater than MAX\_PROBES).
- \* A PL can utilize an event that indicates the network path no longer sustains the sender's PLPMTU size. This could use a mechanism implemented within the PL to detect excessive loss of data sent with a specific packet size and then conclude that this excessive loss could be a result of an invalid PLPMTU (as in PLPMTUD for TCP [RFC4821]).

The three methods can result in different transmission patterns for packet probes and are expected to result in different responsiveness following a change in the actual PMTU.

A PL MAY inhibit sending probe packets when no application data has been sent since the previous probe packet. A PL that resumes sending user data MAY continue PLPMTU discovery for each path. This allows it to use an up-to-date PLPMTU. However, this could result in additional packets being sent.

When the method detects the current PLPMTU is not supported, DPLPMTUD sets a lower PLPMTU, and sets a lower MPS. The PL then confirms that the new PLPMTU can be successfully used across the path. A probe packet could need to have a size less than the size of the data block generated by the application.

#### 4.4. The Maximum Packet Size (MPS)

The result of probing determines a usable PLPMTU, which is used to set the MPS used by the application. The MPS is smaller than the PLPMTU because it is reduced by the size of PL headers (including the overhead of security-related fields such as an AEAD tag and TLS record layer padding). The relationship between the MPS and the PLPMTU is illustrated in Figure 1.

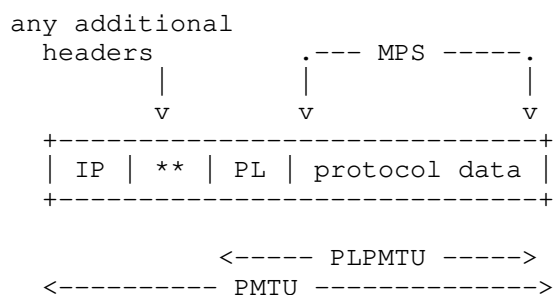


Figure 1: Relationship between MPS and PLPMTU

A PL is unable to send a packet (other than a probe packet) with a size larger than the current PLPMTU at the network layer. To avoid this, a PL MAY be designed to segment data blocks larger than the MPS into multiple datagrams.

DPLPMTUD seeks to avoid IP fragmentation. An attempt to send a data block larger than the MPS will therefore fail if a PL is unable to segment data. To determine the largest data block that can be sent, a PL SHOULD provide applications with a primitive that returns the MPS, derived from the current PLPMTU.

If DPLPMTUD results in a change to the MPS, the application needs to adapt to the new MPS. A particular case can arise when packets have been sent with a size less than the MPS and the PLPMTU was subsequently reduced. If these packets are lost, the PL MAY segment the data using the new MPS. If a PL is unable to re-segment a previously sent datagram (e.g., [RFC4960]), then the sender either discards the datagram or could perform retransmission using network-layer fragmentation to form multiple IP packets not larger than the PLPMTU. For IPv4, the use of endpoint fragmentation by the sender is preferred over clearing the DF bit in the IPv4 header. Operational experience reveals that IP fragmentation can reduce the reliability of Internet communication [I-D.ietf-intarea-frag-fragile], which may reduce the probability of successful retransmission.

#### 4.5. Disabling the Effect of PMTUD

A PL implementing this specification MUST suspend network layer processing of outgoing packets that enforces a PMTU [RFC1191][RFC8201] for each flow utilizing DPLPMTUD, and instead use DPLPMTUD to control the size of packets that are sent by a flow. This removes the need for the network layer to drop or fragment sent packets that have a size greater than the PMTU.

#### 4.6. Response to PTB Messages

This method requires the DPLPMTUD sender to validate any received PTB message before using the PTB information. The response to a PTB message depends on the PL\_PTB\_SIZE calculated from the PTB\_SIZE in the PTB message, the state of the PLPMTUD state machine, and the IP protocol being used.

Section 4.6.1 first describes validation for both IPv4 ICMP Unreachable messages (type 3) and ICMPv6 Packet Too Big messages, both of which are referred to as PTB messages in this document.

##### 4.6.1. Validation of PTB Messages

This section specifies utilization and validation of PTB messages.

- \* A simple implementation MAY ignore received PTB messages and in this case the PLPMTU is not updated when a PTB message is received.
- \* A PL that supports PTB messages MUST validate these messages before they are further processed.

A PL that receives a PTB message from a router or middlebox performs ICMP validation (see Section 4 of [RFC8201] and Section 5.2 of [BCP145]). Because DPLPMTUD operates at the PL, the PL needs to check that each received PTB message is received in response to a packet transmitted by the endpoint PL performing DPLPMTUD.

The PL MUST check the protocol information in the quoted packet carried in an ICMP PTB message payload to validate the message originated from the sending node. This validation includes determining that the combination of the IP addresses, the protocol, the source port and destination port match those returned in the quoted packet - this is also necessary for the PTB message to be passed to the corresponding PL.

The validation SHOULD utilize information that it is not simple for an off-path attacker to determine [BCP145]. For example, it could

check the value of a protocol header field known only to the two PL endpoints. A datagram application that uses well-known source and destination ports ought to also rely on other information to complete this validation.

These checks are intended to provide protection from packets that originate from a node that is not on the network path. A PTB message that does not complete the validation **MUST NOT** be further utilized by the DPLPMTUD method, as discussed in the Security Considerations section.

Section 4.6.2 describes this processing of PTB messages.

#### 4.6.2. Use of PTB Messages

PTB messages that have been validated **MAY** be utilized by the DPLPMTUD algorithm, but **MUST NOT** be used directly to set the PLPMTU.

Before using the size reported in the PTB message it must first be converted to a PL\_PTB\_SIZE. The PL\_PTB\_SIZE is smaller than the PTB\_SIZE because it is reduced by headers below the PL including any IP options or extensions added to the PL packet.

A method that utilizes these PTB messages can improve the speed at which the algorithm detects an appropriate PLPMTU by triggering an immediate probe for the PL\_PTB\_SIZE (resulting in a network-layer packet of size PTB\_SIZE), compared to one that relies solely on probing using a timer-based search algorithm.

A set of checks are intended to provide protection from a router that reports an unexpected PTB\_SIZE. The PL also needs to check that the indicated PL\_PTB\_SIZE is less than the size used by probe packets and at least the minimum size accepted.

This section provides a summary of how PTB messages can be utilized. (This uses the set of constants defined in Section 5.1.2). This processing depends on the PL\_PTB\_SIZE and the current value of a set of variables:

PL\_PTB\_SIZE < MIN\_PLPMTU

- \* Invalid PL\_PTB\_SIZE see Section 4.6.1.
- \* PTB message ought to be discarded without further processing (i.e., PLPMTU is not modified).
- \* The information could be utilized as an input that triggers enabling a resilience mode (see Section 5.3.3).



MIN\_PLPMTU < PL\_PTB\_SIZE < BASE\_PLPMTU

- \* A robust PL MAY enter an error state (see Section 5.2) for an IPv4 path when the PL\_PTB\_SIZE reported in the PTB message is larger than or equal to 68 bytes [RFC0791] and when this is less than the BASE\_PLPMTU.
- \* A robust PL MAY enter an error state (see Section 5.2) for an IPv6 path when the PL\_PTB\_SIZE reported in the PTB message is larger than or equal to 1280 bytes [RFC8200] and when this is less than the BASE\_PLPMTU.

BASE\_PLPMTU <= PL\_PTB\_SIZE < PLPMTU

- \* This could be an indication of a black hole. The PLPMTU SHOULD be set to BASE\_PLPMTU (the PLPMTU is reduced to the BASE\_PLPMTU to avoid unnecessary packet loss when a black hole is encountered).
- \* The PL ought to start a search to quickly discover the new PLPMTU. The PL\_PTB\_SIZE reported in the PTB message can be used to initialize a search algorithm.

PLPMTU < PL\_PTB\_SIZE < PROBED\_SIZE

- \* The PLPMTU continues to be valid, but the size of a packet used to search (PROBED\_SIZE) was larger than the actual PMTU.
- \* The PLPMTU is not updated.
- \* The PL can use the reported PL\_PTB\_SIZE from the PTB message as the next search point when it resumes the search algorithm.

PL\_PTB\_SIZE >= PROBED\_SIZE

- \* Inconsistent network signal.
- \* PTB message ought to be discarded without further processing (i.e., PLPMTU is not modified).
- \* The information could be utilized as an input to trigger enabling a resilience mode.

## 5. Datagram Packetization Layer PMTUD

This section specifies Datagram PLPMTUD (DPLPMTUD). The method can be introduced at various points (as indicated with \* in the figure below) in the IP protocol stack to discover the PLPMTU so that an application can utilize an appropriate MPS for the current network path.

DPLPMTUD SHOULD only be performed at one layer between a pair of endpoints. Therefore, an upper PL or application should avoid using DPLPMTUD when this is already enabled in a lower layer. A PL MUST adjust the MPS indicated by DPLPMTUD to account for any additional overhead introduced by the PL.

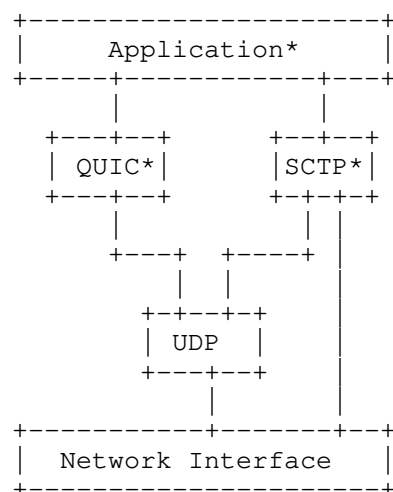


Figure 2: Examples where DPLPMTUD can be implemented

The central idea of DPLPMTUD is probing by a sender. Probe packets are sent to find the maximum size of user message that can be completely transferred across the network path from the sender to the destination.

The following sections identify the components needed for implementation, provides an overview of the phases of operation, and specifies the state machine and search algorithm.

## 5.1. DPLPMTUD Components

This section describes the timers, constants, and variables of DPLPMTUD.

### 5.1.1. Timers

The method utilizes up to three timers:

**PROBE\_TIMER:** The PROBE\_TIMER is configured to expire after a period longer than the maximum time to receive an acknowledgment to a probe packet. This value MUST NOT be smaller than 1 second, and SHOULD be larger than 15 seconds. Guidance on selection of the

timer value are provided in Section 3.1.1 of the UDP Usage Guidelines [BCP145].

**PMTU\_RAISE\_TIMER:** The PMTU\_RAISE\_TIMER is configured to the period a sender will continue to use the current PLPMTU, after which it re-enters the Search phase. This timer has a period of 600 seconds, as recommended by PLPMTUD [RFC4821].

DPLPMTUD MAY inhibit sending probe packets when no application data has been sent since the previous probe packet. A PL preferring to use an up-to-date PMTU once user data is sent again, can choose to continue PMTU discovery for each path. However, this will result in sending additional packets.

**CONFIRMATION\_TIMER:** When an acknowledged PL is used, this timer MUST NOT be used. For other PLs, the CONFIRMATION\_TIMER is configured to the period a PL sender waits before confirming the current PLPMTU is still supported. This is less than the PMTU\_RAISE\_TIMER and used to decrease the PLPMTU (e.g., when a black hole is encountered). Confirmation needs to be frequent enough when data is flowing that the sending PL does not black hole extensive amounts of traffic. Guidance on selection of the timer value are provided in Section 3.1.1 of the UDP Usage Guidelines [BCP145].

DPLPMTUD MAY inhibit sending probe packets when no application data has been sent since the previous probe packet. A PL preferring to use an up-to-date PMTU once user data is sent again, can choose to continue PMTU discovery for each path. However, this could result in sending additional packets.

DPLPMTD specifies various timers, however an implementation could choose to realise these timer functions using a single timer.

#### 5.1.2. Constants

The following constants are defined:

**MAX\_PROBES:** The MAX\_PROBES is the maximum value of the PROBE\_COUNT counter (see Section 5.1.3). MAX\_PROBES represents the limit for the number of consecutive probe attempts of any size. Search algorithms benefit from a MAX\_PROBES value greater than 1 because this can provide robustness to isolated packet loss. The default value of MAX\_PROBES is 3.

**MIN\_PLPMTU:** The MIN\_PLPMTU is the smallest size of PLPMTU that DPLPMTUD will attempt to use. An endpoint could need to be configure the MIN\_PLPMTU to provide space for extension headers and other encapsulations at layers below the PL. This value can

be interface and path dependent. For IPv6, this size is greater than or equal to the size at the PL that results in an 1280 byte IPv6 packet, as specified in [RFC8200]. For IPv4, this size is greater than or equal to the size at the PL that results in an 68 byte IPv4 packet. Note: An IPv4 router is required to be able to forward a datagram of 68 bytes without further fragmentation. This is the combined size of an IPv4 header and the minimum fragment size of 8 bytes. In addition, receivers are required to be able to reassemble fragmented datagrams at least up to 576 bytes, as stated in section 3.3.3 of [RFC1122].

**MAX\_PLPMTU:** The MAX\_PLPMTU is the largest size of PLPMTU. This has to be less than or equal to the maximum size of the PL packet that can be sent on the outgoing interface (constrained by the local interface MTU). When known, this also ought to be less than the maximum size of PL packet that can be received by the remote endpoint (constrained by EMTU\_R). It can be limited by the design or configuration of the PL being used. An application, or PL, MAY choose a smaller MAX\_PLPMTU when there is no need to send packets larger than a specific size.

**BASE\_PLPMTU:** The BASE\_PLPMTU is a configured size expected to work for most paths. The size is equal to or larger than the MIN\_PLPMTU and smaller than the MAX\_PLPMTU. For most PLs a suitable BASE\_PLPMTU will be larger than 1200 bytes. When using IPv4, there is no currently equivalent size specified and a default BASE\_PLPMTU of 1200 bytes is RECOMMENDED.

### 5.1.3. Variables

This method utilizes a set of variables:

**PROBED\_SIZE:** The PROBED\_SIZE is the size of the current probe packet as determined at the PL. This is a tentative value for the PLPMTU, which is awaiting confirmation by an acknowledgment.

**PROBE\_COUNT:** The PROBE\_COUNT is a count of the number of successive unsuccessful probe packets that have been sent. Each time a probe packet is acknowledged, the value is set to zero. (Some probe loss is expected while searching, therefore loss of a single probe is not an indication of a PMTU problem.)

The figure below illustrates the relationship between the packet size constants and variables at a point of time when the DPLPMTUD algorithm performs path probing to increase the size of the PLPMTU. A probe packet has been sent of size PROBED\_SIZE. Once this is acknowledged, the PLPMTU will raise to PROBED\_SIZE allowing the

DPLPMTUD algorithm to further increase PROBED\_SIZE toward sending a probe with the size of the actual PMTU.

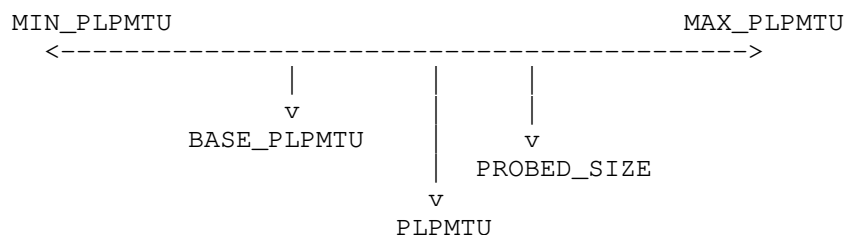


Figure 3: Relationships between packet size constants and variables

#### 5.1.4. Overview of DPLPMTUD Phases

This section provides a high-level informative view of the DPLPMTUD method, by describing the movement of the method through several phases of operation. More detail is available in the state machine Section 5.2.

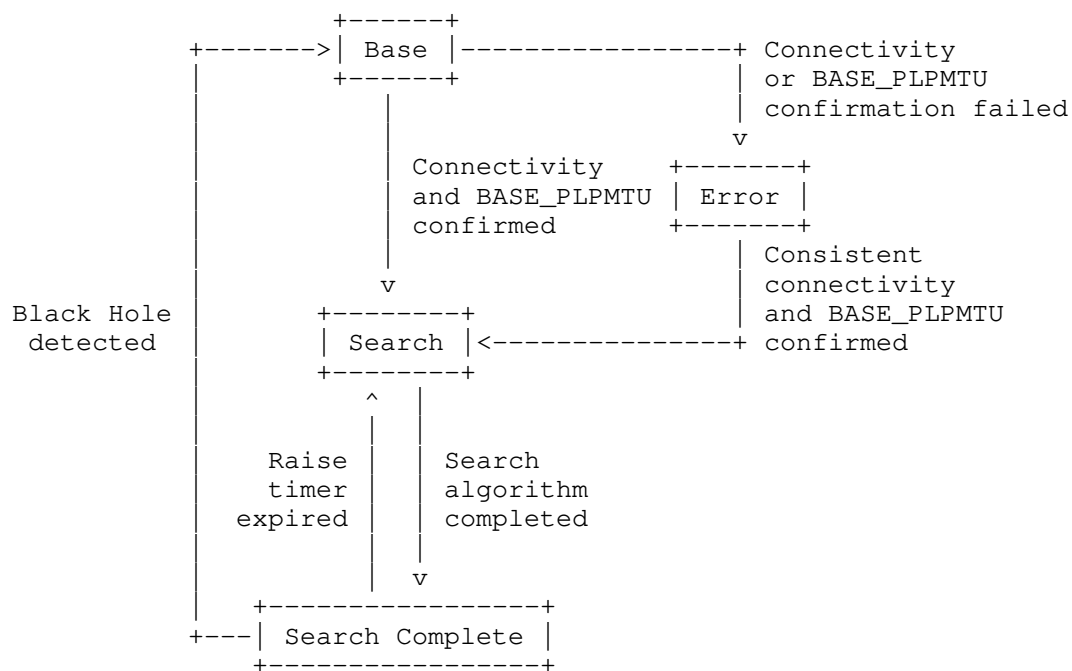


Figure 4: DPLPMTUD Phases

**Base:** The Base Phase confirms connectivity to the remote peer using packets of the `BASE_PLPMTU`. The confirmation of connectivity is implicit for a connection-oriented PL (where it can be performed in a PL connection handshake). A connectionless PL sends a probe packet and uses acknowledgment of this probe packet to confirm that the remote peer is reachable.

The sender also confirms that `BASE_PLPMTU` is supported across the network path. This may be achieved using a PL mechanism (e.g., using a handshake packet of size `BASE_PLPMTU`), or by sending a probe packet of size `BASE_PLPMTU` and confirming that this is received.

A probe packet of size `BASE_PLPMTU` can be sent immediately on the initial entry to the Base Phase (following a connectivity check). A PL that does not wish to support a path with a PLPMTU less than `BASE_PLPMTU` can simplify the phase into a single step by performing the connectivity checks with a probe of the `BASE_PLPMTU` size.

Once confirmed, DPLPMTUD enters the Search Phase. If the Base Phase fails to confirm the `BASE_PLPMTU`, DPLPMTUD enters the Error Phase.

**Search:** The Search Phase utilizes a search algorithm to send probe packets to seek to increase the PLPMTU. The algorithm concludes when it has found a suitable PLPMTU, by entering the Search Complete Phase.

A PL could respond to PTB messages using the PTB to advance or terminate the search, see Section 4.6.

**Search Complete:** The Search Complete Phase is entered when the PLPMTU is supported across the network path. A PL can use a `CONFIRMATION_TIMER` to periodically repeat a probe packet for the current PLPMTU size. If the sender is unable to confirm reachability (e.g., if the `CONFIRMATION_TIMER` expires) or the PL signals a lack of reachability, a black hole has been detected and DPLPMTUD enters the Base phase.

The `PMTU_RAISE_TIMER` is used to periodically resume the search phase to discover if the PLPMTU can be raised. Black Hole Detection causes the sender to enter the Base Phase.

**Error:** The Error Phase is entered when there is conflicting or invalid PLPMTU information for the path (e.g., a failure to support the `BASE_PLPMTU`) that cause DPLPMTUD to be unable to progress and the PLPMTU is lowered.

DPLPMTUD remains in the Error Phase until a consistent view of the path can be discovered and it has also been confirmed that the path supports the BASE\_PLPMTU (or DPLPMTUD is suspended).

A method that only reduces the PLPMTU to a suitable size would be sufficient to ensure reliable operation, but can be very inefficient when the actual PMTU changes or when the method (for whatever reason) makes a suboptimal choice for the PLPMTU.

A full implementation of DPLPMTUD provides an algorithm enabling the DPLPMTUD sender to increase the PLPMTU following a change in the characteristics of the path, such as when a link is reconfigured with a larger MTU, or when there is a change in the set of links traversed by an end-to-end flow (e.g., after a routing or path fail-over decision).

## 5.2. State Machine

A state machine for DPLPMTUD is depicted in Figure 5. If multipath or multihoming is supported, a state machine is needed for each path.

Note: Not all changes are shown to simplify the diagram.

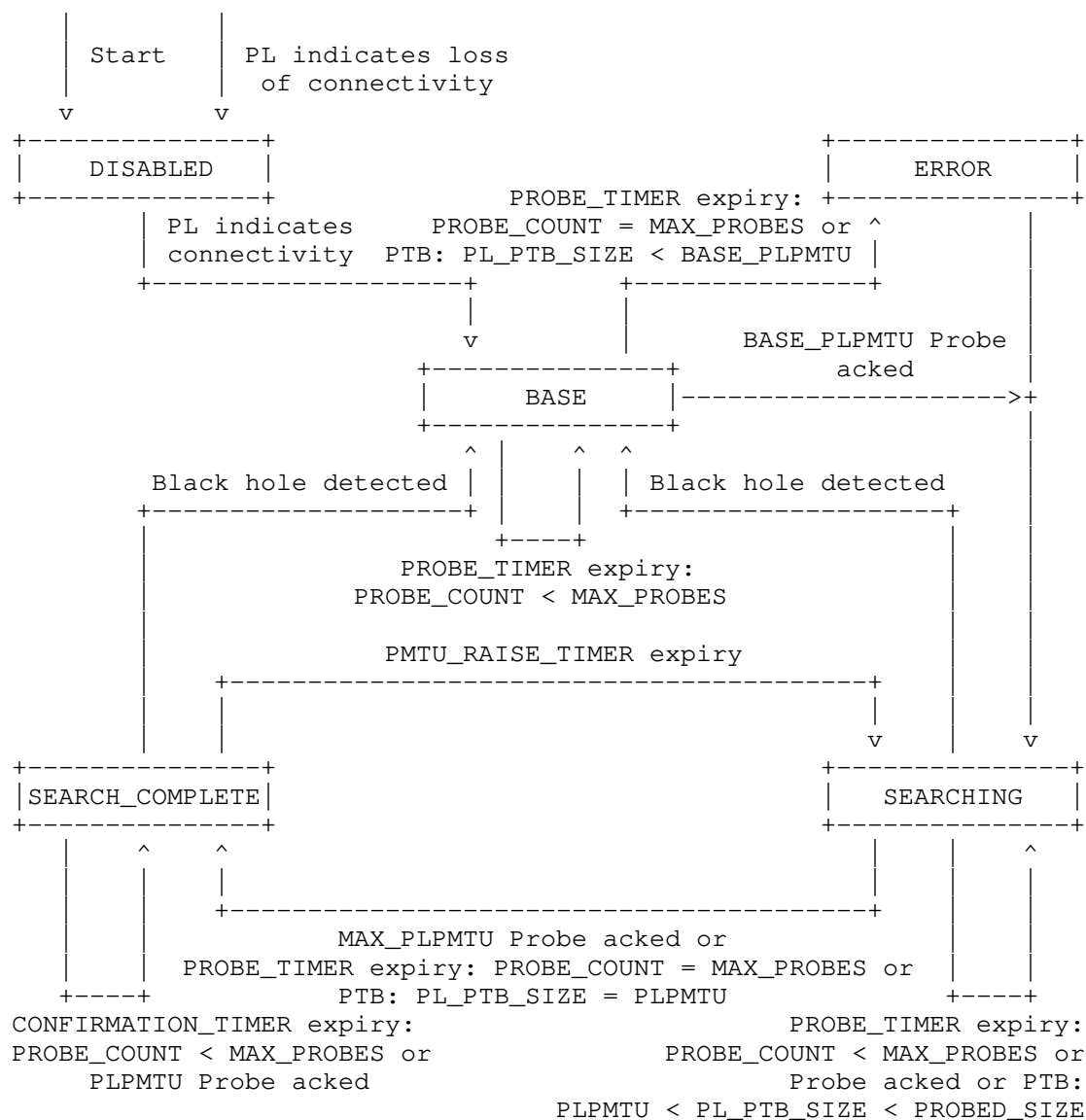


Figure 5: State machine for Datagram PLPMTUD

The following states are defined:

**DISABLED:** The DISABLED state is the initial state before probing has started. It is also entered from any other state, when the PL indicates loss of connectivity. This state is left once the PL



indicates connectivity to the remote PL. When transitioning to the BASE state, a probe packet of size BASE\_PLPMTU can be sent immediately.

**BASE:** The BASE state is used to confirm that the BASE\_PLPMTU size is supported by the network path and is designed to allow an application to continue working when there are transient reductions in the actual PMTU. It also seeks to avoid long periods when a sender searching for a larger PLPMTU is unaware that packets are not being delivered due to a packet or ICMP Black Hole.

On entry, the PROBED\_SIZE is set to the BASE\_PLPMTU size and the PROBE\_COUNT is set to zero.

Each time a probe packet is sent, the PROBE\_TIMER is started. The state is exited when the probe packet is acknowledged, and the PL sender enters the SEARCHING state.

The state is also left when the PROBE\_COUNT reaches MAX\_PROBES or a received PTB message is validated. This causes the PL sender to enter the ERROR state.

**SEARCHING:** The SEARCHING state is the main probing state. This state is entered when probing for the BASE\_PLPMTU completes.

Each time a probe packet is acknowledged, the PROBE\_COUNT is set to zero, the PLPMTU is set to the PROBED\_SIZE and then the PROBED\_SIZE is increased using the search algorithm (as described in Section 5.3).

When a probe packet is sent and not acknowledged within the period of the PROBE\_TIMER, the PROBE\_COUNT is incremented and a new probe packet is transmitted.

The state is exited to enter SEARCH\_COMPLETE when the PROBE\_COUNT reaches MAX\_PROBES, a validated PTB is received that corresponds to the last successfully probed size (PL\_PTB\_SIZE = PLPMTU), or a probe of size MAX\_PLPMTU is acknowledged (PLPMTU = MAX\_PLPMTU).

When a black hole is detected in the SEARCHING state, this causes the PL sender to enter the BASE state.

**SEARCH\_COMPLETE:** The SEARCH\_COMPLETE state indicates that a search has completed. This is the normal maintenance state, where the PL is not probing to update the PLPMTU. DPLPMTUD remains in this state until either the PMTU\_RAISE\_TIMER expires or a black hole is detected.

When DPLPMTUD uses an unacknowledged PL and is in the SEARCH\_COMPLETE state, a CONFIRMATION\_TIMER periodically resets the PROBE\_COUNT and schedules a probe packet with the size of the PLPMTU. If MAX\_PROBES successive PLPMTUD sized probes fail to be acknowledged the method enters the BASE state. When used with an acknowledged PL (e.g., SCTP), DPLPMTUD SHOULD NOT continue to generate PLPMTU probes in this state.

**ERROR:** The ERROR state represents the case where either the network path is not known to support a PLPMTU of at least the BASE\_PLPMTU size or when there is contradictory information about the network path that would otherwise result in excessive variation in the MPS signaled to the higher layer. The state implements a method to mitigate oscillation in the state-event engine. It signals a conservative value of the MPS to the higher layer by the PL. The state is exited when packet probes no longer detect the error. The PL sender then enters the SEARCHING state.

Implementations are permitted to enable endpoint fragmentation if the DPLPMTUD is unable to validate MIN\_PLPMTU within PROBE\_COUNT probes. If DPLPMTUD is unable to validate MIN\_PLPMTU the implementation will transition to the DISABLED state.

Note: MIN\_PLPMTU could be identical to BASE\_PLPMTU, simplifying the actions in this state.

### 5.3. Search to Increase the PLPMTU

This section describes the algorithms used by DPLPMTUD to search for a larger PLPMTU.

#### 5.3.1. Probing for a larger PLPMTU

Implementations use a search algorithm across the search range to determine whether a larger PLPMTU can be supported across a network path.

The method discovers the search range by confirming the minimum PLPMTU and then using the probe method to select a PROBED\_SIZE less than or equal to MAX\_PLPMTU. MAX\_PLPMTU is the minimum of the local MTU and EMTU\_R (when this is learned from the remote endpoint). The MAX\_PLPMTU MAY be reduced by an application that sets a maximum to the size of datagrams it will send.

The PROBE\_COUNT is initialized to zero when the first probe with a size greater than or equal to PLPMTUD is sent. Each probe packet successfully sent to the remote peer is confirmed by acknowledgment at the PL, see Section 4.1.

Each time a probe packet is sent to the destination, the PROBE\_TIMER is started. The timer is canceled when the PL receives acknowledgment that the probe packet has been successfully sent across the path Section 4.1. This confirms that the PROBED\_SIZE is supported, and the PROBED\_SIZE value is then assigned to the PLPMTU. The search algorithm can continue to send subsequent probe packets of an increasing size.

If the timer expires before a probe packet is acknowledged, the probe has failed to confirm the PROBED\_SIZE. Each time the PROBE\_TIMER expires, the PROBE\_COUNT is incremented, the PROBE\_TIMER is reinitialized, and a new probe of the same size or any other size (determined by the search algorithm) can be sent. The maximum number of consecutive failed probes is configured (MAX\_PROBES). If the value of the PROBE\_COUNT reaches MAX\_PROBES, probing will stop, and the PL sender enters the SEARCH\_COMPLETE state.

#### 5.3.2. Selection of Probe Sizes

The search algorithm determines a minimum useful gain in PLPMTU. It would not be constructive for a PL sender to attempt to probe for all sizes. This would incur unnecessary load on the path. Implementations SHOULD select the set of probe packet sizes to maximize the gain in PLPMTU from each search step.

Implementations could optimize the search procedure by selecting step sizes from a table of common PMTU sizes. When selecting the appropriate next size to search, an implementer ought to also consider that there can be common sizes of MPS that applications seek to use, and their could be common sizes of MTU used within the network.

#### 5.3.3. Resilience to Inconsistent Path Information

A decision to increase the PLPMTU needs to be resilient to the possibility that information learned about the network path is inconsistent. A path is inconsistent when, for example, probe packets are lost due to other reasons (i.e., not packet size) or due to frequent path changes. Frequent path changes could occur by unexpected "flapping" - where some packets from a flow pass along one path, but other packets follow a different path with different properties.

A PL sender is able to detect inconsistency from the sequence of PLPMTU probes that are acknowledged or the sequence of PTB messages that it receives. When inconsistent path information is detected, a PL sender could use an alternate search mode that clamps the offered

MPS to a smaller value for a period of time. This avoids unnecessary loss of packets.

#### 5.4. Robustness to Inconsistent Paths

Some paths could be unable to sustain packets of the BASE\_PLPMTU size. The Error State could be implemented to provide robustness to such paths. This allows fallback to a smaller than desired PLPMTU, rather than suffer connectivity failure. This could utilize methods such as endpoint IP fragmentation to enable the PL sender to communicate using packets smaller than the BASE\_PLPMTU.

### 6. Specification of Protocol-Specific Methods

DPLPMTUD requires protocol-specific details to be specified for each PL that is used.

The first subsection provides guidance on how to implement the DPLPMTUD method as a part of an application using UDP or UDP-Lite. The guidance also applies to other datagram services that do not include a specific transport protocol (such as a tunnel encapsulation). The following subsections describe how DPLPMTUD can be implemented as a part of the transport service, allowing applications using the service to benefit from discovery of the PLPMTU without themselves needing to implement this method when using SCTP and QUIC.

#### 6.1. Application support for DPLPMTUD with UDP or UDP-Lite

The current specifications of UDP [RFC0768] and UDP-Lite [RFC3828] do not define a method in the RFC-series that supports PLPMTUD. In particular, the UDP transport does not provide the transport features needed to implement datagram PLPMTUD.

The DPLPMTUD method can be implemented as a part of an application built directly or indirectly on UDP or UDP-Lite, but relies on higher-layer protocol features to implement the method [BCP145].

Some primitives used by DPLPMTUD might not be available via the Datagram API (e.g., the ability to access the PLPMTU from the IP layer cache, or interpret received PTB messages).

In addition, it is recommended that PMTU discovery is not performed by multiple protocol layers. An application SHOULD avoid using DPLPMTUD when the underlying transport system provides this capability. A common method for managing the PLPMTU has benefits, both in the ability to share state between different processes and opportunities to coordinate probing for different PL instances.

#### 6.1.1. Application Request

An application needs an application-layer protocol mechanism (such as a message acknowledgment method) that solicits a response from a destination endpoint. The method SHOULD allow the sender to check the value returned in the response to provide additional protection from off-path insertion of data [BCP145]. Suitable methods include a parameter known only to the two endpoints, such as a session ID or initialized sequence number.

#### 6.1.2. Application Response

An application needs an application-layer protocol mechanism to communicate the response from the destination endpoint. This response could indicate successful reception of the probe across the path, but could also indicate that some (or all packets) have failed to reach the destination.

#### 6.1.3. Sending Application Probe Packets

A probe packet can carry an application data block, but the successful transmission of this data is at risk when used for probing. Some applications might prefer to use a probe packet that does not carry an application data block to avoid disruption to data transfer.

#### 6.1.4. Initial Connectivity

An application that does not have other higher-layer information confirming connectivity with the remote peer SHOULD implement a connectivity mechanism using acknowledged probe packets before entering the BASE state.

#### 6.1.5. Validating the Path

An application that does not have other higher-layer information confirming correct delivery of datagrams SHOULD implement the CONFIRMATION\_TIMER to periodically send probe packets while in the SEARCH\_COMPLETE state.

#### 6.1.6. Handling of PTB Messages

An application that is able and wishes to receive PTB messages MUST perform ICMP validation as specified in Section 5.2 of [BCP145]. This requires that the application checks each received PTB message to validate that it was received in response to transmitted traffic and that the reported PL\_PTB\_SIZE is less than the current probed size (see Section 4.6.2). A validated PTB message MAY be used

as input to the DPLPMTUD algorithm, but MUST NOT be used directly to set the PLPMTU.

## 6.2. DPLPMTUD for SCTP

Section 10.2 of [RFC4821] specified a recommended PLPMTUD probing method for SCTP and Section 7.3 of [RFC4960] recommended an endpoint apply the techniques in RFC4821 on a per-destination-address basis. The specification for DPLPMTUD continues the practice of using the PL to discover the PMTU, but updates, RFC4960 with a recommendation to use the method specified in this document: The RECOMMENDED method for generating probes is to add a chunk consisting only of padding to an SCTP message. The PAD chunk defined in [RFC4820] SHOULD be attached to a minimum length HEARTBEAT (HB) chunk to build a probe packet. This enables probing without affecting the transfer of user messages and without being limited by congestion control or flow control. This is preferred to using DATA chunks (with padding as required) as path probes.

Section 6.9 of [RFC4960] describes dividing the user messages into data chunks sent by the PL when using SCTP. This notes that once an SCTP message has been sent, it cannot be re-segmented. [RFC4960] describes the method to retransmit data chunks when the MPS has reduced, and the use of IP fragmentation for this case. This is unchanged by this document.

### 6.2.1. SCTP/IPv4 and SCTP/IPv6

#### 6.2.1.1. Initial Connectivity

The base protocol is specified in [RFC4960]. This provides an acknowledged PL. A sender can therefore enter the BASE state as soon as connectivity has been confirmed.

#### 6.2.1.2. Sending SCTP Probe Packets

Probe packets consist of an SCTP common header followed by a HEARTBEAT chunk and a PAD chunk. The PAD chunk is used to control the length of the probe packet. The HEARTBEAT chunk is used to trigger the sending of a HEARTBEAT ACK chunk. The reception of the HEARTBEAT ACK chunk acknowledges reception of a successful probe. A successful probe updates the association and path counters, but an unsuccessful probe is discounted (assumed to be a result of choosing too large a PLPMTU).

The SCTP sender needs to be able to determine the total size of a probe packet. The HEARTBEAT chunk could carry a Heartbeat Information parameter that includes, besides the information

suggested in [RFC4960], the probe size to help an implementation associate a HEARTBEAT-ACK with the size of probe that was sent. The sender could also use other methods, such as sending a nonce and verifying the information returned also contains the corresponding nonce. The length of the PAD chunk is computed by reducing the probing size by the size of the SCTP common header and the HEARTBEAT chunk. The payload of the PAD chunk contains arbitrary data. When transmitted at the IP layer, the PMTU size also includes the IPv4 or IPv6 header(s).

Probing can start directly after the PL handshake, this can be done before data is sent. Assuming this behavior (i.e., the PMTU is smaller than or equal to the interface MTU), this process will take several round trip time periods, dependent on the number of DPLPMTUD probes sent. The Heartbeat timer can be used to implement the PROBE\_TIMER.

#### 6.2.1.3. Validating the Path with SCTP

Since SCTP provides an acknowledged PL, a sender MUST NOT implement the CONFIRMATION\_TIMER while in the SEARCH\_COMPLETE state.

#### 6.2.1.4. PTB Message Handling by SCTP

Normal ICMP validation MUST be performed as specified in Appendix C of [RFC4960]. This requires that the first 8 bytes of the SCTP common header are quoted in the payload of the PTB message, which can be the case for ICMPv4 and is normally the case for ICMPv6.

When a PTB message has been validated, the PL\_PTB\_SIZE calculated from the PTB\_SIZE reported in the PTB message SHOULD be used with the DPLPMTUD algorithm, providing that the reported PL\_PTB\_SIZE is less than the current probe size (see Section 4.6).

#### 6.2.2. DPLPMTUD for SCTP/UDP

The UDP encapsulation of SCTP is specified in [RFC6951].

This specification updates the reference to RFC 4821 in section 5.6 of RFC 6951 to refer to XXXTHISRFCXXX. RFC 6951 is updated by addition of the following sentence at the end of section 5.6: "The RECOMMENDED method for determining the MTU of the path is specified in XXXTHISRFCXXX".

XXX RFC EDITOR - please replace XXXTHISRFCXXX when published XXX

#### 6.2.2.1. Initial Connectivity

A sender can enter the BASE state as soon as SCTP connectivity has been confirmed.

#### 6.2.2.2. Sending SCTP/UDP Probe Packets

Packet probing can be performed as specified in Section 6.2.1.2. The size of the probe packet includes the 8 bytes of UDP Header. This has to be considered when filling the probe packet with the PAD chunk.

#### 6.2.2.3. Validating the Path with SCTP/UDP

SCTP provides an acknowledged PL, therefore a sender does not implement the CONFIRMATION\_TIMER while in the SEARCH\_COMPLETE state.

#### 6.2.2.4. Handling of PTB Messages by SCTP/UDP

ICMP validation MUST be performed for PTB messages as specified in Appendix C of [RFC4960]. This requires that the first 8 bytes of the SCTP common header are contained in the PTB message, which can be the case for ICMPv4 (but note the UDP header also consumes a part of the quoted packet header) and is normally the case for ICMPv6. When the validation is completed, the PL\_PTB\_SIZE calculated from the PTB\_SIZE in the PTB message SHOULD be used with the DPLPMTUD providing that the reported PL\_PTB\_SIZE is less than the current probe size.

#### 6.2.3. DPLPMTUD for SCTP/DTLS

The Datagram Transport Layer Security (DTLS) encapsulation of SCTP is specified in [RFC8261]. This is used for data channels in WebRTC implementations. This specification updates the reference to RFC 4821 in section 5 of RFC 8261 to refer to XXXTHISRFCXXX.

XXX RFC EDITOR - please replace XXXTHISRFCXXX when published XXX

#### 6.2.3.1. Initial Connectivity

A sender can enter the BASE state as soon as SCTP connectivity has been confirmed.



#### 6.2.3.2. Sending SCTP/DTLS Probe Packets

Packet probing can be done, as specified in Section 6.2.1.2. The maximum payload is reduced by the size of the DTLS headers, which has to be considered when filling the PAD chunk. The size of the probe packet includes the DTLS PL headers. This has to be considered when filling the probe packet with the PAD chunk.

#### 6.2.3.3. Validating the Path with SCTP/DTLS

Since SCTP provides an acknowledged PL, a sender MUST NOT implement the CONFIRMATION\_TIMER while in the SEARCH\_COMPLETE state.

#### 6.2.3.4. Handling of PTB Messages by SCTP/DTLS

[RFC4960] does not specify a way to validate SCTP/DTLS ICMP message payload and neither does this document. This can prevent processing of PTB messages at the PL.

### 6.3. DPLPMTUD for QUIC

QUIC [I-D.ietf-quic-transport] is a UDP-based PL that provides reception feedback. The UDP payload includes a QUIC packet header, a protected payload, and any authentication fields. It supports padding and packet coalescence that can be used to construct probe packets. From the perspective of DPLPMTUD, QUIC can function as an acknowledged PL. [I-D.ietf-quic-transport] describes the method for using DPLPMTUD with QUIC packets.

## 7. Acknowledgments

This work was partially funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644334 (NEAT). The views expressed are solely those of the author(s).

Thanks to all that have commented or contributed, the TSVWG and QUIC working groups, and Mathew Calder and Julius Flohr for providing early implementations.

## 8. IANA Considerations

This memo includes no request to IANA.

If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

## 9. Security Considerations

The security considerations for the use of UDP and SCTP are provided in the referenced RFCs.

To avoid excessive load, the interval between individual probe packets **MUST** be at least one RTT, and the interval between rounds of probing is determined by the `PMTU_RAISE_TIMER`.

A PL sender needs to ensure that the method used to confirm reception of probe packets protects from off-path attackers injecting packets into the path. This protection is provided in IETF-defined protocols (e.g., TCP, SCTP) using a randomly-initialized sequence number. A description of one way to do this when using UDP is provided in section 5.1 of [BCP145]).

There are cases where ICMP Packet Too Big (PTB) messages are not delivered due to policy, configuration or equipment design (see Section 1.1). This method therefore does not rely upon PTB messages being received, but is able to utilize these when they are received by the sender. PTB messages could potentially be used to cause a node to inappropriately reduce the PLPMTU. A node supporting DPLPMTUD **MUST** therefore appropriately validate the payload of PTB messages to ensure these are received in response to transmitted traffic (i.e., a reported error condition that corresponds to a datagram actually sent by the path layer, see Section 4.6.1).

An on-path attacker able to create a PTB message could forge PTB messages that include a valid quoted IP packet. Such an attack could be used to drive down the PLPMTU. An on-path device could similarly force a reduction of the PLPMTU by implementing a policy that drops packets larger than a configured size. There are two ways this method can be mitigated against such attacks: First, by ensuring that a PL sender never reduces the PLPMTU below the base size, solely in response to receiving a PTB message. This is achieved by first entering the `BASE` state when such a message is received. Second, the design does not require processing of PTB messages, a PL sender could therefore suspend processing of PTB messages (e.g., in a robustness mode after detecting that subsequent probes actually confirm that a size larger than the `PTB_SIZE` is supported by a path).

Parsing the quoted packet inside a PTB message can introduce additional per-packet processing at the PL sender. This processing **SHOULD** be limited to avoid a denial of service attack when arbitrary headers are included. Rate-limiting the processing could result in PTB messages not being received by a PL, however the DPLPMTUD method is robust to such loss.

The successful processing of an ICMP message can trigger a probe when the reported PTB size is valid, but this does not directly update the PLPMTU for the path. This prevents a message attempting to black hole data by indicating a size larger than supported by the path.

It is possible that the information about a path is not stable. This could be a result of forwarding across more than one path that has a different actual PMTU or a single path presents a varying PMTU. The design of a PLPMTUD implementation SHOULD consider how to mitigate the effects of varying path information. One possible mitigation is to provide robustness (see Section 5.4) in the method that avoids oscillation in the MPS.

DPLPMTUD methods can introduce padding data to inflate the length of the datagram to the total size required for a probe packet. The total size of a probe packet includes all headers and padding added to the payload data being sent (e.g., including security-related fields such as an AEAD tag and TLS record layer padding). The value of the padding data does not influence the DPLPMTUD search algorithm, and therefore needs to be set consistent with the policy of the PL.

If a PL can make use of cryptographic confidentiality or data-integrity mechanisms, then the design ought to avoid adding anything (e.g., padding) to DPLPMTUD probe packets that is not also protected by those cryptographic mechanisms.

## 10. References

### 10.1. Normative References

- [BCP145] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, March 2017.  
<<https://www.rfc-editor.org/info/bcp145>>
- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980,  
<<https://www.rfc-editor.org/info/rfc768>>.
- [RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981,  
<<https://www.rfc-editor.org/info/rfc791>>.
- [RFC1191] Mogul, J.C. and S.E. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990,  
<<https://www.rfc-editor.org/info/rfc1191>>.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3828] Larzon, L-A., Degermark, M., Pink, S., Jonsson, L-E., Ed., and G. Fairhurst, Ed., "The Lightweight User Datagram Protocol (UDP-Lite)", RFC 3828, DOI 10.17487/RFC3828, July 2004, <<https://www.rfc-editor.org/info/rfc3828>>.
- [RFC4820] Tuexen, M., Stewart, R., and P. Lei, "Padding Chunk and Parameter for the Stream Control Transmission Protocol (SCTP)", RFC 4820, DOI 10.17487/RFC4820, March 2007, <<https://www.rfc-editor.org/info/rfc4820>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.
- [RFC6951] Tuexen, M. and R. Stewart, "UDP Encapsulation of Stream Control Transmission Protocol (SCTP) Packets for End-Host to End-Host Communication", RFC 6951, DOI 10.17487/RFC6951, May 2013, <<https://www.rfc-editor.org/info/rfc6951>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <<https://www.rfc-editor.org/info/rfc8200>>.
- [RFC8201] McCann, J., Deering, S., Mogul, J., and R. Hinden, Ed., "Path MTU Discovery for IP version 6", STD 87, RFC 8201, DOI 10.17487/RFC8201, July 2017, <<https://www.rfc-editor.org/info/rfc8201>>.
- [RFC8261] Tuexen, M., Stewart, R., Jesup, R., and S. Loreto, "Datagram Transport Layer Security (DTLS) Encapsulation of SCTP Packets", RFC 8261, DOI 10.17487/RFC8261, November 2017, <<https://www.rfc-editor.org/info/rfc8261>>.

## 10.2. Informative References

- [I-D.ietf-intarea-frag-fragile]  
Bonica, R., Baker, F., Huston, G., Hinden, R., Troan, O.,

and F. Gont, "IP Fragmentation Considered Fragile", Work in Progress, Internet-Draft, draft-ietf-intarea-frag-fragile-17, 30 September 2019, <<http://www.ietf.org/internet-drafts/draft-ietf-intarea-frag-fragile-17.txt>>.

[I-D.ietf-intarea-tunnels]

Touch, J. and M. Townsley, "IP Tunnels in the Internet Architecture", Work in Progress, Internet-Draft, draft-ietf-intarea-tunnels-10, 12 September 2019, <<http://www.ietf.org/internet-drafts/draft-ietf-intarea-tunnels-10.txt>>.

[I-D.ietf-quic-transport]

Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-27, 21 February 2020, <<http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-27.txt>>.

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, DOI 10.17487/RFC0792, September 1981, <<https://www.rfc-editor.org/info/rfc792>>.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.

[RFC1812] Baker, F., Ed., "Requirements for IP Version 4 Routers", RFC 1812, DOI 10.17487/RFC1812, June 1995, <<https://www.rfc-editor.org/info/rfc1812>>.

[RFC2923] Lahey, K., "TCP Problems with Path MTU Discovery", RFC 2923, DOI 10.17487/RFC2923, September 2000, <<https://www.rfc-editor.org/info/rfc2923>>.

[RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI 10.17487/RFC4340, March 2006, <<https://www.rfc-editor.org/info/rfc4340>>.

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <<https://www.rfc-editor.org/info/rfc4443>>.

- [RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, DOI 10.17487/RFC4821, March 2007, <<https://www.rfc-editor.org/info/rfc4821>>.
- [RFC4890] Davies, E. and J. Mohacsi, "Recommendations for Filtering ICMPv6 Messages in Firewalls", RFC 4890, DOI 10.17487/RFC4890, May 2007, <<https://www.rfc-editor.org/info/rfc4890>>.
- [RFC5508] Srisuresh, P., Ford, B., Sivakumar, S., and S. Guha, "NAT Behavioral Requirements for ICMP", BCP 148, RFC 5508, DOI 10.17487/RFC5508, April 2009, <<https://www.rfc-editor.org/info/rfc5508>>.

#### Appendix A. Revision Notes

Note to RFC-Editor: please remove this entire section prior to publication.

Individual draft -00:

- \* Comments and corrections are welcome directly to the authors or via the IETF TSVWG working group mailing list.
- \* This update is proposed for WG comments.

Individual draft -01:

- \* Contains the first representation of the algorithm, showing the states and timers
- \* This update is proposed for WG comments.

Individual draft -02:

- \* Contains updated representation of the algorithm, and textual corrections.
- \* The text describing when to set the effective PMTU has not yet been validated by the authors
- \* To determine security to off-path-attacks: We need to decide whether a received PTB message SHOULD/MUST be validated? The text on how to handle a PTB message indicating a link MTU larger than the probe has yet not been validated by the authors
- \* No text currently describes how to handle inconsistent results from arbitrary re-routing along different parallel paths

- \* This update is proposed for WG comments.

Working Group draft -00:

- \* This draft follows a successful adoption call for TSVWG
- \* There is still work to complete, please comment on this draft.

Working Group draft -01:

- \* This draft includes improved introduction.
- \* The draft is updated to require ICMP validation prior to accepting PTB messages - this to be confirmed by WG
- \* Section added to discuss Selection of Probe Size - methods to be evaluated and recommendations to be considered
- \* Section added to align with work proposed in the QUIC WG.

Working Group draft -02:

- \* The draft was updated based on feedback from the WG, and a detailed review by Magnus Westerlund.
- \* The document updates RFC 4821.
- \* Requirements list updated.
- \* Added more explicit discussion of a simpler black-hole detection mode.
- \* This draft includes reorganisation of the section on IETF protocols.
- \* Added more discussion of implementation within an application.
- \* Added text on flapping paths.
- \* Replaced 'effective MTU' with new term PLPMTU.

Working Group draft -03:

- \* Updated figures
- \* Added more discussion on blackhole detection
- \* Added figure describing just blackhole detection

- \* Added figure relating MPS sizes

Working Group draft -04:

- \* Described phases and named these consistently.
- \* Corrected transition from confirmation directly to the search phase (Base has been checked).
- \* Redrawn state diagrams.
- \* Renamed BASE\_MTU to BASE\_PMTU (because it is a base for the PMTU).
- \* Clarified Error state.
- \* Clarified suspending DPLPMTUD.
- \* Verified normative text in requirements section.
- \* Removed duplicate text.
- \* Changed all text to refer to /packet probe/probe packet/  
/validation/verification/ added term /Probe Confirmation/ and  
clarified BlackHole detection.

Working Group draft -05:

- \* Updated security considerations.
- \* Feedback after speaking with Joe Touch helped improve UDP-Options description.

Working Group draft -06:

- \* Updated description of ICMP issues in section 1.1
- \* Update to description of QUIC.

Working group draft -07:

- \* Moved description of the PTB processing method from the PTB requirements section.
- \* Clarified what is performed in the PTB validation check.
- \* Updated security consideration to explain PTB security without needing to read the rest of the document.



- \* Reformatted state machine diagram

Working group draft -08:

- \* Moved to rfcxml v3+
- \* Rendered diagrams to svg in html version.
- \* Removed Appendix A. Event-driven state changes.
- \* Removed section on DPLPMTUD with UDP Options.
- \* Shortened the description of phases.

Working group draft -09:

- \* Remove final mention of UDP Options
- \* Add Initial Connectivity sections to each PL
- \* Add to disable outgoing pmtu enforcement of packets

Working group draft -10:

- \* Address comments from Lars Eggert
- \* Reinforce that PROBE\_COUNT is successive attempts to probe for any size
- \* Redefine MAX\_PROBES to 3
- \* Address PTB\_SIZE of 0 or less than MIN\_PLPMTU

Working group draft -11:

- \* Restore a sentence removed in previous rev
- \* De-acronymise QUIC
- \* Address some nits

Working group draft -12:

- \* Add TSVWG, QUIC and implementers to acknowledgments
- \* Shorten a diagram line.
- \* Address nits from Julius and Wes.

- \* Be clearer when talking about IP layer caches

Working group draft -13, -14:

- \* Updated after WGLC.

Working group draft -15:

- \* Updated after AD evaluation and prepared for IETF-LC.

Working group draft -16:

- \* Updated text after SECDIR review.

Working group draft -17:

- \* Updated text after GENART and IETF-LC.

- \* Renamed BASE\_MTU to BASE\_PLPMTU, and MIN and MAX PMTU to PLPMTU (because these are about a base for the PLPMTU), and ensured consistent separation of PMTU and PLPMTU.

- \* Adopted US-style English throughout.

Working group draft -18:

- \* Updated text and address nits from OPSDIR, ART and IESG reviews.

- \* Order PTB processing based on PL\_PTB\_SIZE

Working group draft -19:

- \* Updated text and address nits based on comments from Tim Chown and Murray S. Kucherawy.

Working group draft -20:

- \* Address nits and comments from IESG

- \* Refer to BCP 145 rather than RFC 8085 in most places.

- \* Update probing method text for SCTP and QUIC.

Working group draft -21:

- \* Update QUIC text for skipping into BASE state.

Working group draft -22:

- \* Add a section reference to MPS
- \* Clarify MIN\_PLPMTU text
- \* Remove most QUIC text
- \* Make QUIC reference informative.

Authors' Addresses

Godred Fairhurst  
University of Aberdeen  
School of Engineering  
Fraser Noble Building  
Aberdeen  
AB24 3UE  
United Kingdom

Email: [gorry@erg.abdn.ac.uk](mailto:gorry@erg.abdn.ac.uk)

Tom Jones  
University of Aberdeen  
School of Engineering  
Fraser Noble Building  
Aberdeen  
AB24 3UE  
United Kingdom

Email: [tom@erg.abdn.ac.uk](mailto:tom@erg.abdn.ac.uk)

Michael Tuexen  
Muenster University of Applied Sciences  
Stegerwaldstrasse 39  
48565 Steinfurt  
Germany

Email: [tuexen@fh-muenster.de](mailto:tuexen@fh-muenster.de)

Irene Ruengeler  
Muenster University of Applied Sciences  
Stegerwaldstrasse 39  
48565 Steinfurt  
Germany

Email: [i.ruengeler@fh-muenster.de](mailto:i.ruengeler@fh-muenster.de)

Timo Voelker  
Muenster University of Applied Sciences  
Stegerwaldstrasse 39  
48565 Steinfurt  
Germany

Email: [timo.voelker@fh-muenster.de](mailto:timo.voelker@fh-muenster.de)

Transport Area Working Group  
Internet-Draft  
Updates: 3819 (if approved)  
Intended status: Best Current Practice  
Expires: November 26, 2021

B. Briscoe  
Independent  
J. Kaippallimalil  
Futurewei  
May 25, 2021

Guidelines for Adding Congestion Notification to Protocols that  
Encapsulate IP  
draft-ietf-tsvwg-ecn-encap-guidelines-16

## Abstract

The purpose of this document is to guide the design of congestion notification in any lower layer or tunnelling protocol that encapsulates IP. The aim is for explicit congestion signals to propagate consistently from lower layer protocols into IP. Then the IP internetwork layer can act as a portability layer to carry congestion notification from non-IP-aware congested nodes up to the transport layer (L4). Following these guidelines should assure interworking among IP layer and lower layer congestion notification mechanisms, whether specified by the IETF or other standards bodies. This document updates the advice to subnetwork designers about ECN in RFC 3819.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 26, 2021.

## Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Update to RFC 3819 . . . . .	5
1.2. Scope . . . . .	5
2. Terminology . . . . .	7
3. Modes of Operation . . . . .	9
3.1. Feed-Forward-and-Up Mode . . . . .	9
3.2. Feed-Up-and-Forward Mode . . . . .	11
3.3. Feed-Backward Mode . . . . .	12
3.4. Null Mode . . . . .	14
4. Feed-Forward-and-Up Mode: Guidelines for Adding Congestion Notification . . . . .	14
4.1. IP-in-IP Tunnels with Shim Headers . . . . .	15
4.2. Wire Protocol Design: Indication of ECN Support . . . . .	16
4.3. Encapsulation Guidelines . . . . .	18
4.4. Decapsulation Guidelines . . . . .	20
4.5. Sequences of Similar Tunnels or Subnets . . . . .	22
4.6. Reframing and Congestion Markings . . . . .	22
5. Feed-Up-and-Forward Mode: Guidelines for Adding Congestion Notification . . . . .	23
6. Feed-Backward Mode: Guidelines for Adding Congestion Notification . . . . .	24
7. IANA Considerations . . . . .	25
8. Security Considerations . . . . .	25
9. Conclusions . . . . .	26
10. Acknowledgements . . . . .	27
11. Contributors . . . . .	27
12. Comments Solicited . . . . .	27
13. References . . . . .	27
13.1. Normative References . . . . .	27
13.2. Informative References . . . . .	28
Appendix A. Changes in This Version (to be removed by RFC Editor) . . . . .	33
Authors' Addresses . . . . .	38

## 1. Introduction

The benefits of Explicit Congestion Notification (ECN) described in [RFC8087] and summarized below can only be fully realized if support for ECN is added to the relevant subnetwork technology, as well as to IP. When a lower layer buffer drops a packet obviously it does not just drop at that layer; the packet disappears from all layers. In contrast, when active queue management (AQM) at a lower layer marks a packet with ECN, the marking needs to be explicitly propagated up the layers. The same is true if AQM marks the outer header of a packet that encapsulates inner tunnelled headers. Forwarding ECN is not as straightforward as other headers because it has to be assumed ECN may be only partially deployed. If a lower layer header that contains ECN congestion indications is stripped off by a subnet egress that is not ECN-aware, or if the ultimate receiver or sender is not ECN-aware, congestion needs to be indicated by dropping a packet, not marking it.

The purpose of this document is to guide the addition of congestion notification to any subnet technology or tunnelling protocol, so that lower layer AQM algorithms can signal congestion explicitly and it will propagate consistently into encapsulated (higher layer) headers, otherwise the signals will not reach their ultimate destination.

ECN is defined in the IP header (v4 and v6) [RFC3168] to allow a resource to notify the onset of queue build-up without having to drop packets, by explicitly marking a proportion of packets with the congestion experienced (CE) codepoint.

Given a suitable marking scheme, ECN removes nearly all congestion loss and it cuts delays for two main reasons:

- o It avoids the delay when recovering from congestion losses, which particularly benefits small flows or real-time flows, making their delivery time predictably short [RFC2884];
- o As ECN is used more widely by end-systems, it will gradually remove the need to configure a degree of delay into buffers before they start to notify congestion (the cause of bufferbloat). This is because drop involves a trade-off between sending a timely signal and trying to avoid impairment, whereas ECN is solely a signal not an impairment, so there is no harm triggering it earlier.

Some lower layer technologies (e.g. MPLS, Ethernet) are used to form subnetworks with IP-aware nodes only at the edges. These networks are often sized so that it is rare for interior queues to overflow. However, until recently this was more due to the inability of TCP to

saturate the links. For many years, fixes such as window scaling [RFC7323] proved hard to deploy. And the Reno variant of TCP has remained in widespread use despite its inability to scale to high flow rates. However, now that modern operating systems are finally capable of saturating interior links, even the buffers of well-provisioned interior switches will need to signal episodes of queuing.

Propagation of ECN is defined for MPLS [RFC5129], and is being defined for TRILL [RFC7780], [I-D.ietf-trill-ecn-support], but it remains to be defined for a number of other subnetwork technologies.

Similarly, ECN propagation is yet to be defined for many tunnelling protocols. [RFC6040] defines how ECN should be propagated for IP-in-IPv4 [RFC2003], IP-in-IPv6 [RFC2473] and IPsec [RFC4301] tunnels, but there are numerous other tunnelling protocols with a shim and/or a layer 2 header between two IP headers (v4 or v6). Some address ECN propagation between the IP headers, but many do not. This document gives guidance on how to address ECN propagation for future tunnelling protocols, and a companion standards track specification [I-D.ietf-tsvwg-rfc6040update-shim] updates those existing IP-shim-(L2)-IP protocols that are under IETF change control and still widely used.

Incremental deployment is the most delicate aspect when adding support for ECN. The original ECN protocol in IP [RFC3168] was carefully designed so that a congested buffer would not mark a packet (rather than drop it) unless both source and destination hosts were ECN-capable. Otherwise its congestion markings would never be detected and congestion would just build up further. However, to support congestion marking below the IP layer or within tunnels, it is not sufficient to only check that the two layer 4 transport endpoints support ECN; correct operation also depends on the decapsulator at each subnet or tunnel egress faithfully propagating congestion notifications to the higher layer. Otherwise, a legacy decapsulator might silently fail to propagate any ECN signals from the outer to the forwarded header. Then the lost signals would never be detected and again congestion would build up further. The guidelines given later require protocol designers to carefully consider incremental deployment, and suggest various safe approaches for different circumstances.

Of course, the IETF does not have standards authority over every link layer protocol. So this document gives guidelines for designing propagation of congestion notification across the interface between IP and protocols that may encapsulate IP (i.e. that can be layered beneath IP). Each lower layer technology will exhibit different issues and compromises, so the IETF or the relevant standards body



must be free to define the specifics of each lower layer congestion notification scheme. Nonetheless, if the guidelines are followed, congestion notification should interwork between different technologies, using IP in its role as a 'portability layer'.

Therefore, the capitalized terms 'SHOULD' or 'SHOULD NOT' are often used in preference to 'MUST' or 'MUST NOT', because it is difficult to know the compromises that will be necessary in each protocol design. If a particular protocol design chooses not to follow a 'SHOULD (NOT)' given in the advice below, it MUST include a sound justification.

It has not been possible to give common guidelines for all lower layer technologies, because they do not all fit a common pattern. Instead they have been divided into a few distinct modes of operation: feed-forward-and-upward; feed-upward-and-forward; feed-backward; and null mode. These modes are described in Section 3, then in the subsequent sections separate guidelines are given for each mode.

#### 1.1. Update to RFC 3819

This document updates the brief advice to subnetwork designers about ECN in [RFC3819], by replacing the last two paragraphs of Section 13 with the following sentence:

By following the guidelines in [this document], subnetwork designers can enable a layer-2 protocol to participate in congestion control without dropping packets via propagation of explicit congestion notification (ECN [RFC3168]) to receivers.

and adding [this document] as an informative reference. {RFC Editor: Please replace both instances of [this document] above with the number of the present RFC when published.}

#### 1.2. Scope

This document only concerns wire protocol processing of explicit notification of congestion. It makes no changes or recommendations concerning algorithms for congestion marking or for congestion response, because algorithm issues should be independent of the layer the algorithm operates in.

The default ECN semantics are described in [RFC3168] and updated by [RFC8311]. Also the guidelines for AQM designers [RFC7567] clarify the semantics of both drop and ECN signals from AQM algorithms. [RFC4774] is the appropriate best current practice specification of how algorithms with alternative semantics for the ECN field can be

partitioned from Internet traffic that uses the default ECN semantics. There are two main examples for how alternative ECN semantics have been defined in practice:

- o RFC 4774 suggests using the ECN field in combination with a Diffserv codepoint such as in PCN [RFC6660], Voice over 3G [UTRAN] or Voice over LTE (VoLTE) [LTE-RA];
- o RFC 8311 suggests using the ECT(1) codepoint of the ECN field to indicate alternative semantics such as for the experimental Low Latency Low Loss Scalable throughput (L4S) service [I-D.ietf-tsvwg-ecn-l4s-id]).

The aim is that the default rules for encapsulating and decapsulating the ECN field are sufficiently generic that tunnels and subnets will encapsulate and decapsulate packets without regard to how algorithms elsewhere are setting or interpreting the semantics of the ECN field. [RFC6040] updates RFC 4774 to allow alternative encapsulation and decapsulation behaviours to be defined for alternative ECN semantics. However it reinforces the same point - that it is far preferable to try to fit within the common ECN encapsulation and decapsulation behaviours, because expecting all lower layer technologies and tunnels to be updated is likely to be completely impractical.

Alternative semantics for the ECN field can be defined to depend on the traffic class indicated by the DSCP. Therefore correct propagation of congestion signals could depend on correct propagation of the DSCP between the layers and along the path. For instance, if the meaning of the ECN field depends on the DSCP (as in PCN or VoLTE) and if the outer DSCP is stripped on decapsulation, as in the pipe model of [RFC2983], the special semantics of the ECN field would be lost. Similarly, if the DSCP is changed at the boundary between Diffserv domains, the special ECN semantics would also be lost. This is an important implication of the localized scope of most Diffserv arrangements. In this document, correct propagation of traffic class information is assumed, while what 'correct' means and how it is achieved is covered elsewhere (e.g. RFC 2983) and is outside the scope of the present document.

The guidelines in this document do ensure that common encapsulation and decapsulation rules are sufficiently generic to cover cases where ECT(1) is used instead of ECT(0) to identify alternative ECN semantics (as in L4S [I-D.ietf-tsvwg-ecn-l4s-id]) and where ECN marking algorithms use ECT(1) to encode 3 severity levels into the ECN field (e.g. PCN [RFC6660]) rather than the default of 2. All these different semantics for the ECN field work because it has been possible to define common default decapsulation rules that allow for all cases.

Note that the guidelines in this document do not necessarily require the subnet wire protocol to be changed to add support for congestion notification. For instance, the Feed-Up-and-Forward Mode (Section 3.2) and the Null Mode (Section 3.4) do not. Another way to add congestion notification without consuming header space in the subnet protocol might be to use a parallel control plane protocol.

This document focuses on the congestion notification interface between IP and lower layer or tunnel protocols that can encapsulate IP, where the term 'IP' includes v4 or v6, unicast, multicast or anycast. However, it is likely that the guidelines will also be useful when a lower layer protocol or tunnel encapsulates itself, e.g. Ethernet MAC in MAC ([IEEE802.1Q]; previously 802.1ah) or when it encapsulates other protocols. In the feed-backward mode, propagation of congestion signals for multicast and anycast packets is out-of-scope (because the complexity would make it unlikely to be attempted).

## 2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Further terminology used within this document:

Protocol data unit (PDU): Information that is delivered as a unit among peer entities of a layered network consisting of protocol control information (typically a header) and possibly user data (payload) of that layer. The scope of this document includes layer 2 and layer 3 networks, where the PDU is respectively termed a frame or a packet (or a cell in ATM). PDU is a general term for any of these. This definition also includes a payload with a shim header lying somewhere between layer 2 and 3.

Transport: The end-to-end transmission control function, conventionally considered at layer-4 in the OSI reference model. Given the audience for this document will often use the word transport to mean low level bit carriage, whenever the term is used it will be qualified, e.g. 'L4 transport'.

Encapsulator: The link or tunnel endpoint function that adds an outer header to a PDU (also termed the 'link ingress', the 'subnet ingress', the 'ingress tunnel endpoint' or just the 'ingress' where the context is clear).

**Decapsulator:** The link or tunnel endpoint function that removes an outer header from a PDU (also termed the 'link egress', the 'subnet egress', the 'egress tunnel endpoint' or just the 'egress' where the context is clear).

**Incoming header:** The header of an arriving PDU before encapsulation.

**Outer header:** The header added to encapsulate a PDU.

**Inner header:** The header encapsulated by the outer header.

**Outgoing header:** The header forwarded by the decapsulator.

**CE:** Congestion Experienced [RFC3168]

**ECT:** ECN-Capable (L4) Transport [RFC3168]

**Not-ECT:** Not ECN-Capable (L4) Transport [RFC3168]

**Load Regulator:** For each flow of PDUs, the transport function that is capable of controlling the data rate. Typically located at the data source, but in-path nodes can regulate load in some congestion control arrangements (e.g. admission control, policing nodes or transport circuit-breakers [RFC8084]). Note the term "a function capable of controlling the load" deliberately includes a transport that does not actually control the load responsively but ideally it ought to (e.g. a sending application without congestion control that uses UDP).

**ECN-PDU:** A PDU at the IP layer or below with a capacity to signal congestion that is part of a congestion control feedback loop within which all the nodes necessary to propagate the signal back to the Load Regulator are capable of doing that propagation. An IP packet with a non-zero ECN field implies that the endpoints are ECN-capable, so this would be an ECN-PDU. However, ECN-PDU is intended to be a general term for a PDU at lower layers, as well as at the IP layer.

**Not-ECN-PDU:** A PDU at the IP layer or below that is part of a congestion control feedback-loop within which at least one node necessary to propagate any explicit congestion notification signals back to the Load Regulator is not capable of doing that propagation.

### 3. Modes of Operation

This section sets down the different modes by which congestion information is passed between the lower layer and the higher one. It acts as a reference framework for the following sections, which give normative guidelines for designers of explicit congestion notification protocols, taking each mode in turn:

**Feed-Forward-and-Up:** Nodes feed forward congestion notification towards the egress within the lower layer then up and along the layers towards the end-to-end destination at the transport layer. The following local optimisation is possible:

**Feed-Up-and-Forward:** A lower layer switch feeds-up congestion notification directly into the higher layer (e.g. into the ECN field in the IP header), irrespective of whether the node is at the egress of a subnet.

**Feed-Backward:** Nodes feed back congestion signals towards the ingress of the lower layer and (optionally) attempt to control congestion within their own layer.

**Null:** Nodes cannot experience congestion at the lower layer except at ingress nodes (which are IP-aware or equivalently higher-layer-aware).

#### 3.1. Feed-Forward-and-Up Mode

Like IP and MPLS, many subnet technologies are based on self-contained protocol data units (PDUs) or frames sent unreliably. They provide no feedback channel at the subnetwork layer, instead relying on higher layers (e.g. TCP) to feed back loss signals.

In these cases, ECN may best be supported by standardising explicit notification of congestion into the lower layer protocol that carries the data forwards. Then a specification is needed for how the egress of the lower layer subnet propagates this explicit signal into the forwarded upper layer (IP) header. This signal continues forwards until it finally reaches the destination transport (at L4). Then typically the destination will feed this congestion notification back to the source transport using an end-to-end protocol (e.g. TCP). This is the arrangement that has already been used to add ECN to IP-in-IP tunnels [RFC6040], IP-in-MPLS and MPLS-in-MPLS [RFC5129].

This mode is illustrated in Figure 1. Along the middle of the figure, layers 2, 3 and 4 of the protocol stack are shown, and one packet is shown along the bottom as it progresses across the network from source to destination, crossing two subnets connected by a

router, and crossing two switches on the path across each subnet. Congestion at the output of the first switch (shown as \*) leads to a congestion marking in the L2 header (shown as C in the illustration of the packet). The chevrons show the progress of the resulting congestion indication. It is propagated from link to link across the subnet in the L2 header, then when the router removes the marked L2 header, it propagates the marking up into the L3 (IP) header. The router forwards the marked L3 header into subnet 2, and when it adds a new L2 header it copies the L3 marking into the L2 header as well, as shown by the 'C's in both layers (assuming the technology of subnet 2 also supports explicit congestion marking).

Note that there is no implication that each 'C' marking is encoded the same; a different encoding might be used for the 'C' marking in each protocol.

Finally, for completeness, we show the L3 marking arriving at the destination, where the host transport protocol (e.g. TCP) feeds it back to the source in the L4 acknowledgement (the 'C' at L4 in the packet at the top of the diagram).

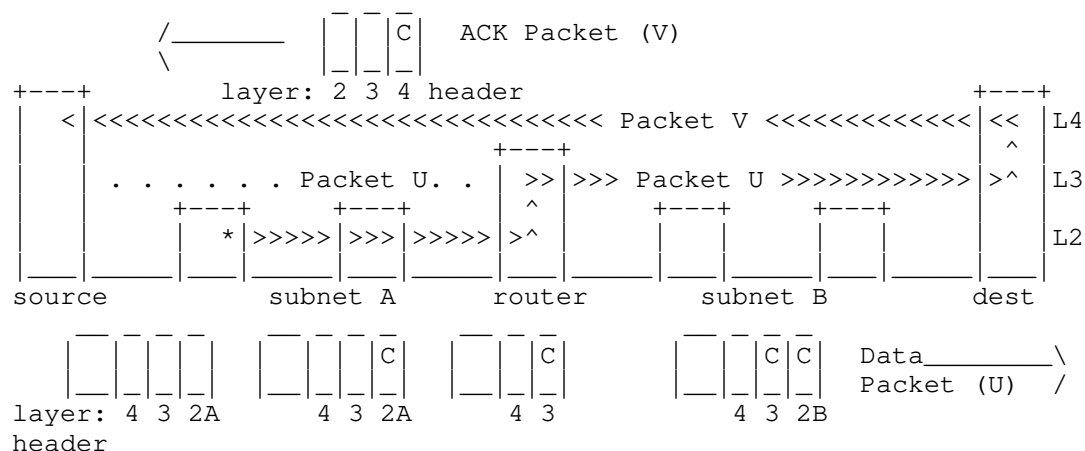


Figure 1: Feed-Forward-and-Up Mode

Of course, modern networks are rarely as simple as this text-book example, often involving multiple nested layers. For example, a 3GPP mobile network may have two IP-in-IP (GTP [GTPv1]) tunnels in series and an MPLS backhaul between the base station and the first router. Nonetheless, the example illustrates the general idea of feeding congestion notification forward then upward whenever a header is removed at the egress of a subnet.



support ECN natively, so when the router adds the layer-2 header it copies the ECN marking from L3 to L2 as well.

### 3.3. Feed-Backward Mode

In some layer 2 technologies, explicit congestion notification has been defined for use internally within the subnet with its own feedback and load regulation, but typically the interface with IP for ECN has not been defined.

For instance, for the available bit-rate (ABR) service in ATM, the relative rate mechanism was one of the more popular mechanisms for managing traffic, tending to supersede earlier designs. In this approach ATM switches send special resource management (RM) cells in both the forward and backward directions to control the ingress rate of user data into a virtual circuit. If a switch buffer is approaching congestion or is congested it sends an RM cell back towards the ingress with respectively the No Increase (NI) or Congestion Indication (CI) bit set in its message type field [ATM-TM-ABR]. The ingress then holds or decreases its sending bit-rate accordingly.



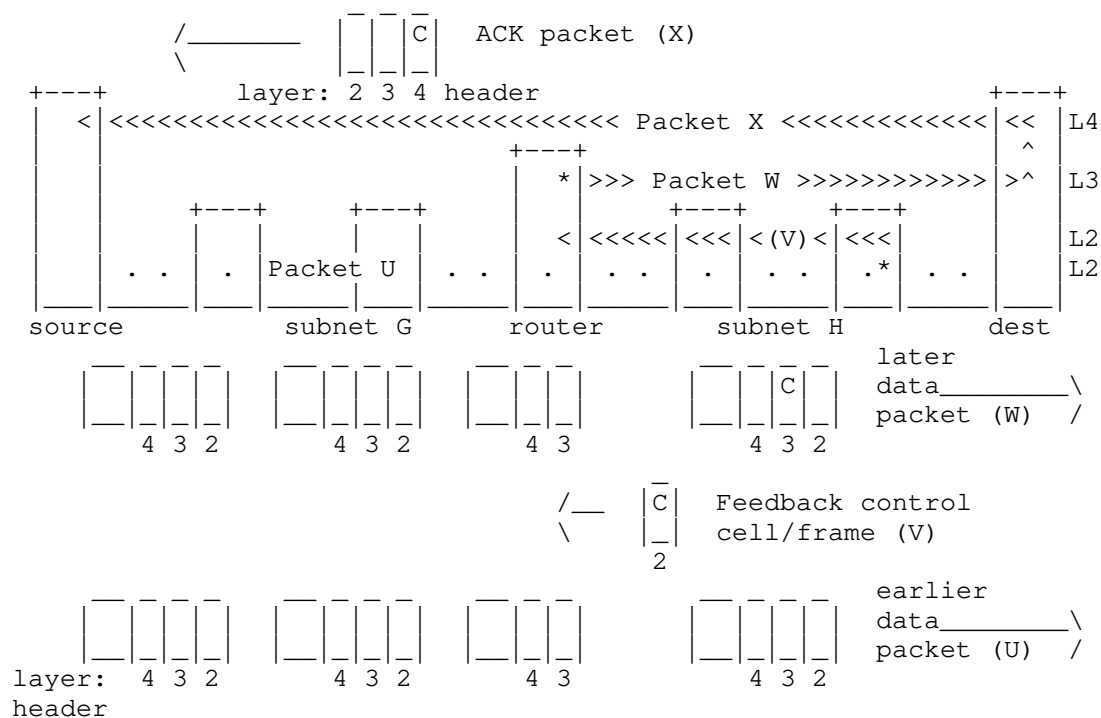


Figure 3: Feed-Backward Mode

ATM's feed-backward approach does not fit well when layered beneath IP's feed-forward approach--unless the initial data source is the same node as the ATM ingress. Figure 3 shows the feed-backward approach being used in subnet H. If the final switch on the path is congested (\*), it does not feed-forward any congestion indications on packet (U). Instead it sends a control cell (V) back to the router at the ATM ingress.

However, the backward feedback does not reach the original data source directly because IP does not support backward feedback (and subnet G is independent of subnet H). Instead, the router in the middle throttles down its sending rate but the original data sources don't reduce their rates. The resulting rate mismatch causes the middle router's buffer at layer 3 to back up until it becomes congested, which it signals forwards on later data packets at layer 3 (e.g. packet W). Note that the forward signal from the middle router is not triggered directly by the backward signal. Rather, it is triggered by congestion resulting from the middle router's mismatched rate response to the backward signal.

In response to this later forward signalling, end-to-end feedback at layer-4 finally completes the tortuous path of congestion indications back to the origin data source, as before.

Quantized congestion notification (QCN [IEEE802.1Q]) would suffer from similar problems if extended to multiple subnets. However, from the start QCN was clearly characterized as solely applicable to a single subnet (see Section 6).

### 3.4. Null Mode

Often link and physical layer resources are 'non-blocking' by design. In these cases congestion notification may be implemented but it does not need to be deployed at the lower layer; ECN in IP would be sufficient.

A degenerate example is a point-to-point Ethernet link. Excess loading of the link merely causes the queue from the higher layer to back up, while the lower layer remains immune to congestion. Even a whole meshed subnetwork can be made immune to interior congestion by limiting ingress capacity and sufficient sizing of interior links, e.g. a non-blocking fat-tree network [Leiserson85]. An alternative to fat links near the root is numerous thin links with multi-path routing to ensure even worst-case patterns of load cannot congest any link, e.g. a Clos network [Clos53].

## 4. Feed-Forward-and-Up Mode: Guidelines for Adding Congestion Notification

Feed-forward-and-up is the mode already used for signalling ECN up the layers through MPLS into IP [RFC5129] and through IP-in-IP tunnels [RFC6040], whether encapsulating with IPv4 [RFC2003], IPv6 [RFC2473] or IPsec [RFC4301]. These RFCs take a consistent approach and the following guidelines are designed to ensure this consistency continues as ECN support is added to other protocols that encapsulate IP. The guidelines are also designed to ensure compliance with the more general best current practice for the design of alternate ECN schemes given in [RFC4774] and extended by [RFC8311].

The rest of this section is structured as follows:

- o Section 4.1 addresses the most straightforward cases, where [RFC6040] can be applied directly to add ECN to tunnels that are effectively IP-in-IP tunnels, but with shim header(s) between the IP headers.
- o The subsequent sections give guidelines for adding ECN to a subnet technology that uses feed-forward-and-up mode like IP, but it is

not so similar to IP that [RFC6040] rules can be applied directly. Specifically:

- \* Sections 4.2, 4.3 and 4.4 respectively address how to add ECN support to the wire protocol and to the encapsulators and decapsulators at the ingress and egress of the subnet.
- \* Section 4.5 deals with the special, but common, case of sequences of tunnels or subnets that all use the same technology
- \* Section 4.6 deals with the question of reframing when IP packets do not map 1:1 into lower layer frames.

#### 4.1. IP-in-IP Tunnels with Shim Headers

A common pattern for many tunnelling protocols is to encapsulate an inner IP header with shim header(s) then an outer IP header. A shim header is defined as one that is not sufficient alone to forward the packet as an outer header. Another common pattern is for a shim to encapsulate a layer 2 (L2) header, which in turn encapsulates (or might encapsulate) an IP header. [I-D.ietf-tsvwg-rfc6040update-shim] clarifies that RFC 6040 is just as applicable when there are shim(s) and possibly a L2 header between two IP headers.

However, it is not always feasible or necessary to propagate ECN between IP headers when separated by a shim. For instance, it might be too costly to dig to arbitrary depths to find an inner IP header, there may be little or no congestion within the tunnel by design (see null mode in Section 3.4 above), or a legacy implementation might not support ECN. In cases where a tunnel does not support ECN, it is important that the ingress does not copy the ECN field from an inner IP header to an outer. Therefore section 4 of [I-D.ietf-tsvwg-rfc6040update-shim] requires network operators to configure the ingress of a tunnel that does not support ECN so that it zeros the ECN field in the outer IP header.

Nonetheless, in many cases it is feasible to propagate the ECN field between IP headers separated by shim header(s) and/or a L2 header. Particularly in the typical case when the outer IP header and the shim(s) are added (or removed) as part of the same procedure. Even if the shim(s) encapsulate a L2 header, it is often possible to find an inner IP header within the L2 PDU and propagate ECN between that and the outer IP header. This can be thought of as a special case of the feed-up-and-forward mode (Section 3.2), so the guidelines for this mode apply (Section 5).

Numerous shim protocols have been defined for IP tunnelling. More recent ones e.g. Geneve [RFC8926] and Generic UDP Encapsulation (GUE) [I-D.ietf-intarea-gue] cite and follow RFC 6040. And some earlier ones, e.g. CAPWAP [RFC5415] and LISP [RFC6830], cite RFC 3168, which is compatible with RFC 6040.

However, as Section 9.3 of RFC 3168 pointed out, ECN support needs to be defined for many earlier shim-based tunnelling protocols, e.g. L2TPv2 [RFC2661], L2TPv3 [RFC3931], GRE [RFC2784], PPTP [RFC2637], GTP [GTPv1], [GTPv1-U], [GTPv2-C] and Teredo [RFC4380] as well as some recent ones, e.g. VXLAN [RFC7348], NVGRE [RFC7637] and NSH [RFC8300].

All these IP-based encapsulations can be updated in one shot by simple reference to RFC 6040. However, it would not be appropriate to update all these protocols from within the present guidance document. Instead a companion specification [I-D.ietf-tsvwg-rfc6040update-shim] has been prepared that has the appropriate standards track status to update standards track protocols. For those that are not under IETF change control [I-D.ietf-tsvwg-rfc6040update-shim] can only recommend that the relevant body updates them.

#### 4.2. Wire Protocol Design: Indication of ECN Support

This section is intended to guide the redesign of any lower layer protocol that encapsulate IP to add native ECN support at the lower layer. It reflects the approaches used in [RFC6040] and in [RFC5129]. Therefore IP-in-IP tunnels or IP-in-MPLS or MPLS-in-MPLS encapsulations that already comply with [RFC6040] or [RFC5129] will already satisfy this guidance.

A lower layer (or subnet) congestion notification system:

1. SHOULD NOT apply explicit congestion notifications to PDUs that are destined for legacy layer-4 transport implementations that will not understand ECN, and
2. SHOULD NOT apply explicit congestion notifications to PDUs if the egress of the subnet might not propagate congestion notifications onward into the higher layer.

We use the term ECN-PDUs for a PDU on a feedback loop that will propagate congestion notification properly because it meets both the above criteria. And a Not-ECN-PDU is a PDU on a feedback loop that does not meet at least one of the criteria, and will therefore not propagate congestion notification properly. A

corollary of the above is that a lower layer congestion notification protocol:

3. SHOULD be able to distinguish ECN-PDUs from Not-ECN-PDUs.

Note that there is no need for all interior nodes within a subnet to be able to mark congestion explicitly. A mix of ECN and drop signals from different nodes is fine. However, if any interior nodes might generate ECN markings, guideline 2 above says that all relevant egress node(s) SHOULD be able to propagate those markings up to the higher layer.

In IP, if the ECN field in each PDU is cleared to the Not-ECT (not ECN-capable transport) codepoint, it indicates that the L4 transport will not understand congestion markings. A congested buffer must not mark these Not-ECT PDUs, and therefore drops them instead.

The mechanism a lower layer uses to distinguish the ECN-capability of PDUs need not mimic that of IP. The above guidelines merely say that the lower layer system, as a whole, should achieve the same outcome. For instance, ECN-capable feedback loops might use PDUs that are identified by a particular set of labels or tags. Alternatively, logical link protocols that use flow state might determine whether a PDU can be congestion marked by checking for ECN-support in the flow state. Other protocols might depend on out-of-band control signals.

The per-domain checking of ECN support in MPLS [RFC5129] is a good example of a way to avoid sending congestion markings to L4 transports that will not understand them, without using any header space in the subnet protocol.

In MPLS, header space is extremely limited, therefore RFC5129 does not provide a field in the MPLS header to indicate whether the PDU is an ECN-PDU or a Not-ECN-PDU. Instead, interior nodes in a domain are allowed to set explicit congestion indications without checking whether the PDU is destined for a L4 transport that will understand them. Nonetheless, this is made safe by requiring that the network operator upgrades all decapsulating edges of a whole domain at once, as soon as even one switch within the domain is configured to mark rather than drop during congestion. Therefore, any edge node that might decapsulate a packet will be capable of checking whether the higher layer transport is ECN-capable. When decapsulating a CE-marked packet, if the decapsulator discovers that the higher layer (inner header) indicates the transport is not ECN-capable, it drops the packet--effectively on behalf of the earlier congested node (see Decapsulation Guideline 1 in Section 4.4).

It was only appropriate to define such an incremental deployment strategy because MPLS is targeted solely at professional operators, who can be expected to ensure that a whole subnetwork is consistently configured. This strategy might not be appropriate for other link technologies targeted at zero-configuration deployment or deployment by the general public (e.g. Ethernet). For such 'plug-and-play' environments it will be necessary to invent a failsafe approach that ensures congestion markings will never fall into black holes, no matter how inconsistently a system is put together. Alternatively, congestion notification relying on correct system configuration could be confined to flavours of Ethernet intended only for professional network operators, such as Provider Backbone Bridges (PBB [IEEE802.1Q]; previously 802.1ah).

ECN support in TRILL [I-D.ietf-trill-ecn-support] provides a good example of how to add ECN to a lower layer protocol without relying on careful and consistent operator configuration. TRILL provides an extension header word with space for flags of different categories depending on whether logic to understand the extension is critical. The congestion experienced marking has been defined as a 'critical ingress-to-egress' flag. So if a transit RBridge sets this flag and an egress RBridge does not have any logic to process it, it will drop it; which is the desired default action anyway. Therefore TRILL R Bridges can be updated with support for ECN in no particular order and, at the egress of the TRILL campus, congestion notification will be propagated to IP as ECN whenever ECN logic has been implemented, or as drop otherwise.

QCN [IEEE802.1Q] is not intended to extend beyond a single subnet, or to interoperate with ECN. Nonetheless, the way QCN indicates to lower layer devices that the end-points will not understand QCN provides another example that a lower layer protocol designer might be able to mimic for their scenario. An operator can define certain Priority Code Points (PCPs [IEEE802.1Q]; previously 802.1p) to indicate non-QCN frames and an ingress bridge is required to map arriving not-QCN-capable IP packets to one of these non-QCN PCPs.

#### 4.3. Encapsulation Guidelines

This section is intended to guide the redesign of any node that encapsulates IP with a lower layer header when adding native ECN support to the lower layer protocol. It reflects the approaches used in [RFC6040] and in [RFC5129]. Therefore IP-in-IP tunnels or IP-in-MPLS or MPLS-in-MPLS encapsulations that already comply with [RFC6040] or [RFC5129] will already satisfy this guidance.

1. Egress Capability Check: A subnet ingress needs to be sure that the corresponding egress of a subnet will propagate any

congestion notification added to the outer header across the subnet. This is necessary in addition to checking that an incoming PDU indicates an ECN-capable (L4) transport. Examples of how this guarantee might be provided include:

- \* by configuration (e.g. if any label switches in a domain support ECN marking, [RFC5129] requires all egress nodes to have been configured to propagate ECN)
  - \* by the ingress explicitly checking that the egress propagates ECN (e.g. an early attempt to add ECN support to TRILL used IS-IS to check path capabilities before adding ECN extension flags to each frame [RFC7780]).
  - \* by inherent design of the protocol (e.g. by encoding ECN marking on the outer header in such a way that a legacy egress that does not understand ECN will consider the PDU corrupt or invalid and discard it, thus at least propagating a form of congestion signal).
2. Egress Fails Capability Check: If the ingress cannot guarantee that the egress will propagate congestion notification, the ingress SHOULD disable ECN at the lower layer when it forwards the PDU. An example of how the ingress might disable ECN at the lower layer would be by setting the outer header of the PDU to identify it as a Not-ECN-PDU, assuming the subnet technology supports such a concept.
  3. Standard Congestion Monitoring Baseline: Once the ingress to a subnet has established that the egress will correctly propagate ECN, on encapsulation it SHOULD encode the same level of congestion in outer headers as is arriving in incoming headers. For example it might copy any incoming congestion notification into the outer header of the lower layer protocol.

This ensures that bulk congestion monitoring of outer headers (e.g. by a network management node monitoring ECN in passing frames) will measure congestion accumulated along the whole upstream path - since the Load Regulator not just since the ingress of the subnet. A node that is not the Load Regulator SHOULD NOT re-initialize the level of CE markings in the outer to zero.

It would still also be possible to measure congestion introduced across one subnet (or tunnel) by subtracting the level of CE markings on inner headers from that on outer headers (see Appendix C of [RFC6040]). For example:

- \* If this guideline has been followed and if the level of CE markings is 0.4% on the outer and 0.1% on the inner, 0.4% congestion has been introduced across all the networks since the load regulator, and 0.3% ( $= 0.4\% - 0.1\%$ ) has been introduced since the ingress to the current subnet (or tunnel);
- \* Without this guideline, if the subnet ingress had re-initialized the outer congestion level to zero, the outer and inner would measure 0.1% and 0.3%. It would still be possible to infer that the congestion introduced since the Load Regulator was 0.4% ( $= 0.1\% + 0.3\%$ ). But only if the monitoring system somehow knows whether the subnet ingress re-initialized the congestion level.

As long as subnet and tunnel technologies use the standard congestion monitoring baseline in this guideline, monitoring systems will know to use the former approach, rather than having to "somehow know" which approach to use.

#### 4.4. Decapsulation Guidelines

This section is intended to guide the redesign of any node that decapsulates IP from within a lower layer header when adding native ECN support to the lower layer protocol. It reflects the approaches used in [RFC6040] and in [RFC5129]. Therefore IP-in-IP tunnels or IP-in-MPLS or MPLS-in-MPLS encapsulations that already comply with [RFC6040] or [RFC5129] will already satisfy this guidance.

A subnet egress SHOULD NOT simply copy congestion notification from outer headers to the forwarded header. It SHOULD calculate the outgoing congestion notification field from the inner and outer headers using the following guidelines. If there is any conflict, rules earlier in the list take precedence over rules later in the list:

1. If the arriving inner header is a Not-ECN-PDU it implies the L4 transport will not understand explicit congestion markings.  
Then:
  - \* If the outer header carries an explicit congestion marking, drop is the only indication of congestion that the L4 transport will understand. If the congestion marking is the most severe possible, the packet MUST be dropped. However, if congestion can be marked with multiple levels of severity and the packet's marking is not the most severe, this requirement can be relaxed to: the packet SHOULD be dropped.



- \* If the outer is an ECN-PDU that carries no indication of congestion or a Not-ECN-PDU the PDU SHOULD be forwarded, but still as a Not-ECN-PDU.
- 2. If the outer header does not support explicit congestion notification (a Not-ECN-PDU), but the inner header does (an ECN-PDU), the inner header SHOULD be forwarded unchanged.
- 3. In some lower layer protocols congestion may be signalled as a numerical level, such as in the control frames of quantized congestion notification (QCN [IEEE802.1Q]). If such a multi-bit encoding encapsulates an ECN-capable IP data packet, a function will be needed to convert the quantized congestion level into the frequency of congestion markings in outgoing IP packets.
- 4. Congestion indications might be encoded by a severity level. For instance increasing levels of congestion might be encoded by numerically increasing indications, e.g. pre-congestion notification (PCN) can be encoded in each PDU at three severity levels in IP or MPLS [RFC6660] and the default encapsulation and decapsulation rules [RFC6040] are compatible with this interpretation of the ECN field.

If the arriving inner header is an ECN-PDU, where the inner and outer headers carry indications of congestion of different severity, the more severe indication SHOULD be forwarded in preference to the less severe.

- 5. The inner and outer headers might carry a combination of congestion notification fields that should not be possible given any currently used protocol transitions. For instance, if Encapsulation Guideline 3 in Section 4.3 had been followed, it should not be possible to have a less severe indication of congestion in the outer than in the inner. It MAY be appropriate to log unexpected combinations of headers and possibly raise an alarm.

If a safe outgoing codepoint can be defined for such a PDU, the PDU SHOULD be forwarded rather than dropped. Some implementers discard PDUs with currently unused combinations of headers just in case they represent an attack. However, an approach using alarms and policy-mediated drop is preferable to hard-coded drop, so that operators can keep track of possible attacks but currently unused combinations are not precluded from future use through new standards actions.

#### 4.5. Sequences of Similar Tunnels or Subnets

In some deployments, particularly in 3GPP networks, an IP packet may traverse two or more IP-in-IP tunnels in sequence that all use identical technology (e.g. GTP).

In such cases, it would be sufficient for every encapsulation and decapsulation in the chain to comply with RFC 6040. Alternatively, as an optimisation, a node that decapsulates a packet and immediately re-encapsulates it for the next tunnel MAY copy the incoming outer ECN field directly to the outgoing outer and the incoming inner ECN field directly to the outgoing inner. Then the overall behavior across the sequence of tunnel segments would still be consistent with RFC 6040.

Appendix C of RFC6040 describes how a tunnel egress can monitor how much congestion has been introduced within a tunnel. A network operator might want to monitor how much congestion had been introduced within a whole sequence of tunnels. Using the technique in Appendix C of RFC6040 at the final egress, the operator could monitor the whole sequence of tunnels, but only if the above optimisation were used consistently along the sequence of tunnels, in order to make it appear as a single tunnel. Therefore, tunnel endpoint implementations SHOULD allow the operator to configure whether this optimisation is enabled.

When ECN support is added to a subnet technology, consideration SHOULD be given to a similar optimisation between subnets in sequence if they all use the same technology.

#### 4.6. Reframing and Congestion Markings

The guidance in this section is worded in terms of framing boundaries, but it applies equally whether the protocol data units are frames, cells or packets.

Where an AQM marks the ECN field of IP packets as they queue into a layer-2 link, there will be no problem with framing boundaries, because the ECN markings would be applied directly to IP packets. The guidance in this section is only applicable where an ECN capability is being added to a layer-2 protocol so that layer-2 frames can be ECN-marked by an AQM at layer-2. This would only be necessary where AQM will be applied at pure layer-2 nodes (without IP-awareness).

When layer-2 frame headers are stripped off and IP PDUs with different boundaries are forwarded, the provisions in RFC7141 for handling congestion indications when splitting or merging packets

apply (see Section 2.4 of [RFC7141]). Those provisions include: "The general rule to follow is that the number of octets in packets with congestion indications SHOULD be equivalent before and after merging or splitting." See RFC 7141 for the complete provisions and related discussion, including an exception to that general rule.

As also recommended in RFC 7141, the mechanism for propagating congestion indications SHOULD ensure that any new incoming congestion indication is propagated immediately, and not held awaiting possible arrival of further congestion indications sufficient to indicate congestion for all of the octets of an outgoing IP PDU.

#### 5. Feed-Up-and-Forward Mode: Guidelines for Adding Congestion Notification

The guidance in this section is applicable, for example, when IP packets:

- o are encapsulated in Ethernet headers, which have no support for ECN;
- o are forwarded by the eNode-B (base station) of a 3GPP radio access network, which is required to apply ECN marking during congestion, [LTE-RA], [UTRAN], but the Packet Data Convergence Protocol (PDCP) that encapsulates the IP header over the radio access has no support for ECN.

This guidance also generalizes to encapsulation by other subnet technologies with no native support for explicit congestion notification at the lower layer, but with support for finding and processing an IP header. It is unlikely to be applicable or necessary for IP-in-IP encapsulation, where feed-forward-and-up mode based on [RFC6040] would be more appropriate.

Marking the IP header while switching at layer-2 (by using a layer-3 switch) or while forwarding in a radio access network seems to represent a layering violation. However, it can be considered as a benign optimisation if the guidelines below are followed. Feed-up-and-forward is certainly not a general alternative to implementing feed-forward congestion notification in the lower layer, because:

- o IPv4 and IPv6 are not the only layer-3 protocols that might be encapsulated by lower layer protocols
- o Link-layer encryption might be in use, making the layer-2 payload inaccessible

- o Many Ethernet switches do not have 'layer-3 switch' capabilities so they cannot read or modify an IP payload
- o It might be costly to find an IP header (v4 or v6) when it may be encapsulated by more than one lower layer header, e.g. Ethernet MAC in MAC ([IEEE802.1Q]; previously 802.1ah).

Nonetheless, configuring lower layer equipment to look for an ECN field in an encapsulated IP header is a useful optimisation. If the implementation follows the guidelines below, this optimisation does not have to be confined to a controlled environment such as within a data centre; it could usefully be applied on any network--even if the operator is not sure whether the above issues will never apply:

1. If a native lower-layer congestion notification mechanism exists for a subnet technology, it is safe to mix feed-up-and-forward with feed-forward-and-up on other switches in the same subnet. However, it will generally be more efficient to use the native mechanism.
  2. The depth of the search for an IP header SHOULD be limited. If an IP header is not found soon enough, or an unrecognized or unreadable header is encountered, the switch SHOULD resort to an alternative means of signalling congestion (e.g. drop, or the native lower layer mechanism if available).
  3. It is sufficient to use the first IP header found in the stack; the egress of the relevant tunnel can propagate congestion notification upwards to any more deeply encapsulated IP headers later.
6. Feed-Backward Mode: Guidelines for Adding Congestion Notification

It can be seen from Section 3.3 that congestion notification in a subnet using feed-backward mode has generally not been designed to be directly coupled with IP layer congestion notification. The subnet attempts to minimize congestion internally, and if the incoming load at the ingress exceeds the capacity somewhere through the subnet, the layer 3 buffer into the ingress backs up. Thus, a feed-backward mode subnet is in some sense similar to a null mode subnet, in that there is no need for any direct interaction between the subnet and higher layer congestion notification. Therefore no detailed protocol design guidelines are appropriate. Nonetheless, a more general guideline is appropriate:

A subnetwork technology intended to eventually interface to IP SHOULD NOT be designed using only the feed-backward mode, which is certainly best for a stand-alone subnet, but would need to be

modified to work efficiently as part of the wider Internet, because IP uses feed-forward-and-up mode.

The feed-backward approach at least works beneath IP, where the term 'works' is used only in a narrow functional sense because feed-backward can result in very inefficient and sluggish congestion control--except if it is confined to the subnet directly connected to the original data source, when it is faster than feed-forward. It would be valid to design a protocol that could work in feed-backward mode for paths that only cross one subnet, and in feed-forward-and-up mode for paths that cross subnets.

In the early days of TCP/IP, a similar feed-backward approach was tried for explicit congestion signalling, using source-quench (SQ) ICMP control packets. However, SQ fell out of favour and is now formally deprecated [RFC6633]. The main problem was that it is hard for a data source to tell the difference between a spoofed SQ message and a quench request from a genuine buffer on the path. It is also hard for a lower layer buffer to address an SQ message to the original source port number, which may be buried within many layers of headers, and possibly encrypted.

QCN (also known as backward congestion notification, BCN; see Sections 30--33 of [IEEE802.1Q]; previously known as 802.1Qau) uses a feed-backward mode structurally similar to ATM's relative rate mechanism. However, QCN confines its applicability to scenarios such as some data centres where all endpoints are directly attached by the same Ethernet technology. If a QCN subnet were later connected into a wider IP-based internetwork (e.g. when attempting to interconnect multiple data centres) it would suffer the inefficiency shown in Figure 3.

## 7. IANA Considerations

This memo includes no request to IANA.

## 8. Security Considerations

If a lower layer wire protocol is redesigned to include explicit congestion signalling in-band in the protocol header, care SHOULD be taken to ensure that the field used is specified as mutable during transit. Otherwise interior nodes signalling congestion would invalidate any authentication protocol applied to the lower layer header--by altering a header field that had been assumed as immutable.

The redesign of protocols that encapsulate IP in order to propagate congestion signals between layers raises potential signal integrity

concerns. Experimental or proposed approaches exist for assuring the end-to-end integrity of in-band congestion signals, e.g.:

- o Congestion exposure (ConEx ) for networks to audit that their congestion signals are not being suppressed by other networks or by receivers, and for networks to police that senders are responding sufficiently to the signals, irrespective of the L4 transport protocol used [RFC7713].
- o A test for a sender to detect whether a network or the receiver is suppressing congestion signals (for example see 2nd para of Section 20.2 of [RFC3168]).

Given these end-to-end approaches are already being specified, it would make little sense to attempt to design hop-by-hop congestion signal integrity into a new lower layer protocol, because end-to-end integrity inherently achieves hop-by-hop integrity.

Section 6 gives vulnerability to spoofing as one of the reasons for deprecating feed-backward mode.

## 9. Conclusions

Following the guidance in this document enables ECN support to be extended to numerous protocols that encapsulate IP (v4 & v6) in a consistent way, so that IP continues to fulfil its role as an end-to-end interoperability layer. This includes:

- o A wide range of tunnelling protocols including those with various forms of shim header between two IP headers, possibly also separated by a L2 header;
- o A wide range of subnet technologies, particularly those that work in the same 'feed-forward-and-up' mode that is used to support ECN in IP and MPLS.

Guidelines have been defined for supporting propagation of ECN between Ethernet and IP on so-called Layer-3 Ethernet switches, using a 'feed-up-and-forward' mode. This approach could enable other subnet technologies to pass ECN signals into the IP layer, even if they do not support ECN natively.

Finally, attempting to add ECN to a subnet technology in feed-backward mode is deprecated except in special cases, due to its likely sluggish response to congestion.

## 10. Acknowledgements

Thanks to Gorry Fairhurst and David Black for extensive reviews. Thanks also to the following reviewers: Joe Touch, Andrew McGregor, Richard Scheffenegger, Ingemar Johansson, Piers O'Hanlon, Donald Eastlake, Jonathan Morton and Michael Welzl, who pointed out that lower layer congestion notification signals may have different semantics to those in IP. Thanks are also due to the tsvwg chairs, TSV ADs and IETF liaison people such as Eric Gray, Dan Romascanu and Gonzalo Camarillo for helping with the liaisons with the IEEE and 3GPP. And thanks to Georg Mayer and particularly to Erik Guttman for the extensive search and categorisation of any 3GPP specifications that cite ECN specifications.

Bob Briscoe was part-funded by the European Community under its Seventh Framework Programme through the Trilogy project (ICT-216372) for initial drafts and through the Reducing Internet Transport Latency (RITE) project (ICT-317700) subsequently. The views expressed here are solely those of the authors.

## 11. Contributors

Pat Thaler  
Broadcom Corporation (retired)  
CA  
USA

Pat was a co-author of this draft, but retired before its publication.

## 12. Comments Solicited

Comments and questions are encouraged and very welcome. They can be addressed to the IETF Transport Area working group mailing list <tsvwg@ietf.org>, and/or to the authors.

## 13. References

### 13.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC3819] Karn, P., Ed., Bormann, C., Fairhurst, G., Grossman, D., Ludwig, R., Mahdavi, J., Montenegro, G., Touch, J., and L. Wood, "Advice for Internet Subnetwork Designers", BCP 89, RFC 3819, DOI 10.17487/RFC3819, July 2004, <<https://www.rfc-editor.org/info/rfc3819>>.
- [RFC4774] Floyd, S., "Specifying Alternate Semantics for the Explicit Congestion Notification (ECN) Field", BCP 124, RFC 4774, DOI 10.17487/RFC4774, November 2006, <<https://www.rfc-editor.org/info/rfc4774>>.
- [RFC5129] Davie, B., Briscoe, B., and J. Tay, "Explicit Congestion Marking in MPLS", RFC 5129, DOI 10.17487/RFC5129, January 2008, <<https://www.rfc-editor.org/info/rfc5129>>.
- [RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <<https://www.rfc-editor.org/info/rfc6040>>.
- [RFC7141] Briscoe, B. and J. Manner, "Byte and Packet Congestion Notification", BCP 41, RFC 7141, DOI 10.17487/RFC7141, February 2014, <<https://www.rfc-editor.org/info/rfc7141>>.

### 13.2. Informative References

- [ATM-TM-ABR] Cisco, "Understanding the Available Bit Rate (ABR) Service Category for ATM VCs", Design Technote 10415, June 2005.
- [Buck00] Buckwalter, J., "Frame Relay: Technology and Practice", Pub. Addison Wesley ISBN-13: 978-0201485240, 2000.
- [Clos53] Clos, C., "A Study of Non-Blocking Switching Networks", Bell Systems Technical Journal 32(2):406--424, March 1953.
- [GTPv1] 3GPP, "GPRS Tunnelling Protocol (GTP) across the Gn and Gp interface", Technical Specification TS 29.060.
- [GTPv1-U] 3GPP, "General Packet Radio System (GPRS) Tunnelling Protocol User Plane (GTPv1-U)", Technical Specification TS 29.281.



- [GTPv2-C] 3GPP, "Evolved General Packet Radio Service (GPRS) Tunnelling Protocol for Control plane (GTPv2-C)", Technical Specification TS 29.274.
- [I-D.ietf-intarea-gue]  
Herbert, T., Yong, L., and O. Zia, "Generic UDP Encapsulation", draft-ietf-intarea-gue-09 (work in progress), October 2019.
- [I-D.ietf-trill-ecn-support]  
Eastlake, D. E. and B. Briscoe, "TRILL (Transparent Interconnection of Lots of Links): ECN (Explicit Congestion Notification) Support", draft-ietf-trill-ecn-support-07 (work in progress), February 2018.
- [I-D.ietf-tsvwg-ecn-l4s-id]  
Schepper, K. D. and B. Briscoe, "Explicit Congestion Notification (ECN) Protocol for Ultra-Low Queuing Delay (L4S)", draft-ietf-tsvwg-ecn-l4s-id-14 (work in progress), March 2021.
- [I-D.ietf-tsvwg-rfc6040update-shim]  
Briscoe, B., "Propagating Explicit Congestion Notification Across IP Tunnel Headers Separated by a Shim", draft-ietf-tsvwg-rfc6040update-shim-13 (work in progress), March 2021.
- [IEEE802.1Q]  
IEEE, "IEEE Standard for Local and Metropolitan Area Networks--Virtual Bridged Local Area Networks--Amendment 6: Provider Backbone Bridges", IEEE Std 802.1Q-2018, July 2018, <<https://ieeexplore.ieee.org/document/8403927>>.
- [ITU-T.I.371]  
ITU-T, "Traffic Control and Congestion Control in B-ISDN", ITU-T Rec. I.371 (03/04), March 2004, <<http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5454061>>.
- [Leiserson85]  
Leiserson, C., "Fat-trees: universal networks for hardware-efficient supercomputing", IEEE Transactions on Computers 34(10):892-901, October 1985.
- [LTE-RA] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2", Technical Specification TS 36.300.

- [RFC2003] Perkins, C., "IP Encapsulation within IP", RFC 2003, DOI 10.17487/RFC2003, October 1996, <<https://www.rfc-editor.org/info/rfc2003>>.
- [RFC2473] Conta, A. and S. Deering, "Generic Packet Tunneling in IPv6 Specification", RFC 2473, DOI 10.17487/RFC2473, December 1998, <<https://www.rfc-editor.org/info/rfc2473>>.
- [RFC2637] Hamzeh, K., Pall, G., Verthein, W., Taarud, J., Little, W., and G. Zorn, "Point-to-Point Tunneling Protocol (PPTP)", RFC 2637, DOI 10.17487/RFC2637, July 1999, <<https://www.rfc-editor.org/info/rfc2637>>.
- [RFC2661] Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G., and B. Palter, "Layer Two Tunneling Protocol "L2TP"", RFC 2661, DOI 10.17487/RFC2661, August 1999, <<https://www.rfc-editor.org/info/rfc2661>>.
- [RFC2784] Farinacci, D., Li, T., Hanks, S., Meyer, D., and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 2784, DOI 10.17487/RFC2784, March 2000, <<https://www.rfc-editor.org/info/rfc2784>>.
- [RFC2884] Hadi Salim, J. and U. Ahmed, "Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks", RFC 2884, DOI 10.17487/RFC2884, July 2000, <<https://www.rfc-editor.org/info/rfc2884>>.
- [RFC2983] Black, D., "Differentiated Services and Tunnels", RFC 2983, DOI 10.17487/RFC2983, October 2000, <<https://www.rfc-editor.org/info/rfc2983>>.
- [RFC3931] Lau, J., Ed., Townsley, M., Ed., and I. Goyret, Ed., "Layer Two Tunneling Protocol - Version 3 (L2TPv3)", RFC 3931, DOI 10.17487/RFC3931, March 2005, <<https://www.rfc-editor.org/info/rfc3931>>.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<https://www.rfc-editor.org/info/rfc4301>>.
- [RFC4380] Huitema, C., "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)", RFC 4380, DOI 10.17487/RFC4380, February 2006, <<https://www.rfc-editor.org/info/rfc4380>>.

- [RFC5415] Calhoun, P., Ed., Montemurro, M., Ed., and D. Stanley, Ed., "Control And Provisioning of Wireless Access Points (CAPWAP) Protocol Specification", RFC 5415, DOI 10.17487/RFC5415, March 2009, <<https://www.rfc-editor.org/info/rfc5415>>.
- [RFC6633] Gont, F., "Deprecation of ICMP Source Quench Messages", RFC 6633, DOI 10.17487/RFC6633, May 2012, <<https://www.rfc-editor.org/info/rfc6633>>.
- [RFC6660] Briscoe, B., Moncaster, T., and M. Menth, "Encoding Three Pre-Congestion Notification (PCN) States in the IP Header Using a Single Diffserv Codepoint (DSCP)", RFC 6660, DOI 10.17487/RFC6660, July 2012, <<https://www.rfc-editor.org/info/rfc6660>>.
- [RFC6830] Farinacci, D., Fuller, V., Meyer, D., and D. Lewis, "The Locator/ID Separation Protocol (LISP)", RFC 6830, DOI 10.17487/RFC6830, January 2013, <<https://www.rfc-editor.org/info/rfc6830>>.
- [RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, DOI 10.17487/RFC7323, September 2014, <<https://www.rfc-editor.org/info/rfc7323>>.
- [RFC7348] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", RFC 7348, DOI 10.17487/RFC7348, August 2014, <<https://www.rfc-editor.org/info/rfc7348>>.
- [RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <<https://www.rfc-editor.org/info/rfc7567>>.
- [RFC7637] Garg, P., Ed. and Y. Wang, Ed., "NVGRE: Network Virtualization Using Generic Routing Encapsulation", RFC 7637, DOI 10.17487/RFC7637, September 2015, <<https://www.rfc-editor.org/info/rfc7637>>.
- [RFC7713] Mathis, M. and B. Briscoe, "Congestion Exposure (ConEx) Concepts, Abstract Mechanism, and Requirements", RFC 7713, DOI 10.17487/RFC7713, December 2015, <<https://www.rfc-editor.org/info/rfc7713>>.

- [RFC7780] Eastlake 3rd, D., Zhang, M., Perlman, R., Banerjee, A., Ghanwani, A., and S. Gupta, "Transparent Interconnection of Lots of Links (TRILL): Clarifications, Corrections, and Updates", RFC 7780, DOI 10.17487/RFC7780, February 2016, <<https://www.rfc-editor.org/info/rfc7780>>.
- [RFC8084] Fairhurst, G., "Network Transport Circuit Breakers", BCP 208, RFC 8084, DOI 10.17487/RFC8084, March 2017, <<https://www.rfc-editor.org/info/rfc8084>>.
- [RFC8087] Fairhurst, G. and M. Welzl, "The Benefits of Using Explicit Congestion Notification (ECN)", RFC 8087, DOI 10.17487/RFC8087, March 2017, <<https://www.rfc-editor.org/info/rfc8087>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8257] Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <<https://www.rfc-editor.org/info/rfc8257>>.
- [RFC8300] Quinn, P., Ed., Elzur, U., Ed., and C. Pignataro, Ed., "Network Service Header (NSH)", RFC 8300, DOI 10.17487/RFC8300, January 2018, <<https://www.rfc-editor.org/info/rfc8300>>.
- [RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <<https://www.rfc-editor.org/info/rfc8311>>.
- [RFC8926] Gross, J., Ed., Ganga, I., Ed., and T. Sridhar, Ed., "Geneve: Generic Network Virtualization Encapsulation", RFC 8926, DOI 10.17487/RFC8926, November 2020, <<https://www.rfc-editor.org/info/rfc8926>>.
- [UTRAN] 3GPP, "UTRAN Overall Description", Technical Specification TS 25.401.

## Appendix A. Changes in This Version (to be removed by RFC Editor)

From ietf-12 to ietf-13

\* Following 3rd tsvwg WGLC:

- + Formalized update to RFC 3819 in its own subsection (1.1) and referred to it in the abstract
- + Scope: Clarified that the specification of alternative ECN semantics using ECT(1) was not in RFC 4774, but rather in RFC 8311, and that the problem with using a DSCP to indicate alternative semantics has issues at domain boundaries as well as tunnels.
- + Terminology: tightened up definitions of ECN-PDU and Not-ECN-PDU, and removed definition of Congestion Baseline, given it was only used once.
- + Mentioned QCN where feed-backward is first introduced (S.3), referring forward to where it is discussed more deeply (S.4).
- + Clarified that IS-IS solution to adding ECN support to TRILL was not pursued
- + Completely rewrote the rationale for the guideline about a Standard Congestion Monitoring Baseline, to focus on standardization of the otherwise unknown scenario used, rather than the relative usefulness of the info in each approach
- + Explained the re-framing problem better and added fragmentation as another possible cause of the problem
- + Acknowledged new reviewers
- + Updated references, replaced citations of 802.1Qau and 802.1ah with rolled up 802.1Q, and added citations of Fat trees and Clos Networks
- + Numerous other editorial improvements

From ietf-11 to ietf-12

\* Updated references

From ietf-10 to ietf-11

- \* Removed short section (was 3) 'Guidelines for All Cases' because it was out of scope, being covered by RFC 4774. Expanded the Scope section (1.2) to explain all this. Explained that the default encap/decap rules already support certain alternative semantics, particularly all three of the alternative semantics for ECT(1): equivalent to ECT(0) , higher severity than ECT(0), and unmarked but implying different marking semantics from ECT(0).
- \* Clarified why the QCN example was being given even though not about increment deployment of ECN
- \* Pointed to the spoofing issue with feed-backward mode from the Security Considerations section, to aid security review.
- \* Removed any ambiguity in the word 'transport' throughout

From ietf-09 to ietf-10

- \* Updated section 5.1 on "IP-in-IP tunnels with Shim Headers" to be consistent with updates to draft-ietf-tsvwg-rfc6040update-shim.
- \* Removed reference to the ECN nonce, which has been made historic by RFC 8311
- \* Removed "Open Issues" Appendix, given all have been addressed.

From ietf-08 to ietf-09

- \* Updated para in Intro that listed all the IP-in-IP tunnelling protocols, to instead refer to draft-ietf-tsvwg-rfc6040update-shim
- \* Updated section 5.1 on "IP-in-IP tunnels with Shim Headers" to summarize guidance that has evolved as rfc6040update-shim has developed.

From ietf-07 to ietf-08: Refreshed to avoid expiry. Updated references.

From ietf-06 to ietf-07:

- \* Added the people involved in liaisons to the acknowledgements.

From ietf-05 to ietf-06:

- \* Introduction: Added GUE and Geneve as examples of tightly coupled shims between IP headers that cite RFC 6040. And added VXLAN to list of those that do not.
- \* Replaced normative text about tightly coupled shims between IP headers, with reference to new draft-ietf-tsvwg-rfc6040update-shim
- \* Wire Protocol Design: Indication of ECN Support: Added TRILL as an example of a well-design protocol that does not need an indication of ECN support in the wire protocol.
- \* Encapsulation Guidelines: In the case of a Not-ECN-PDU with a CE outer, replaced SHOULD be dropped, with explanations of when SHOULD or MUST are appropriate.
- \* Feed-Up-and-Forward Mode: Explained examples more carefully, referred to PDCP and cited UTRAN spec as well as E-UTRAN.
- \* Updated references.
- \* Marked open issues as resolved, but did not delete Open Issues Appendix (yet).

From ietf-04 to ietf-05:

- \* Explained why tightly coupled shim headers only "SHOULD" comply with RFC 6040, not "MUST".
- \* Updated references

From ietf-03 to ietf-04:

- \* Addressed Richard Scheffenegger's review comments: primarily editorial corrections, and addition of examples for clarity.

From ietf-02 to ietf-03:

- \* Updated references, ad cited RFC4774.

From ietf-01 to ietf-02:

- \* Added Section for guidelines that are applicable in all cases.
- \* Updated references.

From ietf-00 to ietf-01: Updated references.

From briscoe-04 to ietf-00: Changed filename following tsvwg adoption.

From briscoe-03 to 04:

- \* Re-arranged the introduction to describe the purpose of the document first before introducing ECN in more depth. And clarified the introduction throughout.
- \* Added applicability to 3GPP TS 36.300.

From briscoe-02 to 03:

- \* Scope section:
  - + Added dependence on correct propagation of traffic class information
  - + For the feed-backward mode, deemed multicast and anycast out of scope
- \* Ensured all guidelines referring to subnet technologies also refer to tunnels and vice versa by adding applicability sentences at the start of sections 4.1, 4.2, 4.3, 4.4, 4.6 and 5.
- \* Added Security Considerations on ensuring congestion signal fields are classed as immutable and on using end-to-end congestion signal integrity technologies rather than hop-by-hop.

From briscoe-01 to 02:

- \* Added authors: JK & PT
- \* Added
  - + Section 4.1 "IP-in-IP Tunnels with Tightly Coupled Shim Headers"
  - + Section 4.5 "Sequences of Similar Tunnels or Subnets"
  - + roadmap at the start of Section 4, given the subsections have become quite fragmented.
  - + Section 9 "Conclusions"



- \* Clarified why transports are starting to be able to saturate interior links
- \* Under Section 1.1, addressed the question of alternative signal semantics and included multicast & anycast.
- \* Under Section 3.1, included a 3GPP example.
- \* Section 4.2. "Wire Protocol Design":
  - + Altered guideline 2. to make it clear that it only applies to the immediate subnet egress, not later ones
  - + Added a reminder that it is only necessary to check that ECN propagates at the egress, not whether interior nodes mark ECN
  - + Added example of how QCN uses 802.1p to indicate support for QCN.
- \* Added references to Appendix C of RFC6040, about monitoring the amount of congestion signals introduced within a tunnel
- \* Appendix A: Added more issues to be addressed, including plan to produce a standards track update to IP-in-IP tunnel protocols.
- \* Updated acks and references

From briscoe-00 to 01:

- \* Intended status: BCP (was Informational) & updates 3819 added.
- \* Briefer Introduction: Introductory para justifying benefits of ECN. Moved all but a brief enumeration of modes of operation to their own new section (from both Intro & Scope). Introduced incr. deployment as most tricky part.
- \* Tightened & added to terminology section
- \* Structured with Modes of Operation, then Guidelines section for each mode.
- \* Tightened up guideline text to remove vagueness / passive voice / ambiguity and highlight main guidelines as numbered items.
- \* Added Outstanding Document Issues Appendix

\* Updated references

Authors' Addresses

Bob Briscoe  
Independent  
UK

EMail: [ietf@bobbriscoe.net](mailto:ietf@bobbriscoe.net)  
URI: <http://bobbriscoe.net/>

John Kaippallimalil  
Futurewei  
5700 Tennyson Parkway, Suite 600  
Plano, Texas 75024  
USA

EMail: [kjohn@futurewei.com](mailto:kjohn@futurewei.com)

Transport Services (tsv)  
Internet-Draft  
Intended status: Experimental  
Expires: 5 September 2022

K. De Schepper  
Nokia Bell Labs  
B. Briscoe, Ed.  
Independent  
4 March 2022

Explicit Congestion Notification (ECN) Protocol for Very Low Queuing  
Delay (L4S)  
draft-ietf-tsvwg-ecn-l4s-id-25

Abstract

This specification defines the protocol to be used for a new network service called low latency, low loss and scalable throughput (L4S). L4S uses an Explicit Congestion Notification (ECN) scheme at the IP layer that is similar to the original (or 'Classic') ECN approach, except as specified within. L4S uses 'scalable' congestion control, which induces much more frequent control signals from the network and it responds to them with much more fine-grained adjustments, so that very low (typically sub-millisecond on average) and consistently low queuing delay becomes possible for L4S traffic without compromising link utilization. Thus even capacity-seeking (TCP-like) traffic can have high bandwidth and very low delay at the same time, even during periods of high traffic load.

The L4S identifier defined in this document distinguishes L4S from 'Classic' (e.g. TCP-Reno-friendly) traffic. It gives an incremental migration path so that suitably modified network bottlenecks can distinguish and isolate existing traffic that still follows the Classic behaviour, to prevent it degrading the low queuing delay and low loss of L4S traffic. This specification defines the rules that L4S transports and network elements need to follow with the intention that L4S flows neither harm each other's performance nor that of Classic traffic. Examples of new active queue management (AQM) marking algorithms and examples of new transports (whether TCP-like or real-time) are specified separately.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 5 September 2022.

## Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction . . . . .	4
1.1. Latency, Loss and Scaling Problems . . . . .	5
1.2. Terminology . . . . .	7
1.3. Scope . . . . .	9
2. Choice of L4S Packet Identifier: Requirements . . . . .	10
3. L4S Packet Identification . . . . .	11
4. Transport Layer Behaviour (the 'Prague Requirements') . . . . .	11
4.1. Codepoint Setting . . . . .	12
4.2. Prerequisite Transport Feedback . . . . .	12
4.3. Prerequisite Congestion Response . . . . .	13
4.3.1. Guidance on Congestion Response in the RFC Series . . . . .	16
4.4. Filtering or Smoothing of ECN Feedback . . . . .	19
5. Network Node Behaviour . . . . .	19
5.1. Classification and Re-Marking Behaviour . . . . .	19
5.2. The Strength of L4S CE Marking Relative to Drop . . . . .	21
5.3. Exception for L4S Packet Identification by Network Nodes with Transport-Layer Awareness . . . . .	22
5.4. Interaction of the L4S Identifier with other Identifiers . . . . .	22
5.4.1. DualQ Examples of Other Identifiers Complementing L4S Identifiers . . . . .	22
5.4.1.1. Inclusion of Additional Traffic with L4S . . . . .	22
5.4.1.2. Exclusion of Traffic From L4S Treatment . . . . .	24
5.4.1.3. Generalized Combination of L4S and Other Identifiers . . . . .	25

5.4.2.	Per-Flow Queuing Examples of Other Identifiers Complementing L4S Identifiers . . . . .	27
5.5.	Limiting Packet Bursts from Links . . . . .	27
5.5.1.	Limiting Packet Bursts from Links Fed by an L4S AQM . . . . .	27
5.5.2.	Limiting Packet Bursts from Links Upstream of an L4S AQM . . . . .	28
6.	Behaviour of Tunnels and Encapsulations . . . . .	28
6.1.	No Change to ECN Tunnels and Encapsulations in General . . . . .	28
6.2.	VPN Behaviour to Avoid Limitations of Anti-Replay . . . . .	29
7.	L4S Experiments . . . . .	30
7.1.	Open Questions . . . . .	30
7.2.	Open Issues . . . . .	32
7.3.	Future Potential . . . . .	32
8.	IANA Considerations . . . . .	33
9.	Security Considerations . . . . .	33
10.	Acknowledgements . . . . .	34
11.	References . . . . .	35
11.1.	Normative References . . . . .	35
11.2.	Informative References . . . . .	35
Appendix A.	Rationale for the 'Prague L4S Requirements' . . . . .	45
A.1.	Rationale for the Requirements for Scalable Transport Protocols . . . . .	46
A.1.1.	Use of L4S Packet Identifier . . . . .	46
A.1.2.	Accurate ECN Feedback . . . . .	46
A.1.3.	Capable of Replacement by Classic Congestion Control . . . . .	46
A.1.4.	Fall back to Classic Congestion Control on Packet Loss . . . . .	47
A.1.5.	Coexistence with Classic Congestion Control at Classic ECN bottlenecks . . . . .	48
A.1.6.	Reduce RTT dependence . . . . .	51
A.1.7.	Scaling down to fractional congestion windows . . . . .	52
A.1.8.	Measuring Reordering Tolerance in Time Units . . . . .	53
A.2.	Scalable Transport Protocol Optimizations . . . . .	56
A.2.1.	Setting ECT in Control Packets and Retransmissions . . . . .	56
A.2.2.	Faster than Additive Increase . . . . .	56
A.2.3.	Faster Convergence at Flow Start . . . . .	57
Appendix B.	Compromises in the Choice of L4S Identifier . . . . .	57
Appendix C.	Potential Competing Uses for the ECT(1) Codepoint . . . . .	62
C.1.	Integrity of Congestion Feedback . . . . .	62
C.2.	Notification of Less Severe Congestion than CE . . . . .	63
Authors' Addresses	. . . . .	64

## 1. Introduction

This specification defines the protocol to be used for a new network service called low latency, low loss and scalable throughput (L4S). L4S uses an Explicit Congestion Notification (ECN) scheme at the IP layer with the same set of codepoint transitions as the original (or 'Classic') Explicit Congestion Notification (ECN [RFC3168]). RFC 3168 required an ECN mark to be equivalent to a drop, both when applied in the network and when responded to by a transport. Unlike Classic ECN marking, the network applies L4S marking more immediately and more aggressively than drop, and the transport response to each mark is reduced and smoothed relative to that for drop. The two changes counterbalance each other so that the throughput of an L4S flow will be roughly the same as a comparable non-L4S flow under the same conditions. Nonetheless, the much more frequent ECN control signals and the finer responses to these signals result in very low queuing delay without compromising link utilization, and this low delay can be maintained during high load. For instance, queuing delay under heavy and highly varying load with the example DCTCP/DualQ solution cited below on a DSL or Ethernet link is sub-millisecond on average and roughly 1 to 2 milliseconds at the 99th percentile without losing link utilization [DualPI2Linux], [DCTtH19]. Note that the inherent queuing delay while waiting to acquire a discontinuous medium such as WiFi has to be minimized in its own right, so it would be additional to the above (see section 6.3 of the L4S architecture [I-D.ietf-tsvwg-l4s-arch]).

L4S relies on 'scalable' congestion controls for these delay properties and for preserving low delay as flow rate scales, hence the name. The congestion control used in Data Center TCP (DCTCP) is an example of a scalable congestion control, but DCTCP is applicable solely to controlled environments like data centres [RFC8257], because it is too aggressive to co-exist with existing TCP-Reno-friendly traffic. The DualQ Coupled AQM, which is defined in a complementary experimental specification [I-D.ietf-tsvwg-aqm-dualq-coupled], is an AQM framework that enables scalable congestion controls derived from DCTCP to co-exist with existing traffic, each getting roughly the same flow rate when they compete under similar conditions. Note that a scalable congestion control is still not safe to deploy on the Internet unless it satisfies the requirements listed in Section 4.

L4S is not only for elastic (TCP-like) traffic - there are scalable congestion controls for real-time media, such as the L4S variant of the SCReAM [RFC8298] real-time media congestion avoidance technique (RMCAT). The factor that distinguishes L4S from Classic traffic is its behaviour in response to congestion. The transport wire protocol, e.g. TCP, QUIC, SCTP, DCCP, RTP/RTCP, is orthogonal (and therefore not suitable for distinguishing L4S from Classic packets).

The L4S identifier defined in this document is the key piece that distinguishes L4S from 'Classic' (e.g. Reno-friendly) traffic. It gives an incremental migration path so that suitably modified network bottlenecks can distinguish and isolate existing Classic traffic from L4S traffic to prevent the former from degrading the very low delay and loss of the new scalable transports, without harming Classic performance at these bottlenecks. Initial implementation of the separate parts of the system has been motivated by the performance benefits.

#### 1.1. Latency, Loss and Scaling Problems

Latency is becoming the critical performance factor for many (most?) applications on the public Internet, e.g. interactive Web, Web services, voice, conversational video, interactive video, interactive remote presence, instant messaging, online gaming, remote desktop, cloud-based applications, and video-assisted remote control of machinery and industrial processes. In the 'developed' world, further increases in access network bit-rate offer diminishing returns, whereas latency is still a multi-faceted problem. In the last decade or so, much has been done to reduce propagation time by placing caches or servers closer to users. However, queuing remains a major intermittent component of latency.

The Diffserv architecture provides Expedited Forwarding [RFC3246], so that low latency traffic can jump the queue of other traffic. If growth in high-throughput latency-sensitive applications continues, periods with solely latency-sensitive traffic will become increasingly common on links where traffic aggregation is low. For instance, on the access links dedicated to individual sites (homes, small enterprises or mobile devices). These links also tend to become the path bottleneck under load. During these periods, if all the traffic were marked for the same treatment, at these bottlenecks Diffserv would make no difference. Instead, it becomes imperative to remove the underlying causes of any unnecessary delay.

The bufferbloat project has shown that excessively-large buffering ('bufferbloat') has been introducing significantly more delay than the underlying propagation time. These delays appear only intermittently -- only when a capacity-seeking (e.g. TCP) flow is long enough for the queue to fill the buffer, making every packet in other flows sharing the buffer sit through the queue.

Active queue management (AQM) was originally developed to solve this problem (and others). Unlike Diffserv, which gives low latency to some traffic at the expense of others, AQM controls latency for all traffic in a class. In general, AQM methods introduce an increasing level of discard from the buffer the longer the queue persists above a shallow threshold. This gives sufficient signals to capacity-seeking (aka. greedy) flows to keep the buffer empty for its intended purpose: absorbing bursts. However, RED [RFC2309] and other algorithms from the 1990s were sensitive to their configuration and hard to set correctly. So, this form of AQM was not widely deployed.

More recent state-of-the-art AQM methods, e.g. FQ-CoDel [RFC8290], PIE [RFC8033], Adaptive RED [ARED01], are easier to configure, because they define the queuing threshold in time not bytes, so it is invariant for different link rates. However, no matter how good the AQM, the sawtooth sending window of a Classic congestion control will either cause queuing delay to vary or cause the link to be underutilized. Even with a perfectly tuned AQM, the additional queuing delay will be of the same order as the underlying speed-of-light delay across the network, thereby roughly doubling the total round-trip time.

If a sender's own behaviour is introducing queuing delay variation, no AQM in the network can 'un-vary' the delay without significantly compromising link utilization. Even flow-queuing (e.g. [RFC8290]), which isolates one flow from another, cannot isolate a flow from the delay variations it inflicts on itself. Therefore those applications that need to seek out high bandwidth but also need low latency will have to migrate to scalable congestion control.

Altering host behaviour is not enough on its own though. Even if hosts adopt low latency behaviour (scalable congestion controls), they need to be isolated from the behaviour of existing Classic congestion controls that induce large queue variations. L4S enables that migration by providing latency isolation in the network and distinguishing the two types of packets that need to be isolated: L4S and Classic. L4S isolation can be achieved with a queue per flow (e.g. [RFC8290]) but a DualQ [I-D.ietf-tsvwg-aqm-dualq-coupled] is sufficient, and actually gives better tail latency. Both approaches are addressed in this document.



The DualQ solution was developed to make very low latency available without requiring per-flow queues at every bottleneck. This was because per-flow-queuing (FQ) has well-known downsides - not least the need to inspect transport layer headers in the network, which makes it incompatible with privacy approaches such as IPsec VPN tunnels, and incompatible with link layer queue management, where transport layer headers can be hidden, e.g. 5G.

Latency is not the only concern addressed by L4S: It was known when TCP congestion avoidance was first developed that it would not scale to high bandwidth-delay products (footnote 6 of Jacobson and Karels [TCP-CA]). Given regular broadband bit-rates over WAN distances are already [RFC3649] beyond the scaling range of Reno congestion control, 'less unscalable' Cubic [RFC8312] and Compound [I-D.sridharan-tcpm-ctcp] variants of TCP have been successfully deployed. However, these are now approaching their scaling limits. Unfortunately, fully scalable congestion controls such as DCTCP [RFC8257] outcompete Classic ECN congestion controls sharing the same queue, which is why they have been confined to private data centres or research testbeds.

It turns out that these scalable congestion control algorithms that solve the latency problem can also solve the scalability problem of Classic congestion controls. The finer sawteeth in the congestion window have low amplitude, so they cause very little queuing delay variation and the average time to recover from one congestion signal to the next (the average duration of each sawtooth) remains invariant, which maintains constant tight control as flow-rate scales. A background paper [DCttH19] gives the full explanation of why the design solves both the latency and the scaling problems, both in plain English and in more precise mathematical form. The explanation is summarised without the maths in Section 4 of the L4S architecture [I-D.ietf-tsvwg-l4s-arch].

## 1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

Note: The L4S architecture [I-D.ietf-tsvwg-l4s-arch] repeats the following definitions, but if there are accidental differences those below take precedence.

Classic Congestion Control: A congestion control behaviour that can

co-exist with standard Reno [RFC5681] without causing significantly negative impact on its flow rate [RFC5033]. With Classic congestion controls, such as Reno or Cubic, because flow rate has scaled since TCP congestion control was first designed in 1988, it now takes hundreds of round trips (and growing) to recover after a congestion signal (whether a loss or an ECN mark) as shown in the examples in section 5.1 of the L4S architecture [I-D.ietf-tsvwg-l4s-arch] and in [RFC3649]. Therefore control of queuing and utilization becomes very slack, and the slightest disturbances (e.g. from new flows starting) prevent a high rate from being attained.

**Scalable Congestion Control:** A congestion control where the average time from one congestion signal to the next (the recovery time) remains invariant as the flow rate scales, all other factors being equal. This maintains the same degree of control over queueing and utilization whatever the flow rate, as well as ensuring that high throughput is robust to disturbances. For instance, DCTCP averages 2 congestion signals per round-trip whatever the flow rate, as do other recently developed scalable congestion controls, e.g. Relentless TCP [Mathis09], TCP Prague [I-D.briscoe-iccrp-prague-congestion-control], [PragueLinux], BBRv2 [BBRv2], [I-D.cardwell-iccrp-bbr-congestion-control] and the L4S variant of SCREAM for real-time media [SCReAM], [RFC8298]). See Section 4.3 for more explanation.

**Classic service:** The Classic service is intended for all the congestion control behaviours that co-exist with Reno [RFC5681] (e.g. Reno itself, Cubic [RFC8312], Compound [I-D.sridharan-tcpm-ctcp], TFRC [RFC5348]). The term 'Classic queue' means a queue providing the Classic service.

**Low-Latency, Low-Loss Scalable throughput (L4S) service:** The 'L4S' service is intended for traffic from scalable congestion control algorithms, such as TCP Prague [I-D.briscoe-iccrp-prague-congestion-control], which was derived from DCTCP [RFC8257]. The L4S service is for more general traffic than just TCP Prague -- it allows the set of congestion controls with similar scaling properties to Prague to evolve, such as the examples listed above (Relentless, SCReAM). The term 'L4S queue' means a queue providing the L4S service.

The terms Classic or L4S can also qualify other nouns, such as 'queue', 'codepoint', 'identifier', 'classification', 'packet', 'flow'. For example: an L4S packet means a packet with an L4S identifier sent from an L4S congestion control.

Both Classic and L4S services can cope with a proportion of unresponsive or less-responsive traffic as well, but in the L4S case its rate has to be smooth enough or low enough not to build a queue (e.g. DNS, VoIP, game sync datagrams, etc).

**Reno-friendly:** The subset of Classic traffic that is friendly to the standard Reno congestion control defined for TCP in [RFC5681]. The TFRC spec. [RFC5348] indirectly implies that 'friendly' is defined as "generally within a factor of two of the sending rate of a TCP flow under the same conditions". Reno-friendly is used here in place of 'TCP-friendly', given the latter has become imprecise, because the TCP protocol is now used with so many different congestion control behaviours, and Reno is used in non-TCP transports such as QUIC [RFC9000].

**Classic ECN:** The original Explicit Congestion Notification (ECN) protocol [RFC3168], which requires ECN signals to be treated the same as drops, both when generated in the network and when responded to by the sender. For L4S, the names used for the four codepoints of the 2-bit IP-ECN field are unchanged from those defined in [RFC3168]: Not ECT, ECT(0), ECT(1) and CE, where ECT stands for ECN-Capable Transport and CE stands for Congestion Experienced. A packet marked with the CE codepoint is termed 'ECN-marked' or sometimes just 'marked' where the context makes ECN obvious.

**Site:** A home, mobile device, small enterprise or campus, where the network bottleneck is typically the access link to the site. Not all network arrangements fit this model but it is a useful, widely applicable generalization.

### 1.3. Scope

The new L4S identifier defined in this specification is applicable for IPv4 and IPv6 packets (as for Classic ECN [RFC3168]). It is applicable for the unicast, multicast and anycast forwarding modes.

The L4S identifier is an orthogonal packet classification to the Differentiated Services Code Point (DSCP) [RFC2474]. Section 5.4 explains what this means in practice.

This document is intended for experimental status, so it does not update any standards track RFCs. Therefore it depends on [RFC8311], which is a standards track specification that:

- \* updates the ECN proposed standard [RFC3168] to allow experimental track RFCs to relax the requirement that an ECN mark must be equivalent to a drop (when the network applies markings and/or

when the sender responds to them). For instance, in the ABE experiment [RFC8511] this permits a sender to respond less to ECN marks than to drops;

- \* changes the status of the experimental ECN nonce [RFC3540] to historic;
- \* makes consequent updates to the following additional proposed standard RFCs to reflect the above two bullets:
  - ECN for RTP [RFC6679];
  - the congestion control specifications of various DCCP congestion control identifier (CCID) profiles [RFC4341], [RFC4342], [RFC5622].

This document is about identifiers that are used for interoperation between hosts and networks. So the audience is broad, covering developers of host transports and network AQMs, as well as covering how operators might wish to combine various identifiers, which would require flexibility from equipment developers.

## 2. Choice of L4S Packet Identifier: Requirements

This subsection briefly records the process that led to the chosen L4S identifier.

The identifier for packets using the Low Latency, Low Loss, Scalable throughput (L4S) service needs to meet the following requirements:

- \* it SHOULD survive end-to-end between source and destination endpoints: across the boundary between host and network, between interconnected networks, and through middleboxes;
- \* it SHOULD be visible at the IP layer;
- \* it SHOULD be common to IPv4 and IPv6 and transport-agnostic;
- \* it SHOULD be incrementally deployable;
- \* it SHOULD enable an AQM to classify packets encapsulated by outer IP or lower-layer headers;
- \* it SHOULD consume minimal extra codepoints;
- \* it SHOULD be consistent on all the packets of a transport layer flow, so that some packets of a flow are not served by a different queue to others.

Whether the identifier would be recoverable if the experiment failed is a factor that could be taken into account. However, this has not been made a requirement, because that would favour schemes that would be easier to fail, rather than those more likely to succeed.

It is recognised that any choice of identifier is unlikely to satisfy all these requirements, particularly given the limited space left in the IP header. Therefore a compromise will always be necessary, which is why all the above requirements are expressed with the word 'SHOULD' not 'MUST'.

After extensive assessment of alternative schemes, "ECT(1) and CE codepoints" was chosen as the best compromise. Therefore this scheme is defined in detail in the following sections, while Appendix B records its pros and cons against the above requirements.

### 3. L4S Packet Identification

The L4S treatment is an experimental track alternative packet marking treatment to the Classic ECN treatment in [RFC3168], which has been updated by [RFC8311] to allow experiments such as the one defined in the present specification. [RFC4774] discusses some of the issues and evaluation criteria when defining alternative ECN semantics. Like Classic ECN, L4S ECN identifies both network and host behaviour: it identifies the marking treatment that network nodes are expected to apply to L4S packets, and it identifies packets that have been sent from hosts that are expected to comply with a broad type of sending behaviour.

For a packet to receive L4S treatment as it is forwarded, the sender sets the ECN field in the IP header to the ECT(1) codepoint. See Section 4 for full transport layer behaviour requirements, including feedback and congestion response.

A network node that implements the L4S service always classifies arriving ECT(1) packets for L4S treatment and by default classifies CE packets for L4S treatment unless the heuristics described in Section 5.3 are employed. See Section 5 for full network element behaviour requirements, including classification, ECN-marking and interaction of the L4S identifier with other identifiers and per-hop behaviours.

### 4. Transport Layer Behaviour (the 'Prague Requirements')

#### 4.1. Codepoint Setting

A sender that wishes a packet to receive L4S treatment as it is forwarded, MUST set the ECN field in the IP header (v4 or v6) to the ECT(1) codepoint.

#### 4.2. Prerequisite Transport Feedback

For a transport protocol to provide scalable congestion control (Section 4.3) it MUST provide feedback of the extent of CE marking on the forward path. When ECN was added to TCP [RFC3168], the feedback method reported no more than one CE mark per round trip. Some transport protocols derived from TCP mimic this behaviour while others report the accurate extent of ECN marking. This means that some transport protocols will need to be updated as a prerequisite for scalable congestion control. The position for a few well-known transport protocols is given below.

**TCP:** Support for the accurate ECN feedback requirements [RFC7560] (such as that provided by AccECN [I-D.ietf-tcpm-accurate-ecn]) by both ends is a prerequisite for scalable congestion control in TCP. Therefore, the presence of ECT(1) in the IP headers even in one direction of a TCP connection will imply that both ends support accurate ECN feedback. However, the converse does not apply. So even if both ends support AccECN, either of the two ends can choose not to use a scalable congestion control, whatever the other end's choice.

**SCTP:** A suitable ECN feedback mechanism for SCTP could add a chunk to report the number of received CE marks (e.g. [I-D.stewart-tsvwg-sctpecn]), and update the ECN feedback protocol sketched out in Appendix A of the original standards track specification of SCTP [RFC4960].

**RTP over UDP:** A prerequisite for scalable congestion control is for both (all) ends of one media-level hop to signal ECN support [RFC6679] and use the new generic RTCP feedback format of [RFC8888]. The presence of ECT(1) implies that both (all) ends of that media-level hop support ECN. However, the converse does not apply. So each end of a media-level hop can independently choose not to use a scalable congestion control, even if both ends support ECN.

**QUIC:** Support for sufficiently fine-grained ECN feedback is provided by the v1 IETF QUIC transport [RFC9000].

**DCCP:** The ACK vector in DCCP [RFC4340] is already sufficient to

report the extent of CE marking as needed by a scalable congestion control.

#### 4.3. Prerequisite Congestion Response

As a condition for a host to send packets with the L4S identifier (ECT(1)), it SHOULD implement a congestion control behaviour that ensures that, in steady state, the average duration between induced ECN marks does not increase as flow rate scales up, all other factors being equal. This is termed a scalable congestion control. This invariant duration ensures that, as flow rate scales, the average period with no feedback information about capacity does not become excessive. It also ensures that queue variations remain small, without having to sacrifice utilization.

With a congestion control that sawtooths to probe capacity, this duration is called the recovery time, because each time the sawtooth yields, on average it take this time to recover to its previous high point. A scalable congestion control does not have to sawtooth, but it has to coexist with scalable congestion controls that do.

For instance, for DCTCP [RFC8257], TCP Prague [I-D.briscoe-iccrp-prague-congestion-control], [PragueLinux] and the L4S variant of SCReAM [RFC8298], the average recovery time is always half a round trip (or half a reference round trip), whatever the flow rate.

As with all transport behaviours, a detailed specification (probably an experimental RFC) is expected for each congestion control, following the guidelines for specifying new congestion control algorithms in [RFC5033]. In addition it is expected to document these L4S-specific matters, specifically the timescale over which the proportionality is averaged, and control of burstiness. The recovery time requirement above is worded as a 'SHOULD' rather than a 'MUST' to allow reasonable flexibility for such implementations.

The condition 'all other factors being equal', allows the recovery time to be different for different round trip times, as long as it does not increase with flow rate for any particular RTT.

Saying that the recovery time remains roughly invariant is equivalent to saying that the number of ECN CE marks per round trip remains invariant as flow rate scales, all other factors being equal. For instance, an average recovery time of half of 1 RTT is equivalent to 2 ECN marks per round trip. For those familiar with steady-state congestion response functions, it is also equivalent to say that the congestion window is inversely proportional to the proportion of bytes in packets marked with the CE codepoint (see section 2 of [PI2]).

In order to coexist safely with other Internet traffic, a scalable congestion control MUST NOT tag its packets with the ECT(1) codepoint unless it complies with the following bulleted requirements:

1. A scalable congestion control MUST be capable of being replaced by a Classic congestion control (by application and/or by administrative control). If a Classic congestion control is activated, it will not tag its packets with the ECT(1) codepoint (see Appendix A.1.3 for rationale).
2. As well as responding to ECN markings, a scalable congestion control MUST react to packet loss in a way that will coexist safely with Classic congestion controls such as standard Reno [RFC5681], as required by [RFC5033] (see Appendix A.1.4 for rationale).
3. In uncontrolled environments, monitoring MUST be implemented to support detection of problems with an ECN-capable AQM at the path bottleneck that appears not to support L4S and might be in a shared queue. Such monitoring SHOULD be applied to live traffic that is using Scalable congestion control. Alternatively, monitoring need not be applied to live traffic, if monitoring has been arranged to cover the paths that live traffic takes through uncontrolled environments.

A function to detect the above problems with an ECN-capable AQM MUST also be implemented and used. The detection function SHOULD be capable of making the congestion control adapt its ECN-marking response in real-time to coexist safely with Classic congestion controls such as standard Reno [RFC5681], as required by [RFC5033]. This could be complemented by more detailed offline detection of potential problems. If only offline detection is used and potential problems with such an AQM are detected on certain paths, the scalable congestion control MUST be replaced by a Classic congestion control, at least for the problem paths.

See Section 4.3.1, Appendix A.1.5 and the L4S operational guidance [I-D.ietf-tsvwg-l4sops] for rationale.



Note that a scalable congestion control is not expected to change to setting ECT(0) while it transiently adapts to coexist with Classic congestion controls, whereas a replacement congestion control that solely behaves in the Classic way will set ECT(0).

4. In the range between the minimum likely RTT and typical RTTs expected in the intended deployment scenario, a scalable congestion control MUST converge towards a rate that is as independent of RTT as is possible without compromising stability or efficiency (see Appendix A.1.6 for rationale).
5. A scalable congestion control SHOULD remain responsive to congestion when typical RTTs over the public Internet are significantly smaller because they are no longer inflated by queuing delay. It would be preferable for the minimum window of a scalable congestion control to be lower than 1 segment rather than use the timeout approach described for TCP in S.6.1.2 of the ECN spec [RFC3168] (or an equivalent for other transports). However, a lower minimum is not set as a formal requirement for L4S experiments (see Appendix A.1.7 for rationale).
6. A scalable congestion control's loss detection SHOULD be resilient to reordering over an adaptive time interval that scales with throughput and adapts to reordering (as in RACK [RFC8985]), as opposed to counting only in fixed units of packets (as in the 3 DupACK rule of New Reno [RFC5681] and [RFC6675], which is not scalable). As data rates increase (e.g., due to new and/or improved technology), congestion controls that detect loss by counting in units of packets become more likely to incorrectly treat reordering events as congestion-caused loss events (see Appendix A.1.8 for further rationale). This requirement does not apply to congestion controls that are solely used in controlled environments where the network introduces hardly any reordering.
7. A scalable congestion control is expected to limit the queue caused by bursts of packets. It would not seem necessary to set the limit any lower than 10% of the minimum RTT expected in a typical deployment (e.g. additional queuing of roughly 250 us for the public Internet). This would be converted to a number of packets under the worst-case assumption that the bottleneck link capacity equals the current flow rate. No normative requirement to limit bursts is given here and, until there is more industry experience from the L4S experiment, it is not even known whether one is needed - it seems to be in an L4S sender's self-interest to limit bursts.

Each sender in a session can use a scalable congestion control independently of the congestion control used by the receiver(s) when they send data. Therefore there might be ECT(1) packets in one direction and ECT(0) or Not-ECT in the other.

Later (Section 5.4.1.1) this document discusses the conditions for mixing other "'Safe' Unresponsive Traffic" (e.g. DNS, LDAP, NTP, voice, game sync packets) with L4S traffic. To be clear, although such traffic can share the same queue as L4S traffic, it is not appropriate for the sender to tag it as ECT(1), except in the (unlikely) case that it satisfies the above conditions.

#### 4.3.1. Guidance on Congestion Response in the RFC Series

RFC 3168 requires the congestion responses to a CE-marked packet and a dropped packet to be the same. RFC 8311 is a standards-track update to RFC 3168 intended to enable experimentation with ECN, including the L4S experiment. RFC 8311 allows an experimental congestion control's response to a CE-marked packet to differ from the response to a dropped packet, provided that the differences are documented in an experimental RFC, such as the present document.

BCP 124 [RFC4774] gives guidance to protocol designers, when specifying alternative semantics for the ECN field. RFC 8311 explained that it did not need to update the best current practice in BCP 124 in order to relax the 'equivalence with drop' requirement because, although BCP 124 quotes the same requirement from RFC 3168, the BCP does not impose requirements based on it. BCP 124 describes three options for incremental deployment, with Option 3 (in Section 4.3 of BCP 124) best matching the L4S case. Option 3's requirement for end-nodes is that they respond to CE marks "in a way that is friendly to flows using IETF-conformant congestion control." This echoes other general congestion control requirements in the RFC series, for example [RFC5033], which says "...congestion controllers that have a significantly negative impact on traffic using standard congestion control may be suspect", or [RFC8085] concerning UDP congestion control says "Bulk-transfer applications that choose not to implement TFRC or TCP-like windowing SHOULD implement a congestion control scheme that results in bandwidth (capacity) use that competes fairly with TCP within an order of magnitude."

The third normative bullet in Section 4.3 above (which concerns L4S response to congestion from a Classic ECN AQM) aims to ensure that these 'coexistence' requirements are satisfied, but it makes some compromises. This subsection highlights and justifies those compromises and Appendix A.1.5 and the L4S operational guidance [I-D.ietf-tsvwg-l4sops] give detailed analysis, examples and references (the normative text in that bullet takes precedence if any

informative elaboration leads to ambiguity). The approach is based on an assessment of the risk of harm, which is a combination of the prevalence of the conditions necessary for harm to occur, and the potential severity of the harm if they do.

Prevalence: There are three cases:

- \* Drop Tail: Coexistence between L4S and Classic flows is not in doubt where the bottleneck does not support any form of ECN, which has remained by far the most prevalent case since the ECN RFC was published in 2001.
- \* L4S: Coexistence is not in doubt if the bottleneck supports L4S.
- \* Classic ECN [RFC3168]: The compromises centre around cases where the bottleneck supports Classic ECN but not L4S. But it depends on which sub-case:
  - Shared Queue with Classic ECN: The members of the Transport Working group are not aware of any current deployments of single-queue Classic ECN bottlenecks in the Internet. Nonetheless, at the scale of the Internet, rarity need not imply small numbers, nor that there will be rarity in future.
  - Per-Flow-queues with Classic ECN: Most AQMs with per-flow-queuing (FQ) deployed from 2012 onwards had Classic ECN enabled by default, specifically FQ-CoDel [RFC8290] and COBALT [COBALT]. But the compromises only apply to the second of two further sub-cases:
    - o With per-flow-queuing, co-existence between Classic and L4S flows is not normally a problem, because different flows are not meant to be in the same queue (BCP 124 [RFC4774] did not foresee the introduction of per-flow-queuing, which appeared as a potential isolation technique some eight years after the BCP was published).
    - o However, the isolation between L4S and Classic flows is not perfect in cases where the hashes of flow IDs collide or where multiple flows within a layer-3 VPN are encapsulated within one flow ID.

To summarize, the coexistence problem is confined to cases of imperfect flow isolation in an FQ, or in potential cases where a Classic ECN AQM has been deployed in a shared queue (see the L4S operational guidance [I-D.ietf-tsvwg-l4sops] for further details

including recent surveys attempting to quantify prevalence). Further, if one of these cases does occur, the coexistence problem does not arise unless sources of Classic and L4S flows are simultaneously sharing the same bottleneck queue (e.g. different applications in the same household) and flows of each type have to be large enough to coincide for long enough for any throughput imbalance to have developed.

**Severity:** Where long-running L4S and Classic flows coincide in a shared queue, testing of one L4S congestion control (TCP Prague) has found that the imbalance in average throughput between an L4S and a Classic flow can reach 25:1 in favour of L4S in the worst case [ecn-fallback]. However, when capacity is most scarce, the Classic flow gets a higher proportion of the link, for instance over a 4 Mb/s link the throughput ratio is below ~10:1 over paths with a base RTT below 100 ms, and falls below ~5:1 for base RTTs below 20ms.

These throughput ratios can clearly fall well outside current RFC guidance on coexistence. However, the tendency towards leaving a greater share for Classic flows at lower link rate and the very limited prevalence of the conditions necessary for harm to occur led to the possibility of allowing the RFC requirements to be compromised, albeit briefly:

- \* The recommended approach is still to detect and adapt to a Classic ECN AQM in real-time, which is fully consistent with all the RFCs on coexistence. In other words, the "SHOULD"s in the third bullet of Section 4.3 above expect the sender to implement something similar to the proof of concept code that detects the presence of a Classic ECN AQM and falls back to a Classic congestion response within a few round trips [ecn-fallback]. However, although this code reliably detects a Classic ECN AQM, the current code can also wrongly categorize an L4S AQM as Classic, most often in cases when link rate is low or RTT is high. Although this is the safe way round, and although implementers are expected to be able to improve on this proof of concept, concerns have been raised that implementers might lose faith in such detection and disable it.
- \* Therefore the third bullet in Section 4.3 above allows a compromise where coexistence could diverge from the requirements in the RFC Series briefly, but mandatory monitoring is required, in order to detect such cases and trigger remedial action. This approach tolerates a brief divergence from the RFCs given the likely low prevalence and given harm here means a flow progresses more slowly than otherwise, but it does progress. The L4S operational guidance [I-D.ietf-tsvwg-l4sops] outlines a range of example remedial actions that include alterations either to the

sender or to the network. However, the final normative requirement in the third bullet of Section 4.3 above places ultimate responsibility for remedial action on the sender. If coexistence problems with a Classic ECN AQM are detected (implying they have not been resolved by the network), it says the sender "MUST" revert to a Classic congestion control."

[I-D.ietf-tsvwg-l4sops] also gives example ways in which L4S congestion controls can be rolled out initially in lower risk scenarios.

#### 4.4. Filtering or Smoothing of ECN Feedback

Section 5.2 below specifies that an L4S AQM is expected to signal L4S ECN immediately, to avoid introducing delay due to filtering or smoothing. This contrasts with a Classic AQM, which filters out variations in the queue before signalling ECN marking or drop. In the L4S architecture [I-D.ietf-tsvwg-l4s-arch], responsibility for smoothing out these variations shifts to the sender's congestion control.

This shift of responsibility has the advantage that each sender can smooth variations over a timescale proportionate to its own RTT. Whereas, in the Classic approach, the network doesn't know the RTTs of any of the flows, so it has to smooth out variations for a worst-case RTT to ensure stability. For all the typical flows with shorter RTT than the worst-case, this makes congestion control unnecessarily sluggish.

This also gives an L4S sender the choice not to smooth, depending on its context (start-up, congestion avoidance, etc). Therefore, this document places no requirement on an L4S congestion control to smooth out variations in any particular way. Implementers are encouraged to openly publish the approach they take to smoothing, and the results and experience they gain during the L4S experiment.

### 5. Network Node Behaviour

#### 5.1. Classification and Re-Marking Behaviour

A network node that implements the L4S service:

- \* MUST classify arriving ECT(1) packets for L4S treatment, unless overridden by another classifier (e.g., see Section 5.4.1.2);

- \* MUST classify arriving CE packets for L4S treatment as well, unless overridden by a another classifier or unless the exception referred to next applies;

CE packets might have originated as ECT(1) or ECT(0), but the above rule to classify them as if they originated as ECT(1) is the safe choice (see Appendix B for rationale). The exception is where some flow-aware in-network mechanism happens to be available for distinguishing CE packets that originated as ECT(0), as described in Section 5.3, but there is no implication that such a mechanism is necessary.

An L4S AQM treatment follows similar codepoint transition rules to those in RFC 3168. Specifically, the ECT(1) codepoint MUST NOT be changed to any other codepoint than CE, and CE MUST NOT be changed to any other codepoint. An ECT(1) packet is classified as ECN-capable and, if congestion increases, an L4S AQM algorithm will increasingly mark the ECN field as CE, otherwise forwarding packets unchanged as ECT(1). Necessary conditions for an L4S marking treatment are defined in Section 5.2.

Under persistent overload an L4S marking treatment MUST begin applying drop to L4S traffic until the overload episode has subsided, as recommended for all AQM methods in [RFC7567] (Section 4.2.1), which follows the similar advice in RFC 3168 (Section 7). During overload, it MUST apply the same drop probability to L4S traffic as it would to Classic traffic.

Where an L4S AQM is transport-aware, this requirement could be satisfied by using drop in only the most overloaded individual per-flow AQMs. In a DualQ with flow-aware queue protection (e.g. [I-D.briscoe-docsis-q-protection]), this could be achieved by redirecting packets in those flows contributing most to the overload out of the L4S queue so that they are subjected to drop in the Classic queue.

For backward compatibility in uncontrolled environments, a network node that implements the L4S treatment MUST also implement an AQM treatment for the Classic service as defined in Section 1.2. This Classic AQM treatment need not mark ECT(0) packets, but if it does, see Section 5.2 for the strengths of the markings relative to drop. It MUST classify arriving ECT(0) and Not-ECT packets for treatment by this Classic AQM (for the DualQ Coupled AQM, see the extensive discussion on classification in Sections 2.3 and 2.5.1.1 of [I-D.ietf-tsvwg-aqm-dualq-coupled]).

In case unforeseen problems arise with the L4S experiment, it MUST be possible to configure an L4S implementation to disable the L4S treatment. Once disabled, all packets of all ECN codepoints will receive Classic treatment and ECT(1) packets MUST be treated as if they were Not-ECT.

## 5.2. The Strength of L4S CE Marking Relative to Drop

The relative strengths of L4S CE and drop are irrelevant where AQMs are implemented in separate queues per-application-flow, which are then explicitly scheduled (e.g. with an FQ scheduler as in FQ-CoDel [RFC8290]). Nonetheless, the relationship between them needs to be defined for the coupling between L4S and Classic congestion signals in a DualQ Coupled AQM [I-D.ietf-tsvwg-aqm-dualq-coupled], as below.

Unless an AQM node schedules application flows explicitly, the likelihood that the AQM drops a Not-ECT Classic packet ( $p_C$ ) MUST be roughly proportional to the square of the likelihood that it would have marked it if it had been an L4S packet ( $p_L$ ). That is

$$p_C \sim (p_L / k)^2$$

The constant of proportionality ( $k$ ) does not have to be standardised for interoperability, but a value of 2 is RECOMMENDED. The term 'likelihood' is used above to allow for marking and dropping to be either probabilistic or deterministic.

This formula ensures that Scalable and Classic flows will converge to roughly equal congestion windows, for the worst case of Reno congestion control. This is because the congestion windows of Scalable and Classic congestion controls are inversely proportional to  $p_L$  and  $\sqrt{p_C}$  respectively. So squaring  $p_C$  in the above formula counterbalances the square root that characterizes Reno-friendly flows.

Note that, contrary to RFC 3168, an AQM implementing the L4S and Classic treatments does not mark an ECT(1) packet under the same conditions that it would have dropped a Not-ECT packet, as allowed by [RFC8311], which updates RFC 3168. However, if it marks ECT(0) packets, it does so under the same conditions that it would have dropped a Not-ECT packet [RFC3168].

Also, In the L4S architecture [I-D.ietf-tsvwg-l4s-arch], the sender, not the network, is responsible for smoothing out variations in the queue. So, an L4S AQM MUST signal congestion as soon as possible. Then, an L4S sender generally interprets CE marking as an unsmoothed signal.

This requirement does not prevent an L4S AQM from mixing in additional congestion signals that are smoothed, such as the signals from a Classic smoothed AQM that are coupled with unsmoothed L4S signals in the coupled DualQ [I-D.ietf-tsvwg-aqm-dualq-coupled]. But only as long as the onset of congestion can be signalled immediately, and can be interpreted by the sender as if it has been signalled immediately, which is important for interoperability

### 5.3. Exception for L4S Packet Identification by Network Nodes with Transport-Layer Awareness

To implement L4S packet classification, a network node does not need to identify transport-layer flows. Nonetheless, if an L4S network node classifies packets by their transport-layer flow ID and their ECN field, and if all the ECT packets in a flow have been ECT(0), the node MAY classify any CE packets in the same flow as if they were Classic ECT(0) packets. In all other cases, a network node MUST classify all CE packets as if they were ECT(1) packets. Examples of such other cases are: i) if no ECT packets have yet been identified in a flow; ii) if it is not desirable for a network node to identify transport-layer flows; or iii) if some ECT packets in a flow have been ECT(1) (this advice will need to be verified as part of L4S experiments).

### 5.4. Interaction of the L4S Identifier with other Identifiers

The examples in this section concern how additional identifiers might complement the L4S identifier to classify packets between class-based queues. Firstly Section 5.4.1 considers two queues, L4S and Classic, as in the Coupled DualQ AQM [I-D.ietf-tsvwg-aqm-dualq-coupled], either alone (Section 5.4.1.1) or within a larger queuing hierarchy (Section 5.4.1.2). Then Section 5.4.2 considers schemes that might combine per-flow 5-tuples with other identifiers.

#### 5.4.1. DualQ Examples of Other Identifiers Complementing L4S Identifiers

##### 5.4.1.1. Inclusion of Additional Traffic with L4S

In a typical case for the public Internet a network element that implements L4S in a shared queue might want to classify some low-rate but unresponsive traffic (e.g. DNS, LDAP, NTP, voice, game sync packets) into the low latency queue to mix with L4S traffic. In this case it would not be appropriate to call the queue an L4S queue, because it is shared by L4S and non-L4S traffic. Instead it will be called the low latency or L queue. The L queue then offers two different treatments:



- \* The L4S treatment, which is a combination of the L4S AQM treatment and a priority scheduling treatment;
- \* The low latency treatment, which is solely the priority scheduling treatment, without ECN-marking by the AQM.

To identify packets for just the scheduling treatment, it would be inappropriate to use the L4S ECT(1) identifier, because such traffic is unresponsive to ECN marking. Examples of relevant non-ECN identifiers are:

- \* address ranges of specific applications or hosts configured to be, or known to be, safe, e.g. hard-coded IoT devices sending low intensity traffic;
- \* certain low data-volume applications or protocols (e.g. ARP, DNS);
- \* specific Diffserv codepoints that indicate traffic with limited burstiness such as the EF (Expedited Forwarding [RFC3246]), Voice-Admit [RFC5865] or proposed NQB (Non-Queue-Building [I-D.ietf-tsvwg-nqb]) service classes or equivalent local-use DSCPs (see [I-D.briscoe-tsvwg-l4s-diffserv]).

In summary, a network element that implements L4S in a shared queue MAY classify additional types of packets into the L queue based on identifiers other than the ECN field, but the types SHOULD be 'safe' to mix with L4S traffic, where 'safe' is explained in Section 5.4.1.1.1.

A packet that carries one of these non-ECN identifiers to classify it into the L queue would not be subject to the L4S ECN marking treatment, unless it also carried an ECT(1) or CE codepoint. The specification of an L4S AQM MUST define the behaviour for packets with unexpected combinations of codepoints, e.g. a non-ECN-based classifier for the L queue, but ECT(0) in the ECN field (for examples see section 2.5.1.1 of the DualQ spec [I-D.ietf-tsvwg-aqm-dualq-coupled]).

For clarity, non-ECN identifiers, such as the examples itemized above, might be used by some network operators who believe they identify non-L4S traffic that would be safe to mix with L4S traffic. They are not alternative ways for a host to indicate that it is sending L4S packets. Only the ECT(1) ECN codepoint indicates to a network element that a host is sending L4S packets (and CE indicates that it could have originated as ECT(1)). Specifically ECT(1) indicates that the host claims its behaviour satisfies the prerequisite transport requirements in Section 4.

In order to include non-L4S packets in the L queue, a network node MUST NOT alter Not-ECT or ECT(0) in the IP-ECN field to an L4S identifier. This ensures that these codepoints survive for any potential use later on the network path.

#### 5.4.1.1.1. 'Safe' Unresponsive Traffic

The above section requires unresponsive traffic to be 'safe' to mix with L4S traffic. Ideally this means that the sender never sends any sequence of packets at a rate that exceeds the available capacity of the bottleneck link. However, typically an unresponsive transport does not even know the bottleneck capacity of the path, let alone its available capacity. Nonetheless, an application can be considered safe enough if it paces packets out (not necessarily completely regularly) such that its maximum instantaneous rate from packet to packet stays well below a typical broadband access rate.

This is a vague but useful definition, because many low latency applications of interest, such as DNS, voice, game sync packets, RPC, ACKs, keep-alives, could match this description.

Low rate streams such as voice and game sync packets, might not use continuously adapting ECN-based congestion control, but they ought to at least use a 'circuit-breaker' style of congestion response [RFC8083]. If the volume of traffic from unresponsive applications is high enough to overload the link, this will at least protect the capacity available to responsive applications. However, queuing delay in the L queue will probably rise to that controlled by the Classic (drop-based) AQM. If a network operator considers that such self-restraint is not enough, it might want to police the L queue (see Section 8.2 of the L4S architecture [I-D.ietf-tsvwg-l4s-arch]).

#### 5.4.1.2. Exclusion of Traffic From L4S Treatment

To extend the above example, an operator might want to exclude some traffic from the L4S treatment for a policy reason, e.g. security (traffic from malicious sources) or commercial (e.g. initially the operator may wish to confine the benefits of L4S to business customers).

In this exclusion case, the classifier MUST classify on the relevant locally-used identifiers (e.g. source addresses) before classifying the non-matching traffic on the end-to-end L4S ECN identifier.

A network node MUST NOT alter the end-to-end L4S ECN identifier from L4S to Classic, because an operator decision to exclude certain traffic from L4S treatment is local-only. The end-to-end L4S

identifier then survives for other operators to use, or indeed, they can apply their own policy, independently based on their own choice of locally-used identifiers. This approach also allows any operator to remove its locally-applied exclusions in future, e.g. if it wishes to widen the benefit of the L4S treatment to all its customers.

A network node that supports L4S but excludes certain packets carrying the L4S identifier from L4S treatment **MUST** still apply marking or dropping that is compatible with an L4S congestion response. For instance, it could either drop such packets with the same likelihood as Classic packets or it could ECN-mark them with a likelihood appropriate to L4S traffic (e.g. the coupled probability in a DualQ coupled AQM) but aiming for the Classic delay target. It **MUST NOT** ECN-mark such packets with a Classic marking probability, which could confuse the sender.

#### 5.4.1.3. Generalized Combination of L4S and Other Identifiers

L4S concerns low latency, which it can provide for all traffic without differentiation and without necessarily affecting bandwidth allocation. Diffserv provides for differentiation of both bandwidth and low latency, but its control of latency depends on its control of bandwidth. The two can be combined if a network operator wants to control bandwidth allocation but it also wants to provide low latency - for any amount of traffic within one of these allocations of bandwidth (rather than only providing low latency by limiting bandwidth) [I-D.briscoe-tsvwg-l4s-diffserv].

The DualQ examples so far have been framed in the context of providing the default Best Efforts Per-Hop Behaviour (PHB) using two queues - a Low Latency (L) queue and a Classic (C) Queue. This single DualQ structure is expected to be the most common and useful arrangement. But, more generally, an operator might choose to control bandwidth allocation through a hierarchy of Diffserv PHBs at a node, and to offer one (or more) of these PHBs using a pair of queues for a low latency and a Classic variant of the PHB.

In the first case, if we assume that a network element provides no PHBs except the DualQ, if a packet carries ECT(1) or CE, the network element would classify it for the L4S treatment irrespective of its DSCP. And, if a packet carried (say) the EF DSCP, the network element could classify it into the L queue irrespective of its ECN codepoint. However, where the DualQ is in a hierarchy of other PHBs, the classifier would classify some traffic into other PHBs based on DSCP before classifying between the low latency and Classic queues (based on ECT(1), CE and perhaps also the EF DSCP or other identifiers as in the above example).

[I-D.briscoe-tsvwg-l4s-diffserv] gives a number of examples of such arrangements to address various requirements.

[I-D.briscoe-tsvwg-l4s-diffserv] describes how an operator might use L4S to offer low latency as well as using Diffserv for bandwidth differentiation. It identifies two main types of approach, which can be combined: the operator might split certain Diffserv PHBs between L4S and a corresponding Classic service. Or it might split the L4S and/or the Classic service into multiple Diffserv PHBs. In either of these cases, a packet would have to be classified on its Diffserv and ECN codepoints.

In summary, there are numerous ways in which the L4S ECN identifier (ECT(1) and CE) could be combined with other identifiers to achieve particular objectives. The following categorization articulates those that are valid, but it is not necessarily exhaustive. Those tagged 'Recommended-standard-use' could be set by the sending host or a network. Those tagged 'Local-use' would only be set by a network:

1. Identifiers Complementing the L4S Identifier
  - a. Including More Traffic in the L Queue  
(Could use Recommended-standard-use or Local-use identifiers)
  - b. Excluding Certain Traffic from the L Queue  
(Local-use only)
2. Identifiers to place L4S classification in a PHB Hierarchy  
(Could use Recommended-standard-use or Local-use identifiers)
  - a. PHBs Before L4S ECN Classification
  - b. PHBs After L4S ECN Classification

#### 5.4.2. Per-Flow Queuing Examples of Other Identifiers Complementing L4S Identifiers

At a node with per-flow queueing (e.g. FQ-CoDel [RFC8290]), the L4S identifier could complement the Layer-4 flow ID as a further level of flow granularity (i.e. Not-ECT and ECT(0) queued separately from ECT(1) and CE packets). "Risk of reordering Classic CE packets" in Appendix B discusses the resulting ambiguity if packets originally marked ECT(0) are marked CE by an upstream AQM before they arrive at a node that classifies CE as L4S. It argues that the risk of reordering is vanishingly small and the consequence of such a low level of reordering is minimal.

Alternatively, it could be assumed that it is not in a flow's own interest to mix Classic and L4S identifiers. Then the AQM could use the ECN field to switch itself between a Classic and an L4S AQM behaviour within one per-flow queue. For instance, for ECN-capable packets, the AQM might consist of a simple marking threshold and an L4S ECN identifier might simply select a shallower threshold than a Classic ECN identifier would.

#### 5.5. Limiting Packet Bursts from Links

As well as senders needing to limit packet bursts (Section 4.3), links need to limit the degree of burstiness they introduce. In both cases (senders and links) this is a tradeoff, because batch-handling of packets is done for good reason, e.g. processing efficiency or to make efficient use of medium acquisition delay. Some take the attitude that there is no point reducing burst delay at the sender below that introduced by links (or vice versa). However, delay reduction proceeds by cutting down 'the longest pole in the tent', which turns the spotlight on the next longest, and so on.

This document does not set any quantified requirements for links to limit burst delay, primarily because link technologies are outside the remit of L4S specifications. Nonetheless, the following two subsections outline opportunities for addressing bursty links in the process of L4S implementation and deployment.

##### 5.5.1. Limiting Packet Bursts from Links Fed by an L4S AQM

It would not make sense to implement an L4S AQM that feeds into a particular link technology without also reviewing opportunities to reduce any form of burst delay introduced by that link technology. This would at least limit the bursts that the link would otherwise introduce into the onward traffic, which would cause jumpy feedback to the sender as well as potential extra queuing delay downstream. This document does not presume to even give guidance on an

appropriate target for such burst delay until there is more industry experience of L4S. However, as suggested in Section 4.3 it would not seem necessary to limit bursts lower than roughly 10% of the minimum base RTT expected in the typical deployment scenario (e.g. 250 us burst duration for links within the public Internet).

#### 5.5.2. Limiting Packet Bursts from Links Upstream of an L4S AQM

The initial scope of the L4S experiment is to deploy L4S AQMs at bottlenecks and L4S congestion controls at senders. This is expected to highlight interactions with the most bursty upstream links and lead operators to tune down the burstiness of those links in their network that are configurable, or failing that, to have to compromise on the delay target of some L4S AQMs. It might also require specific redesign work relevant to the most problematic link types. Such knock-on effects of initial L4S deployment would all be part of the learning from the L4S experiment.

The details of such link changes are beyond the scope of the present document. Nonetheless, where L4S technology is being implemented on an outgoing interface of a device, it would make sense to consider opportunities for reducing bursts arriving at other incoming interface(s). For instance, where an L4S AQM is implemented to feed into the upstream WAN interface of a home gateway, there would be opportunities to alter the WiFi profiles sent out of any WiFi interfaces from the same device, in order to mitigate incoming bursts of aggregated WiFi frames from other WiFi stations.

### 6. Behaviour of Tunnels and Encapsulations

#### 6.1. No Change to ECN Tunnels and Encapsulations in General

The L4S identifier is expected to work through and within any tunnel without modification, as long as the tunnel propagates the ECN field in any of the ways that have been defined since the first variant in the year 2001 [RFC3168]. L4S will also work with (but does not rely on) any of the more recent updates to ECN propagation in [RFC4301], [RFC6040] or [I-D.ietf-tsvwg-rfc6040update-shim]. However, it is likely that some tunnels still do not implement ECN propagation at all. In these cases, L4S will work through such tunnels, but within them the outer header of L4S traffic will appear as Classic.

AQMs are typically implemented where an IP-layer buffer feeds into a lower layer, so they are agnostic to link layer encapsulations. Where a bottleneck link is not IP-aware, the L4S identifier is still expected to work within any lower layer encapsulation without modification, as long as it propagates the ECN field as defined for the link technology, for example for MPLS [RFC5129] or

TRILL [I-D.ietf-trill-ecn-support]. In some of these cases, e.g. layer-3 Ethernet switches, the AQM accesses the IP layer header within the outer encapsulation, so again the L4S identifier is expected to work without modification. Nonetheless, the programme to define ECN for other lower layers is still in progress [I-D.ietf-tsvwg-ecn-encap-guidelines].

## 6.2. VPN Behaviour to Avoid Limitations of Anti-Replay

If a mix of L4S and Classic packets is sent into the same security association (SA) of a virtual private network (VPN), and if the VPN egress is employing the optional anti-replay feature, it could inappropriately discard Classic packets (or discard the records in Classic packets) by mistaking their greater queuing delay for a replay attack (see "Dropped Packets for Tunnels with Replay Protection Enabled" in [Heist21] for the potential performance impact). This known problem is common to both IPsec [RFC4301] and DTLS [RFC6347] VPNs, given they use similar anti-replay window mechanisms. The mechanism used can only check for replay within its window, so if the window is smaller than the degree of reordering, it can only assume there might be a replay attack and discard all the packets behind the trailing edge of the window. The specifications of IPsec AH [RFC4302] and ESP [RFC4303] suggest that an implementer scales the size of the anti-replay window with interface speed, and DTLS 1.3 [I-D.ietf-tls-dtls13] says "The receiver SHOULD pick a window large enough to handle any plausible reordering, which depends on the data rate." However, in practice, the size of a VPN's anti-replay window is not always scaled appropriately.

If a VPN carrying traffic participating in the L4S experiment experiences inappropriate replay detection, the foremost remedy would be to ensure that the egress is configured to comply with the above window-sizing requirements.

If an implementation of a VPN egress does not support a sufficiently large anti-replay window, e.g. due to hardware limitations, one of the temporary alternatives listed in order of preference below might be feasible instead:

- \* If the VPN can be configured to classify packets into different SAs indexed by DSCP, apply the appropriate locally defined DSCPs to Classic and L4S packets. The DSCPs could be applied by the network (based on the least significant bit of the ECN field), or by the sending host. Such DSCPs would only need to survive as far as the VPN ingress.
- \* If the above is not possible and it is necessary to use L4S, either of the following might be appropriate as a last resort:

- disable anti-replay protection at the VPN egress, after considering the security implications (optional anti-replay is mandatory in both IPsec and DTLS);
- configure the tunnel ingress not to propagate ECN to the outer, which would lose the benefits of L4S and Classic ECN over the VPN.

Modification to VPN implementations is outside the present scope, which is why this section has so far focused on reconfiguration. Although this document does not define any requirements for VPN implementations, determining whether there is a need for such requirements could be one aspect of L4S experimentation.

## 7. L4S Experiments

This section describes open questions that L4S Experiments ought to focus on. This section also documents outstanding open issues that will need to be investigated as part of L4S experimentation, given they could not be fully resolved during the WG phase. It also lists metrics that will need to be monitored during experiments (summarizing text elsewhere in L4S documents) and finally lists some potential future directions that researchers might wish to investigate.

In addition to this section, the DualQ spec [I-D.ietf-tsvwg-aqm-dualq-coupled] sets operational and management requirements for experiments with DualQ Coupled AQMs; and General operational and management requirements for experiments with L4S congestion controls are given in Section 4 and Section 5 above, e.g. co-existence and scaling requirements, incremental deployment arrangements.

The specification of each scalable congestion control will need to include protocol-specific requirements for configuration and monitoring performance during experiments. Appendix A of the guidelines in [RFC5706] provides a helpful checklist.

### 7.1. Open Questions

L4S experiments would be expected to answer the following questions:

- \* Have all the parts of L4S been deployed, and if so, what proportion of paths support it?
  - What types of L4S AQMs were deployed, e.g. FQ, coupled DualQ, uncoupled DualQ, other? And how prevalent was each?



- Are the signalling patterns emitted by the deployed AQMs in any way different from those expected when the Prague requirements for endpoints were written?
- \* Does use of L4S over the Internet result in significantly improved user experience?
- \* Has L4S enabled novel interactive applications?
- \* Did use of L4S over the Internet result in improvements to the following metrics:
  - queue delay (mean and 99th percentile) under various loads;
  - utilization;
  - starvation / fairness;
  - scaling range of flow rates and RTTs?
- \* How dependent was the performance of L4S service on the bottleneck bandwidth or the path RTT?
- \* How much do bursty links in the Internet affect L4S performance (see "Underutilization with Bursty Links" in [Heist21]) and how prevalent are they? How much limitation of burstiness from upstream links was needed and/or was realized - both at senders and at links, especially radio links or how much did L4S target delay have to be increased to accommodate the bursts (see bullet #7 in Section 4.3 and Section 5.5.2)?
- \* Is the initial experiment with mis-marked bursty traffic at high RTT (see "Underutilization with Bursty Traffic" in [Heist21]) indicative of similar problems at lower RTTs and, if so, how effective is the suggested remedy in Appendix A.1 of the DualQ spec [I-D.ietf-tsvwg-aqm-dualq-coupled] (or possible other remedies)?
- \* Was per-flow queue protection typically (un)necessary?
  - How well did overload protection or queue protection work?
- \* How well did L4S flows coexist with Classic flows when sharing a bottleneck?
  - How frequently did problems arise?

- What caused any coexistence problems, and were any problems due to single-queue Classic ECN AQMs (this assumes single-queue Classic ECN AQMs can be distinguished from FQ ones)?
- \* How prevalent were problems with the L4S service due to tunnels / encapsulations that do not support ECN decapsulation?
- \* How easy was it to implement a fully compliant L4S congestion control, over various different transport protocols (TCP, QUIC, RMCAT, etc)?

Monitoring for harm to other traffic, specifically bandwidth starvation or excess queuing delay, will need to be conducted alongside all early L4S experiments. It is hard, if not impossible, for an individual flow to measure its impact on other traffic. So such monitoring will need to be conducted using bespoke monitoring across flows and/or across classes of traffic.

## 7.2. Open Issues

- \* What is the best way forward to deal with L4S over single-queue Classic ECN AQM bottlenecks, given current problems with misdetecting L4S AQMs as Classic ECN AQMs? See the L4S operational guidance [I-D.ietf-tsvwg-l4sops].
- \* Fixing the poor Interaction between current L4S congestion controls and CoDel with only Classic ECN support during flow startup. Originally, this was due to a bug in the initialization of the congestion EWMA in the Linux implementation of TCP Prague. That was quickly fixed, which removed the main performance impact, but further improvement would be useful (either by modifying CoDel, Scalable congestion controls, or both).

## 7.3. Future Potential

Researchers might find that L4S opens up the following interesting areas for investigation:

- \* Potential for faster convergence time and tracking of available capacity;
- \* Potential for improvements to particular link technologies, and cross-layer interactions with them;
- \* Potential for using virtual queues, e.g. to further reduce latency jitter, or to leave headroom for capacity variation in radio networks;

- \* Development and specification of reverse path congestion control using L4S building blocks (e.g. AccECN, QUIC);
- \* Once queuing delay is cut down, what becomes the 'second longest pole in the tent' (other than the speed of light)?
- \* Novel alternatives to the existing set of L4S AQMs;
- \* Novel applications enabled by L4S.

## 8. IANA Considerations

The 01 codepoint of the ECN Field of the IP header is specified by the present Experimental RFC. The process for an experimental RFC to assign this codepoint in the IP header (v4 and v6) is documented in Proposed Standard [RFC8311], which updates the Proposed Standard [RFC3168].

When the present document is published as an RFC, IANA is asked to update the 01 entry in the registry, "ECN Field (Bits 6-7)" to the following (see <https://www.iana.org/assignments/dscp-registry/dscp-registry.xhtml#ecn-field>):

Binary	Keyword	References
01	ECT(1) (ECN-Capable Transport (1)) [1]	[RFC8311] [RFC Errata 5399] [RFCXXXX]

Table 1

[XXXX is the number that the RFC Editor assigns to the present document (this sentence to be removed by the RFC Editor)].

## 9. Security Considerations

Approaches to assure the integrity of signals using the new identifier are introduced in Appendix C.1. See the security considerations in the L4S architecture [I-D.ietf-tsvwg-l4s-arch] for further discussion of mis-use of the identifier, as well as extensive discussion of policing rate and latency in regard to L4S.

If the anti-replay window of a VPN egress is too small, it will mistake deliberate delay differences as a replay attack, and discard higher delay packets (e.g. Classic) carried within the same security association (SA) as low delay packets (e.g. L4S). Section 6.2 recommends that VPNs used in L4S experiments are configured with a sufficiently large anti-replay window, as required by the relevant specifications. It also discusses other alternatives.

If a user taking part in the L4S experiment sets up a VPN without being aware of the above advice, and if the user allows anyone to send traffic into their VPN, they would open up a DoS vulnerability in which an attacker could induce the VPN's anti-replay mechanism to discard enough of the user's Classic (C) traffic (if they are receiving any) to cause a significant rate reduction. While the user is actively downloading C traffic, the attacker sends C traffic into the VPN to fill the remainder of the bottleneck link, then sends intermittent L4S packets to maximize the chance of exceeding the VPN's replay window. The user can prevent this attack by following the recommendations in Section 6.2.

The recommendation to detect loss in time units prevents the ACK-splitting attacks described in [Savage-TCP].

## 10. Acknowledgements

Thanks to Richard Scheffenegger, John Leslie, David Taeht, Jonathan Morton, Gorrry Fairhurst, Michael Welzl, Mikael Abrahamsson and Andrew McGregor for the discussions that led to this specification. Ing-jyh (Inton) Tsang was a contributor to the early drafts of this document. And thanks to Mikael Abrahamsson, Lloyd Wood, Nicolas Kuhn, Greg White, Tom Henderson, David Black, Gorrry Fairhurst, Brian Carpenter, Jake Holland, Rod Grimes, Richard Scheffenegger, Sebastian Moeller, Neal Cardwell, Praveen Balasubramanian, Reza Marandian Hagh, Pete Heist, Stuart Cheshire, Vidhi Goel, Mirja Kuehlewind and Ermin Sakic for providing help and reviewing this draft and thanks to Ingemar Johansson for reviewing and providing substantial text. Thanks to Sebastian Moeller for identifying the interaction with VPN anti-replay and to Jonathan Morton for identifying the attack based on this. Particular thanks to tsvwg chairs Gorrry Fairhurst, David Black and Wes Eddy for patiently helping this and the other L4S drafts through the IETF process. Appendix A listing the Prague L4S Requirements is based on text authored by Marcelo Bagnulo Braun that was originally an appendix to [I-D.ietf-tsvwg-l4s-arch]. That text was in turn based on the collective output of the attendees listed in the minutes of a 'bar BoF' on DCTCP Evolution during IETF-94 [TCPPrague].

The authors' contributions were part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The contribution of Koen De Schepper was also part-funded by the 5Growth and DAEMON EU H2020 projects. Bob Briscoe was also funded partly by the Research Council of Norway through the TimeIn project, partly by CableLabs and partly by the Comcast Innovation Fund. The views expressed here are solely those of the authors.

## 11. References

### 11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC4774] Floyd, S., "Specifying Alternate Semantics for the Explicit Congestion Notification (ECN) Field", BCP 124, RFC 4774, DOI 10.17487/RFC4774, November 2006, <<https://www.rfc-editor.org/info/rfc4774>>.
- [RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", RFC 6679, DOI 10.17487/RFC6679, August 2012, <<https://www.rfc-editor.org/info/rfc6679>>.

### 11.2. Informative References

- [A2DTCP] Zhang, T., Wang, J., Huang, J., Huang, Y., Chen, J., and Y. Pan, "Adaptive-Acceleration Data Center TCP", IEEE Transactions on Computers 64(6):1522-1533, June 2015, <<http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6871352>>.
- [Ahmed19] Ahmed, A.S., "Extending TCP for Low Round Trip Delay", Masters Thesis, Uni Oslo, August 2019, <<https://www.duo.uio.no/handle/10852/70966>>.
- [Alizadeh-stability] Alizadeh, M., Javanmard, A., and B. Prabhakar, "Analysis of DCTCP: Stability, Convergence, and Fairness", ACM

SIGMETRICS 2011 , June 2011,  
<[https://people.csail.mit.edu/alizadeh/papers/  
dctcp\\_analysis-sigmetrics11.pdf](https://people.csail.mit.edu/alizadeh/papers/dctcp_analysis-sigmetrics11.pdf)>.

- [ARED01] Floyd, S., Gummadi, R., and S. Shenker, "Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management", ACIRI Technical Report , August 2001, <<http://www.icir.org/floyd/red.html>>.
- [BBRv2] Cardwell, N., "BRTCP BBR v2 Alpha/Preview Release", github repository; Linux congestion control module, <<https://github.com/google/bbr/blob/v2alpha/README.md>>.
- [COBALT] Palmei, J., Gupta, S., Imputato, P., Morton, J., Tahiliani, M., Avallone, S., and D. Taht, "Design and Evaluation of COBALT Queue Discipline", In Proc. IEEE Int'l Symp. on Local and Metropolitan Area Networks 2019, ppl--6, 2019, <<https://doi.org/10.1109/LANMAN.2019.8847054>>.
- [DCttH19] De Schepper, K., Bondarenko, O., Tilmans, O., and B. Briscoe, "'Data Centre to the Home': Ultra-Low Latency for All", Updated RITE project Technical Report , July 2019, <[https://bobbriscoe.net/pubs.html#DCttH\\_TR](https://bobbriscoe.net/pubs.html#DCttH_TR)>.
- [DualPI2Linux] Albisser, O., De Schepper, K., Briscoe, B., Tilmans, O., and H. Steen, "DUALPI2 - Low Latency, Low Loss and Scalable (L4S) AQM", Proc. Linux Netdev 0x13 , March 2019, <<https://www.netdevconf.org/0x13/session.html?talk-DUALPI2-AQM>>.
- [ecn-fallback] Briscoe, B. and A.S. Ahmed, "TCP Prague Fall-back on Detection of a Classic ECN AQM", bobbriscoe.net Technical Report TR-BB-2019-002, April 2020, <<https://arxiv.org/abs/1911.00710>>.
- [Heist21] Heist, P. and J. Morton, "L4S Tests", github README, May 2021, <<https://github.com/heistp/l4s-tests/>>.
- [I-D.briscoe-docsis-q-protection] Briscoe, B. and G. White, "The DOCSIS(r) Queue Protection Algorithm to Preserve Low Latency", Work in Progress, Internet-Draft, draft-briscoe-docsis-q-protection-02, 31 January 2022, <<https://datatracker.ietf.org/doc/html/draft-briscoe-docsis-q-protection-02>>.

- [I-D.briscoe-iccrp-prague-congestion-control]  
Schepper, K. D., Tilmans, O., and B. Briscoe, "Prague Congestion Control", Work in Progress, Internet-Draft, draft-briscoe-iccrp-prague-congestion-control-00, 9 March 2021, <<https://datatracker.ietf.org/doc/html/draft-briscoe-iccrp-prague-congestion-control-00>>.
- [I-D.briscoe-tsvwg-l4s-diffserv]  
Briscoe, B., "Interactions between Low Latency, Low Loss, Scalable Throughput (L4S) and Differentiated Services", Work in Progress, Internet-Draft, draft-briscoe-tsvwg-l4s-diffserv-02, 4 November 2018, <<https://datatracker.ietf.org/doc/html/draft-briscoe-tsvwg-l4s-diffserv-02>>.
- [I-D.cardwell-iccrp-bbr-congestion-control]  
Cardwell, N., Cheng, Y., Yeganeh, S. H., Swett, I., and V. Jacobson, "BBR Congestion Control", Work in Progress, Internet-Draft, draft-cardwell-iccrp-bbr-congestion-control-01, 7 November 2021, <<https://datatracker.ietf.org/doc/html/draft-cardwell-iccrp-bbr-congestion-control-01>>.
- [I-D.ietf-tcpm-accurate-ecn]  
Briscoe, B., Kühlewind, M., and R. Scheffenegger, "More Accurate ECN Feedback in TCP", Work in Progress, Internet-Draft, draft-ietf-tcpm-accurate-ecn-16, 3 February 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-accurate-ecn-16>>.
- [I-D.ietf-tcpm-generalized-ecn]  
Bagnulo, M. and B. Briscoe, "ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control Packets", Work in Progress, Internet-Draft, draft-ietf-tcpm-generalized-ecn-09, 31 January 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-generalized-ecn-09>>.
- [I-D.ietf-tls-dtls13]  
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-43, 30 April 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-tls-dtls13-43>>.

[I-D.ietf-trill-ecn-support]

Eastlake, D. E. and B. Briscoe, "TRILL (TRansparent Interconnection of Lots of Links): ECN (Explicit Congestion Notification) Support", Work in Progress, Internet-Draft, draft-ietf-trill-ecn-support-07, 25 February 2018, <<https://datatracker.ietf.org/doc/html/draft-ietf-trill-ecn-support-07>>.

[I-D.ietf-tsvwg-aqm-dualq-coupled]

Schepper, K. D., Briscoe, B., and G. White, "DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)", Work in Progress, Internet-Draft, draft-ietf-tsvwg-aqm-dualq-coupled-22, 4 March 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-aqm-dualq-coupled-22>>.

[I-D.ietf-tsvwg-ecn-encap-guidelines]

Briscoe, B. and J. Kaippallimalil, "Guidelines for Adding Congestion Notification to Protocols that Encapsulate IP", Work in Progress, Internet-Draft, draft-ietf-tsvwg-ecn-encap-guidelines-16, 25 May 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-ecn-encap-guidelines-16>>.

[I-D.ietf-tsvwg-l4s-arch]

Briscoe, B., Schepper, K. D., Bagnulo, M., and G. White, "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture", Work in Progress, Internet-Draft, draft-ietf-tsvwg-l4s-arch-16, 1 February 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-l4s-arch-16>>.

[I-D.ietf-tsvwg-l4sops]

White, G., "Operational Guidance for Deployment of L4S in the Internet", Work in Progress, Internet-Draft, draft-ietf-tsvwg-l4sops-02, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-l4sops-02>>.

[I-D.ietf-tsvwg-nqb]

White, G. and T. Fossati, "A Non-Queue-Building Per-Hop Behavior (NQB PHB) for Differentiated Services", Work in Progress, Internet-Draft, draft-ietf-tsvwg-nqb-10, 4 March 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-nqb-10>>.



- [I-D.ietf-tsvwg-rfc6040update-shim]  
Briscoe, B., "Propagating Explicit Congestion Notification Across IP Tunnel Headers Separated by a Shim", Work in Progress, Internet-Draft, draft-ietf-tsvwg-rfc6040update-shim-14, 25 May 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-rfc6040update-shim-14>>.
- [I-D.sridharan-tcpm-ctcp]  
Sridharan, M., Tan, K., Bansal, D., and D. Thaler, "Compound TCP: A New TCP Congestion Control for High-Speed and Long Distance Networks", Work in Progress, Internet-Draft, draft-sridharan-tcpm-ctcp-02, 11 November 2008, <<https://datatracker.ietf.org/doc/html/draft-sridharan-tcpm-ctcp-02>>.
- [I-D.stewart-tsvwg-sctp-ecn]  
Stewart, R. R., Tuexen, M., and X. Dong, "ECN for Stream Control Transmission Protocol (SCTP)", Work in Progress, Internet-Draft, draft-stewart-tsvwg-sctp-ecn-05, 15 January 2014, <<https://datatracker.ietf.org/doc/html/draft-stewart-tsvwg-sctp-ecn-05>>.
- [LinuxPacedChirping]  
Misund, J. and B. Briscoe, "Paced Chirping - Rethinking TCP start-up", Proc. Linux Netdev 0x13 , March 2019, <<https://www.netdevconf.org/0x13/session.html?talk-chirp>>.
- [Mathis09] Mathis, M., "Relentless Congestion Control", PFLDNet'09 , May 2009, <[http://www.hpcc.jp/pfldnet2009/Program\\_files/1569198525.pdf](http://www.hpcc.jp/pfldnet2009/Program_files/1569198525.pdf)>.
- [Paced-Chirping]  
Misund, J., "Rapid Acceleration in TCP Prague", Masters Thesis , May 2018, <<https://riteproject.files.wordpress.com/2018/07/misundjoakimmastersthesissubmitted180515.pdf>>.
- [PI2] De Schepper, K., Bondarenko, O., Tsang, I., and B. Briscoe, "PI<sup>2</sup> : A Linearized AQM for both Classic and Scalable TCP", Proc. ACM CoNEXT 2016 pp.105-119, December 2016, <<http://dl.acm.org/citation.cfm?doid=2999572.2999578>>.
- [PragueLinux]  
Briscoe, B., De Schepper, K., Albisser, O., Misund, J., Tilmans, O., Kühlewind, M., and A.S. Ahmed, "Implementing the 'TCP Prague' Requirements for Low Latency Low Loss

Scalable Throughput (L4S)", Proc. Linux Netdev 0x13 ,  
March 2019, <[https://www.netdevconf.org/0x13/  
session.html?talk-tcp-prague-l4s](https://www.netdevconf.org/0x13/session.html?talk-tcp-prague-l4s)>.

- [QV] Briscoe, B. and P. Hurtig, "Up to Speed with Queue View",  
RITE Technical Report D2.3; Appendix C.2, August 2015,  
<[https://riteproject.files.wordpress.com/2015/12/rite-  
deliverable-2-3.pdf](https://riteproject.files.wordpress.com/2015/12/rite-deliverable-2-3.pdf)>.
- [RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering,  
S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G.,  
Partridge, C., Peterson, L., Ramakrishnan, K., Shenker,  
S., Wroclawski, J., and L. Zhang, "Recommendations on  
Queue Management and Congestion Avoidance in the  
Internet", RFC 2309, DOI 10.17487/RFC2309, April 1998,  
<<https://www.rfc-editor.org/info/rfc2309>>.
- [RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black,  
"Definition of the Differentiated Services Field (DS  
Field) in the IPv4 and IPv6 Headers", RFC 2474,  
DOI 10.17487/RFC2474, December 1998,  
<<https://www.rfc-editor.org/info/rfc2474>>.
- [RFC3246] Davie, B., Charny, A., Bennet, J.C.R., Benson, K., Le  
Boudec, J.Y., Courtney, W., Davari, S., Firoiu, V., and D.  
Stiliadis, "An Expedited Forwarding PHB (Per-Hop  
Behavior)", RFC 3246, DOI 10.17487/RFC3246, March 2002,  
<<https://www.rfc-editor.org/info/rfc3246>>.
- [RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit  
Congestion Notification (ECN) Signaling with Nonces",  
RFC 3540, DOI 10.17487/RFC3540, June 2003,  
<<https://www.rfc-editor.org/info/rfc3540>>.
- [RFC3649] Floyd, S., "HighSpeed TCP for Large Congestion Windows",  
RFC 3649, DOI 10.17487/RFC3649, December 2003,  
<<https://www.rfc-editor.org/info/rfc3649>>.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the  
Internet Protocol", RFC 4301, DOI 10.17487/RFC4301,  
December 2005, <<https://www.rfc-editor.org/info/rfc4301>>.
- [RFC4302] Kent, S., "IP Authentication Header", RFC 4302,  
DOI 10.17487/RFC4302, December 2005,  
<<https://www.rfc-editor.org/info/rfc4302>>.

- [RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <<https://www.rfc-editor.org/info/rfc4303>>.
- [RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI 10.17487/RFC4340, March 2006, <<https://www.rfc-editor.org/info/rfc4340>>.
- [RFC4341] Floyd, S. and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control", RFC 4341, DOI 10.17487/RFC4341, March 2006, <<https://www.rfc-editor.org/info/rfc4341>>.
- [RFC4342] Floyd, S., Kohler, E., and J. Padhye, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC)", RFC 4342, DOI 10.17487/RFC4342, March 2006, <<https://www.rfc-editor.org/info/rfc4342>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.
- [RFC5033] Floyd, S. and M. Allman, "Specifying New Congestion Control Algorithms", BCP 133, RFC 5033, DOI 10.17487/RFC5033, August 2007, <<https://www.rfc-editor.org/info/rfc5033>>.
- [RFC5129] Davie, B., Briscoe, B., and J. Tay, "Explicit Congestion Marking in MPLS", RFC 5129, DOI 10.17487/RFC5129, January 2008, <<https://www.rfc-editor.org/info/rfc5129>>.
- [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <<https://www.rfc-editor.org/info/rfc5348>>.
- [RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", RFC 5562, DOI 10.17487/RFC5562, June 2009, <<https://www.rfc-editor.org/info/rfc5562>>.

- [RFC5622] Floyd, S. and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion ID 4: TCP-Friendly Rate Control for Small Packets (TFRC-SP)", RFC 5622, DOI 10.17487/RFC5622, August 2009, <<https://www.rfc-editor.org/info/rfc5622>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/info/rfc5681>>.
- [RFC5706] Harrington, D., "Guidelines for Considering Operations and Management of New Protocols and Protocol Extensions", RFC 5706, DOI 10.17487/RFC5706, November 2009, <<https://www.rfc-editor.org/info/rfc5706>>.
- [RFC5865] Baker, F., Polk, J., and M. Dolly, "A Differentiated Services Code Point (DSCP) for Capacity-Admitted Traffic", RFC 5865, DOI 10.17487/RFC5865, May 2010, <<https://www.rfc-editor.org/info/rfc5865>>.
- [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <<https://www.rfc-editor.org/info/rfc5925>>.
- [RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <<https://www.rfc-editor.org/info/rfc6040>>.
- [RFC6077] Papadimitriou, D., Ed., Welzl, M., Scharf, M., and B. Briscoe, "Open Research Issues in Internet Congestion Control", RFC 6077, DOI 10.17487/RFC6077, February 2011, <<https://www.rfc-editor.org/info/rfc6077>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6660] Briscoe, B., Moncaster, T., and M. Menth, "Encoding Three Pre-Congestion Notification (PCN) States in the IP Header Using a Single Diffserv Codepoint (DSCP)", RFC 6660, DOI 10.17487/RFC6660, July 2012, <<https://www.rfc-editor.org/info/rfc6660>>.
- [RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, DOI 10.17487/RFC6675, August 2012, <<https://www.rfc-editor.org/info/rfc6675>>.

- [RFC7560] Kuehlewind, M., Ed., Scheffenegger, R., and B. Briscoe, "Problem Statement and Requirements for Increased Accuracy in Explicit Congestion Notification (ECN) Feedback", RFC 7560, DOI 10.17487/RFC7560, August 2015, <<https://www.rfc-editor.org/info/rfc7560>>.
- [RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <<https://www.rfc-editor.org/info/rfc7567>>.
- [RFC7713] Mathis, M. and B. Briscoe, "Congestion Exposure (ConEx) Concepts, Abstract Mechanism, and Requirements", RFC 7713, DOI 10.17487/RFC7713, December 2015, <<https://www.rfc-editor.org/info/rfc7713>>.
- [RFC8033] Pan, R., Natarajan, P., Baker, F., and G. White, "Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem", RFC 8033, DOI 10.17487/RFC8033, February 2017, <<https://www.rfc-editor.org/info/rfc8033>>.
- [RFC8083] Perkins, C. and V. Singh, "Multimedia Congestion Control: Circuit Breakers for Unicast RTP Sessions", RFC 8083, DOI 10.17487/RFC8083, March 2017, <<https://www.rfc-editor.org/info/rfc8083>>.
- [RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <<https://www.rfc-editor.org/info/rfc8085>>.
- [RFC8257] Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <<https://www.rfc-editor.org/info/rfc8257>>.
- [RFC8290] Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm", RFC 8290, DOI 10.17487/RFC8290, January 2018, <<https://www.rfc-editor.org/info/rfc8290>>.
- [RFC8298] Johansson, I. and Z. Sarker, "Self-Clocked Rate Adaptation for Multimedia", RFC 8298, DOI 10.17487/RFC8298, December 2017, <<https://www.rfc-editor.org/info/rfc8298>>.

- [RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <<https://www.rfc-editor.org/info/rfc8311>>.
- [RFC8312] Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", RFC 8312, DOI 10.17487/RFC8312, February 2018, <<https://www.rfc-editor.org/info/rfc8312>>.
- [RFC8511] Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst, "TCP Alternative Backoff with ECN (ABE)", RFC 8511, DOI 10.17487/RFC8511, December 2018, <<https://www.rfc-editor.org/info/rfc8511>>.
- [RFC8888] Sarker, Z., Perkins, C., Singh, V., and M. Ramalho, "RTP Control Protocol (RTCP) Feedback for Congestion Control", RFC 8888, DOI 10.17487/RFC8888, January 2021, <<https://www.rfc-editor.org/info/rfc8888>>.
- [RFC8985] Cheng, Y., Cardwell, N., Dukkkipati, N., and P. Jha, "The RACK-TLP Loss Detection Algorithm for TCP", RFC 8985, DOI 10.17487/RFC8985, February 2021, <<https://www.rfc-editor.org/info/rfc8985>>.
- [RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <<https://www.rfc-editor.org/info/rfc9000>>.
- [Savage-TCP] Savage, S., Cardwell, N., Wetherall, D., and T. Anderson, "TCP Congestion Control with a Misbehaving Receiver", ACM SIGCOMM Computer Communication Review 29(5):71--78, October 1999.
- [SCReAM] Johansson, I., "SCReAM", github repository; , <<https://github.com/EricssonResearch/scream/blob/master/README.md>>.
- [sub-mss-prob] Briscoe, B. and K. De Schepper, "Scaling TCP's Congestion Window for Small Round Trip Times", BT Technical Report TR-TUB8-2015-002, May 2015, <<https://arxiv.org/abs/1904.07598>>.

- [TCP-CA] Jacobson, V. and M.J. Karels, "Congestion Avoidance and Control", Laurence Berkeley Labs Technical Report , November 1988, <<http://ee.lbl.gov/papers/congavoid.pdf>>.
- [TCPPrague] Briscoe, B., "Notes: DCTCP evolution 'bar BoF': Tue 21 Jul 2015, 17:40, Prague", tcpprague mailing list archive , July 2015, <<https://www.ietf.org/mail-archive/web/tcpprague/current/msg00001.html>>.
- [VCP] Xia, Y., Subramanian, L., Stoica, I., and S. Kalyanaraman, "One more bit is enough", Proc. SIGCOMM'05, ACM CCR 35(4)37--48, 2005, <<http://doi.acm.org/10.1145/1080091.1080098>>.

#### Appendix A. Rationale for the 'Prague L4S Requirements'

This appendix is informative, not normative. It gives a list of modifications to current scalable congestion controls so that they can be deployed over the public Internet and coexist safely with existing traffic. The list complements the normative requirements in Section 4 that a sender has to comply with before it can set the L4S identifier in packets it sends into the Internet. As well as rationale for safety improvements (the requirements in Section 4) this appendix also includes preferable performance improvements (optimizations).

The requirements and recommendations in Section 4) have become known as the Prague L4S Requirements, because they were originally identified at an ad hoc meeting during IETF-94 in Prague [TCPPrague]. They were originally called the 'TCP Prague Requirements', but they are not solely applicable to TCP, so the name and wording has been generalized for all transport protocols, and the name 'TCP Prague' is now used for a specific implementation of the requirements.

At the time of writing, DCTCP [RFC8257] is the most widely used scalable transport protocol. In its current form, DCTCP is specified to be deployable only in controlled environments. Deploying it in the public Internet would lead to a number of issues, both from the safety and the performance perspective. The modifications and additional mechanisms listed in this section will be necessary for its deployment over the global Internet. Where an example is needed, DCTCP is used as a base, but the requirements in Section 4 apply equally to other scalable congestion controls, covering adaptive real-time media, etc., not just capacity-seeking behaviours.

## A.1. Rationale for the Requirements for Scalable Transport Protocols

### A.1.1. Use of L4S Packet Identifier

Description: A scalable congestion control needs to distinguish the packets it sends from those sent by Classic congestion controls (see the precise normative requirement wording in Section 4.1).

Motivation: It needs to be possible for a network node to classify L4S packets without flow state into a queue that applies an L4S ECN marking behaviour and isolates L4S packets from the queuing delay of Classic packets.

### A.1.2. Accurate ECN Feedback

Description: The transport protocol for a scalable congestion control needs to provide timely, accurate feedback about the extent of ECN marking experienced by all packets (see the precise normative requirement wording in Section 4.2).

Motivation: Classic congestion controls only need feedback about the existence of a congestion episode within a round trip, not precisely how many packets were marked with ECN or dropped. Therefore, in 2001, when ECN feedback was added to TCP [RFC3168], it could not inform the sender of more than one ECN mark per RTT. Since then, requirements for more accurate ECN feedback in TCP have been defined in [RFC7560] and [I-D.ietf-tcpm-accurate-ecn] specifies a change to the TCP protocol to satisfy these requirements. Most other transport protocols already satisfy this requirement (see Section 4.2).

### A.1.3. Capable of Replacement by Classic Congestion Control

Description: It needs to be possible to replace the implementation of a scalable congestion control with a Classic control (see the precise normative requirement wording in Section 4.3).

Motivation: L4S is an experimental protocol, therefore it seems prudent to be able to disable it at source in case of insurmountable problems, perhaps due to some unexpected interaction on a particular sender; over a particular path or network; with a particular receiver or even ultimately an insurmountable problem with the experiment as a whole.



## A.1.4. Fall back to Classic Congestion Control on Packet Loss

Description: As well as responding to ECN markings in a scalable way, a scalable congestion control needs to react to packet loss in a way that will coexist safely with a Reno congestion control [RFC5681] (see the precise normative requirement wording in Section 4.3).

Motivation: Part of the safety conditions for deploying a scalable congestion control on the public Internet is to make sure that it behaves properly when it builds a queue at a network bottleneck that has not been upgraded to support L4S. Packet loss can have many causes, but it usually has to be conservatively assumed that it is a sign of congestion. Therefore, on detecting packet loss, a scalable congestion control will need to fall back to Classic congestion control behaviour. If it does not comply, it could starve Classic traffic.

A scalable congestion control can be used for different types of transport, e.g. for real-time media or for reliable transport like TCP. Therefore, the particular Classic congestion control behaviour to fall back on will need to be dependent on the specific congestion control implementation. In the particular case of DCTCP, the DCTCP specification [RFC8257] states that "It is RECOMMENDED that an implementation deal with loss episodes in the same way as conventional TCP." For safe deployment, Section 4.3 requires any specification of a scalable congestion control for the public Internet to define the above requirement as a "MUST".

Even though a bottleneck is L4S capable, it might still become overloaded and have to drop packets. In this case, the sender may receive a high proportion of packets marked with the CE bit set and also experience loss. Current DCTCP implementations each react differently to this situation. At least one implementation reacts only to the drop signal (e.g. by halving the CWND) and at least another DCTCP implementation reacts to both signals (e.g. by halving the CWND due to the drop and also further reducing the CWND based on the proportion of marked packet). A third approach for the public Internet has been proposed that adjusts the loss response to result in a halving when combined with the ECN response. We believe that further experimentation is needed to understand what is the best behaviour for the public Internet, which may or not be one of these existing approaches.

#### A.1.5. Coexistence with Classic Congestion Control at Classic ECN bottlenecks

Description: Monitoring has to be in place so that a non-L4S but ECN-capable AQM can be detected at path bottlenecks. This is in case such an AQM has been implemented in a shared queue, in which case any long-running scalable flow would predominate over any simultaneous long-running Classic flow sharing the queue. The precise requirement wording in Section 4.3 is written so that such a problem could either be resolved in real-time, or via administrative intervention.

Motivation: Similarly to the discussion in Appendix A.1.4, this requirement in Section 4.3 is a safety condition to ensure an L4S congestion control coexists well with Classic flows when it builds a queue at a shared network bottleneck that has not been upgraded to support L4S. Nonetheless, if necessary, it is considered reasonable to resolve such problems over management timescales (possibly involving human intervention) because:

- \* although a Classic flow can considerably reduce its throughput in the face of a competing scalable flow, it still makes progress and does not starve;
- \* implementations of a Classic ECN AQM in a queue that is intended to be shared are believed to be rare;
- \* detection of such AQMs is not always clear-cut; so focused out-of-band testing (or even contacting the relevant network operator) would improve certainty.

Therefore, the relevant normative requirement (Section 4.3) is divided into three stages: monitoring, detection and action:

Monitoring: Monitoring involves collection of the measurement data to be analysed. Monitoring is expressed as a 'MUST' for uncontrolled environments, although the placement of the monitoring function is left open. Whether monitoring has to be applied in real-time is expressed as a 'SHOULD'. This allows for the possibility that the operator of an L4S sender (e.g. a CDN) might prefer to test out-of-band for signs of Classic ECN AQMs, perhaps to avoid continually consuming resources to monitor live traffic.

Detection: Detection involves analysis of the monitored data to detect the likelihood of a Classic ECN AQM. Detection can either directly detect actual coexistence problems between flows, or it can aim to identify AQM technologies that are likely to present coexistence problems, based on knowledge of AQMs deployed at the

time. The requirements recommend that detection occurs live in real-time. However, detection is allowed to be deferred (e.g. it might involve further testing targeted at candidate AQMs);

Action: This involves the act of switching the sender to a Classic congestion control. This might occur in real-time within the congestion control for the subsequent duration of a flow, or it might involve administrative action to switch to Classic congestion control for a specific interface or for a certain set of destination addresses.

Instead of the sender taking action itself, the operator of the sender (e.g. a CDN) might prefer to ask the network operator to modify the Classic AQM's treatment of L4S packets; or to ensure L4S packets bypass the AQM; or to upgrade the AQM to support L4S (see the L4S operational guidance [I-D.ietf-tsvwg-l4sops]). Once L4S flows no longer shared the Classic ECN AQM they would obviously no longer detect it, and the requirement to act on it would no longer apply.

The whole set of normative requirements concerning Classic ECN AQMs in Section 4.3 is worded so that it does not apply in controlled environments, such as private networks or data centre networks. CDN servers placed within an access ISP's network can be considered as a single controlled environment, but any onward networks served by the access network, including all the attached customer networks, would be unlikely to fall under the same degree of coordinated control. Monitoring is expressed as a 'MUST' for these uncontrolled segments of paths (e.g. beyond the access ISP in a home network), because there is a possibility that there might be a shared queue Classic ECN AQM in that segment. Nonetheless, the intent of the wording is to only require occasional monitoring of these uncontrolled regions, and not to burden CDN operators if monitoring never uncovers any potential problems.

More detailed discussion of all the above options and alternatives can be found in the L4S operational guidance [I-D.ietf-tsvwg-l4sops].

Having said all the above, the approach recommended in Section 4.3 is to monitor, detect and act in real-time on live traffic. A passive monitoring algorithm to detect a Classic ECN AQM at the bottleneck and fall back to Classic congestion control is described in an extensive technical report [ecn-fallback], which also provides a link to Linux source code, and a large online visualization of its evaluation results. Very briefly, the algorithm primarily monitors RTT variation using the same algorithm that maintains the mean deviation of TCP's smoothed RTT, but it smooths over a duration of the order of a Classic sawtooth. The outcome is also conditioned on

other metrics such as the presence of CE marking and congestion avoidance phase having stabilized. The report also identifies further work to improve the approach, for instance improvements with low capacity links and combining the measurements with a cache of what had been learned about a path in previous connections. The report also suggests alternative approaches.

Although using passive measurements within live traffic (as above) can detect a Classic ECN AQM, it is much harder (perhaps impossible) to determine whether or not the AQM is in a shared queue. Nonetheless, this is much easier using active test traffic out-of-band, because two flows can be used. Section 4 of the same report [ecn-fallback] describes a simple technique to detect a Classic ECN AQM and determine whether it is in a shared queue, summarized here.

An L4S-enabled test server could be set up so that, when a test client accesses it, it serves a script that gets the client to open two parallel long-running flows. It could serve one with a Classic congestion control (C, that sets ECT(0)) and one with a scalable CC (L, that sets ECT(1)). If neither flow induces any ECN marks, it can be presumed the path does not contain a Classic ECN AQM. If either flow induces some ECN marks, the server could measure the relative flow rates and round trip times of the two flows. Table 2 shows the AQM that can be inferred for various cases (presuming the AQM behaviours known at the time of writing).

Rate	RTT	Inferred AQM
L > C	L = C	Classic ECN AQM (FIFO)
L = C	L = C	Classic ECN AQM (FQ)
L = C	L < C	FQ-L4S AQM
L ~ C	L < C	Coupled DualQ AQM

Table 2: Out-of-band testing with two parallel flows. L:=L4S, C:=Classic.

Finally, we motivate the recommendation in Section 4.3 that a scalable congestion control is not expected to change to setting ECT(0) while it adapts its behaviour to coexist with Classic flows. This is because the sender needs to continue to check whether it made the right decision – and switch back if it was wrong, or if a different link becomes the bottleneck:

- \* If, as recommended, the sender changes only its behaviour but not its codepoint to Classic, its codepoint will still be compatible with either an L4S or a Classic AQM. If the bottleneck does actually support both, it will still classify ECT(1) into the same L4S queue, where the sender can measure that switching to Classic behaviour was wrong, so that it can switch back.
- \* In contrast, if the sender changes both its behaviour and its codepoint to Classic, even if the bottleneck supports both, it will classify ECT(0) into the Classic queue, reinforcing the sender's incorrect decision so that it never switches back.
- \* Also, not changing codepoint avoids the risk of being flipped to a different path by a load balancer or multipath routing that hashes on the whole of the ex-ToS byte (unfortunately still a common pathology).

Note that if a flow is configured to only use a Classic congestion control, it is then entirely appropriate not to use ECT(1).

#### A.1.6. Reduce RTT dependence

Description: A scalable congestion control needs to reduce RTT bias as much as possible at least over the low to typical range of RTTs that will interact in the intended deployment scenario (see the precise normative requirement wording in Section 4.3).

Motivation: The throughput of Classic congestion controls is known to be inversely proportional to RTT, so one would expect flows over very low RTT paths to nearly starve flows over larger RTTs. However, Classic congestion controls have never allowed a very low RTT path to exist because they induce a large queue. For instance, consider two paths with base RTT 1 ms and 100 ms. If a Classic congestion control induces a 100 ms queue, it turns these RTTs into 101 ms and 200 ms leading to a throughput ratio of about 2:1. Whereas if a scalable congestion control induces only a 1 ms queue, the ratio is 2:101, leading to a throughput ratio of about 50:1.

Therefore, with very small queues, long RTT flows will essentially starve, unless scalable congestion controls comply with this requirement in Section 4.3.

The RTT bias in current Classic congestion controls works satisfactorily when the RTT is higher than typical, and L4S does not change that. So, there is no additional requirement in Section 4.3 for high RTT L4S flows to remove RTT bias - they can but they don't have to.

## A.1.7. Scaling down to fractional congestion windows

Description: A scalable congestion control needs to remain responsive to congestion when typical RTTs over the public Internet are significantly smaller because they are no longer inflated by queuing delay (see the precise normative requirement wording in Section 4.3).

Motivation: As currently specified, the minimum congestion window of ECN-capable TCP (and its derivatives) is expected to be 2 sender maximum segment sizes (SMSS), or 1 SMSS after a retransmission timeout. Once the congestion window reaches this minimum, if there is further ECN-marking, TCP is meant to wait for a retransmission timeout before sending another segment (see section 6.1.2 of the ECN spec [RFC3168]). In practice, most known window-based congestion control algorithms become unresponsive to ECN congestion signals at this point. No matter how much ECN marking, the congestion window no longer reduces. Instead, the sender's lack of any further congestion response forces the queue to grow, overriding any AQM and increasing queuing delay (making the window large enough to become responsive again). This can result in a stable but deeper queue, or it might drive the queue to loss, then the retransmission timeout mechanism acts as a backstop.

Most window-based congestion controls for other transport protocols have a similar minimum window, albeit when measured in bytes for those that use smaller packets.

L4S mechanisms significantly reduce queuing delay so, over the same path, the RTT becomes lower. Then this problem becomes surprisingly common [sub-mss-prob]. This is because, for the same link capacity, smaller RTT implies a smaller window. For instance, consider a residential setting with an upstream broadband Internet access of 8 Mb/s, assuming a max segment size of 1500 B. Two upstream flows will each have the minimum window of 2 SMSS if the RTT is 6 ms or less, which is quite common when accessing a nearby data centre. So, any more than two such parallel TCP flows will become unresponsive to ECN and increase queuing delay.

Unless scalable congestion controls address the requirement in Section 4.3 from the start, they will frequently become unresponsive to ECN, negating the low latency benefit of L4S, for themselves and for others.

That would seem to imply that scalable congestion controllers ought to be required to be able work with a congestion window less than 1 SMSS. For instance, if an ECN-capable TCP gets an ECN-mark when it is already sitting at a window of 1 SMSS, RFC 3168 requires it to defer sending for a retransmission timeout. A less drastic but more

complex mechanism can maintain a congestion window less than 1 SMSS (significantly less if necessary), as described in [Ahmed19]. Other approaches are likely to be feasible.

However, the requirement in Section 4.3 is worded as a "SHOULD" because it is believed that the existence of a minimum window is not all bad. When competing with an unresponsive flow, a minimum window naturally protects the flow from starvation by at least keeping some data flowing.

By stating the requirement to go lower than 1 SMSS as a "SHOULD", while the requirement in RFC 3168 still stands as well, we shall be able to watch the choices of minimum window evolve in different scalable congestion controllers.

#### A.1.8. Measuring Reordering Tolerance in Time Units

Description: When detecting loss, a scalable congestion control needs to be tolerant to reordering over an adaptive time interval, which scales with throughput, rather than counting only in fixed units of packets, which does not scale (see the precise normative requirement wording in Section 4.3).

Motivation: A primary purpose of L4S is scalable throughput (it's in the name). Scalability in all dimensions is, of course, also a goal of all IETF technology. The inverse linear congestion response in Section 4.3 is necessary, but not sufficient, to solve the congestion control scalability problem identified in [RFC3649]. As well as maintaining frequent ECN signals as rate scales, it is also important to ensure that a potentially false perception of loss does not limit throughput scaling.

End-systems cannot know whether a missing packet is due to loss or reordering, except in hindsight - if it appears later. So they can only deem that there has been a loss if a gap in the sequence space has not been filled, either after a certain number of subsequent packets has arrived (e.g. the 3 DupACK rule of standard TCP congestion control [RFC5681]) or after a certain amount of time (e.g. the RACK approach [RFC8985]).

As we attempt to scale packet rate over the years:

- \* Even if only some sending hosts still deem that loss has occurred by counting reordered packets, all networks will have to keep reducing the time over which they keep packets in order. If some link technologies keep the time within which reordering occurs roughly unchanged, then loss over these links, as perceived by these hosts, will appear to continually rise over the years.

- \* In contrast, if all senders detect loss in units of time, the time over which the network has to keep packets in order stays roughly invariant.

Therefore hosts have an incentive to detect loss in time units (so as not to fool themselves too often into detecting losses when there are none). And for hosts that are changing their congestion control implementation to L4S, there is no downside to including time-based loss detection code in the change (loss recovery implemented in hardware is an exception, covered later). Therefore requiring L4S hosts to detect loss in time-based units would not be a burden.

If the requirement in Section 4.3 were not placed on L4S hosts, even though it would be no burden on hosts to comply, all networks would face unnecessary uncertainty over whether some L4S hosts might be detecting loss by counting packets. Then all link technologies will have to unnecessarily keep reducing the time within which reordering occurs. That is not a problem for some link technologies, but it becomes increasingly challenging for other link technologies to continue to scale, particularly those relying on channel bonding for scaling, such as LTE, 5G and DOCSIS.

Given Internet paths traverse many link technologies, any scaling limit for these more challenging access link technologies would become a scaling limit for the Internet as a whole.

It might be asked how it helps to place this loss detection requirement only on L4S hosts, because networks will still face uncertainty over whether non-L4S flows are detecting loss by counting DupACKs. The answer is that those link technologies for which it is challenging to keep squeezing the reordering time will only need to do so for non-L4S traffic (which they can do because the L4S identifier is visible at the IP layer). Therefore, they can focus their processing and memory resources into scaling non-L4S (Classic) traffic. Then, the higher the proportion of L4S traffic, the less of a scaling challenge they will have.

To summarize, there is no reason for L4S hosts not to be part of the solution instead of part of the problem.



Requirement ("MUST") or recommendation ("SHOULD")? As explained above, this is a subtle interoperability issue between hosts and networks, which seems to need a "MUST". Unless networks can be certain that all L4S hosts follow the time-based approach, they still have to cater for the worst case - continually squeeze reordering into a smaller and smaller duration - just for hosts that might be using the counting approach. However, it was decided to express this as a recommendation, using "SHOULD". The main justification was that networks can still be fairly certain that L4S hosts will follow this recommendation, because following it offers only gain and no pain.

Details:

The speed of loss recovery is much more significant for short flows than long, therefore a good compromise is to adapt the reordering window; from a small fraction of the RTT at the start of a flow, to a larger fraction of the RTT for flows that continue for many round trips.

This is broadly the approach adopted by TCP RACK (Recent ACKnowledgements) [RFC8985]. However, RACK starts with the 3 DupACK approach, because the RTT estimate is not necessarily stable. As long as the initial window is paced, such initial use of 3 DupACK counting would amount to time-based loss detection and therefore would satisfy the time-based loss detection recommendation of Section 4.3. This is because pacing of the initial window would ensure that 3 DupACKs early in the connection would be spread over a small fraction of the round trip.

As mentioned above, hardware implementations of loss recovery using DupACK counting exist (e.g. some implementations of RoCEv2 for RDMA). For low latency, these implementations can change their congestion control to implement L4S, because the congestion control (as distinct from loss recovery) is implemented in software. But they cannot easily satisfy this loss recovery requirement. However, it is believed they do not need to, because such implementations are believed to solely exist in controlled environments, where the network technology keeps reordering extremely low anyway. This is why controlled environments with hardly any reordering are excluded from the scope of the normative recommendation in Section 4.3.

Detecting loss in time units also prevents the ACK-splitting attacks described in [Savage-TCP].

## A.2. Scalable Transport Protocol Optimizations

### A.2.1. Setting ECT in Control Packets and Retransmissions

Description: This item concerns TCP and its derivatives (e.g. SCTP) as well as RTP/RTCP [RFC6679]. The original specification of ECN for TCP precluded the use of ECN on control packets and retransmissions. Similarly RFC 6679 precludes the use of ECT on RTCP datagrams, in case the path changes after it has been checked for ECN traversal. To improve performance, scalable transport protocols ought to enable ECN at the IP layer in TCP control packets (SYN, SYN-ACK, pure ACKs, etc.) and in retransmitted packets. The same is true for other transports, e.g. SCTP, RTCP.

Motivation (TCP): RFC 3168 prohibits the use of ECN on these types of TCP packet, based on a number of arguments. This means these packets are not protected from congestion loss by ECN, which considerably harms performance, particularly for short flows. ECN++ [I-D.ietf-tcpm-generalized-ecn] proposes experimental use of ECN on all types of TCP packet as long as AccECN feedback [I-D.ietf-tcpm-accurate-ecn] is available (which itself satisfies the accurate feedback requirement in Section 4.2 for using a scalable congestion control).

Motivation (RTCP): L4S experiments in general will need to observe the rule in the RTP ECN spec [RFC6679] that precludes ECT on RTCP datagrams. Nonetheless, as ECN usage becomes more widespread, it would be useful to conduct specific experiments with ECN-capable RTCP to gather data on whether such caution is necessary.

### A.2.2. Faster than Additive Increase

Description: It would improve performance if scalable congestion controls did not limit their congestion window increase to the standard additive increase of 1 SMSS per round trip [RFC5681] during congestion avoidance. The same is true for derivatives of TCP congestion control, including similar approaches used for real-time media.

Motivation: As currently defined [RFC8257], DCTCP uses the traditional Reno additive increase in congestion avoidance phase. When the available capacity suddenly increases (e.g. when another flow finishes, or if radio capacity increases) it can take very many round trips to take advantage of the new capacity. TCP Cubic [RFC8312] was designed to solve this problem, but as flow rates have continued to increase, the delay accelerating into available capacity has become prohibitive. See, for instance, the examples in

Section 5.1 of the L4S architecture [I-D.ietf-tsvwg-l4s-arch]. Even when out of its Reno-compatibility mode, every 8x scaling of Cubic's flow rate leads to 2x more acceleration delay.

In the steady state, DCTCP induces about 2 ECN marks per round trip, so it is possible to quickly detect when these signals have disappeared and seek available capacity more rapidly, while minimizing the impact on other flows (Classic and scalable) [LinuxPacedChirping]. Alternatively, approaches such as Adaptive Acceleration (A2DTCP [A2DTCP]) have been proposed to address this problem in data centres, which might be deployable over the public Internet.

#### A.2.3. Faster Convergence at Flow Start

Description: It would improve performance if scalable congestion controls converged (reached their steady-state share of the capacity) faster than Classic congestion controls or at least no slower. This affects the flow start behaviour of any L4S congestion control derived from a Classic transport that uses TCP slow start, including those for real-time media.

Motivation: As an example, a new DCTCP flow takes longer than a Classic congestion control to obtain its share of the capacity of the bottleneck when there are already ongoing flows using the bottleneck capacity. In a data centre environment DCTCP takes about a factor of 1.5 to 2 longer to converge due to the much higher typical level of ECN marking that DCTCP background traffic induces, which causes new flows to exit slow start early [Alizadeh-stability]. In testing for use over the public Internet the convergence time of DCTCP relative to a regular loss-based TCP slow start is even less favourable [Paced-Chirping] due to the shallow ECN marking threshold needed for L4S. It is exacerbated by the typically greater mismatch between the link rate of the sending host and typical Internet access bottlenecks. This problem is detrimental in general, but would particularly harm the performance of short flows relative to Classic congestion controls.

#### Appendix B. Compromises in the Choice of L4S Identifier

This appendix is informative, not normative. As explained in Section 2, there is insufficient space in the IP header (v4 or v6) to fully accommodate every requirement. So the choice of L4S identifier involves tradeoffs. This appendix records the pros and cons of the choice that was made.

Non-normative recap of the chosen codepoint scheme:

Packets with ECT(1) and conditionally packets with CE signify L4S semantics as an alternative to the semantics of Classic ECN [RFC3168], specifically:

- The ECT(1) codepoint signifies that the packet was sent by an L4S-capable sender.
- Given shortage of codepoints, both L4S and Classic ECN sides of an AQM have to use the same CE codepoint to indicate that a packet has experienced congestion. If a packet that had already been marked CE in an upstream buffer arrived at a subsequent AQM, this AQM would then have to guess whether to classify CE packets as L4S or Classic ECN. Choosing the L4S treatment is a safer choice, because then a few Classic packets might arrive early, rather than a few L4S packets arriving late.
- Additional information might be available if the classifier were transport-aware. Then it could classify a CE packet for Classic ECN treatment if the most recent ECT packet in the same flow had been marked ECT(0). However, the L4S service ought not to need transport-layer awareness.

Cons:

Consumes the last ECN codepoint: The L4S service could potentially supersede the service provided by Classic ECN, therefore using ECT(1) to identify L4S packets could ultimately mean that the ECT(0) codepoint was 'wasted' purely to distinguish one form of ECN from its successor.

ECN hard in some lower layers: It is not always possible to support the equivalent of an IP-ECN field in an AQM acting in a buffer below the IP layer [I-D.ietf-tsvwg-ecn-encap-guidelines]. Then, depending on the lower layer scheme, the L4S service might have to drop rather than mark frames even though they might encapsulate an ECN-capable packet.

Risk of reordering Classic CE packets within a flow: Classifying all CE packets into the L4S queue risks any CE packets that were originally ECT(0) being incorrectly classified as L4S. If there were delay in the Classic queue, these incorrectly classified CE packets would arrive early, which is a form of reordering. Reordering within a microflow can cause TCP senders (and senders of similar transports) to retransmit spuriously. However, the risk of spurious retransmissions would be extremely low for the following reasons:

1. It is quite unusual to experience queuing at more than one bottleneck on the same path (the available capacities have to be identical).
2. In only a subset of these unusual cases would the first bottleneck support Classic ECN marking while the second supported L4S ECN marking, which would be the only scenario where some ECT(0) packets could be CE marked by an AQM supporting Classic ECN then the remainder experienced further delay through the Classic side of a subsequent L4S DualQ AQM.
3. Even then, when a few packets are delivered early, it takes very unusual conditions to cause a spurious retransmission, in contrast to when some packets are delivered late. The first bottleneck has to apply CE-marks to at least N contiguous packets and the second bottleneck has to inject an uninterrupted sequence of at least N of these packets between two packets earlier in the stream (where N is the reordering window that the transport protocol allows before it considers a packet is lost).

For example consider  $N=3$ , and consider the sequence of packets 100, 101, 102, 103,... and imagine that packets 150,151,152 from later in the flow are injected as follows: 100, 150, 151, 101, 152, 102, 103... If this were late reordering, even one packet arriving out of sequence would trigger a spurious retransmission, but there is no spurious retransmission here with early reordering, because packet 101 moves the cumulative ACK counter forward before 3 packets have arrived out of order. Later, when packets 148, 149, 153... arrive, even though there is a 3-packet hole, there will be no problem, because the packets to fill the hole are already in the receive buffer.

4. Even with the current TCP recommendation of  $N=3$  [RFC5681] spurious retransmissions will be unlikely for all the above reasons. As RACK [RFC8985] is becoming widely deployed, it tends to adapt its reordering window to a larger value of N, which will make the chance of a contiguous sequence of N early arrivals vanishingly small.
5. Even a run of 2 CE marks within a Classic ECN flow is unlikely, given FQ-CoDel is the only known widely deployed AQM that supports Classic ECN marking and it takes great care to separate out flows and to space any markings evenly along each flow.

It is extremely unlikely that the above set of 5 eventualities that are each unusual in themselves would all happen simultaneously. But, even if they did, the consequences would hardly be dire: the odd spurious fast retransmission. Whenever the traffic source (a Classic congestion control) mistakes the reordering of a string of CE marks for a loss, one might think that it will reduce its congestion window as well as emitting a spurious retransmission. However, it would have already reduced its congestion window when the CE markings arrived early. If it is using ABE [RFC8511], it might reduce cwnd a little more for a loss than for a CE mark. But it will revert that reduction once it detects that the retransmission was spurious.

In conclusion, the impact of early reordering on spurious retransmissions due to CE being ambiguous will generally be vanishingly small.

Insufficient anti-replay window in some pre-existing VPNs: If delay is reduced for a subset of the flows within a VPN, the anti-replay feature of some VPNs is known to potentially mistake the difference in delay for a replay attack. Section 6.2 recommends that the anti-replay window at the VPN egress is sufficiently sized, as required by the relevant specifications. However, in some VPN implementations the maximum anti-replay window is insufficient to cater for a large delay difference at prevailing packet rates. Section 6.2 suggests alternative work-rounds for such cases, but end-users using L4S over a VPN will need to be able to recognize the symptoms of this problem, in order to seek out these work-rounds.

Hard to distinguish Classic ECN AQM: With this scheme, when a source receives ECN feedback, it is not explicitly clear which type of AQM generated the CE markings. This is not a problem for Classic ECN sources that send ECT(0) packets, because an L4S AQM will recognize the ECT(0) packets as Classic and apply the appropriate Classic ECN marking behaviour.

However, in the absence of explicit disambiguation of the CE markings, an L4S source needs to use heuristic techniques to work out which type of congestion response to apply (see Appendix A.1.5). Otherwise, if long-running Classic flow(s) are sharing a Classic ECN AQM bottleneck with long-running L4S flow(s), which then apply an L4S response to Classic CE signals, the L4S flows would outcompete the Classic flow(s). Experiments have shown that L4S flows can take about 20 times more capacity share than equivalent Classic flows. Nonetheless, as link capacity reduces (e.g. to 4 Mb/s), the inequality reduces. So Classic flows always make progress and are not starved.

When L4S was first proposed (in 2015, 14 years after the Classic ECN spec [RFC3168] was published), it was believed that Classic ECN AQMs had failed to be deployed, because research measurements had found little or no evidence of CE marking. In subsequent years Classic ECN was included in per-flow-queuing (FQ) deployments, however an FQ scheduler stops an L4S flow outcompeting Classic, because it enforces equality between flow rates. It is not known whether there have been any non-FQ deployments of Classic ECN AQMs in the subsequent years, or whether there will be in future.

An algorithm for detecting a Classic ECN AQM as soon as a flow stabilizes after start-up has been proposed [ecn-fallback] (see Appendix A.1.5 for a brief summary). Testbed evaluations of v2 of the algorithm have shown detection is reasonably good for Classic ECN AQMs, in a wide range of circumstances. However, although it can correctly detect an L4S ECN AQM in many circumstances, it is often incorrect at low link capacities and/or high RTTs. Although this is the safe way round, there is a danger that it will discourage use of the algorithm.

Non-L4S service for control packets: Solely for the case of TCP, the Classic ECN RFCs [RFC3168] and [RFC5562] require a sender to clear the ECN field to Not-ECT on retransmissions and on certain control packets specifically pure ACKs, window probes and SYNs. When L4S packets are classified by the ECN field, these TCP control packets would not be classified into an L4S queue, and could therefore be delayed relative to the other packets in the flow. This would not cause reordering (because retransmissions are already out of order, and these control packets typically carry no data). However, it would make critical TCP control packets more vulnerable to loss and delay. To address this problem, ECN++ [I-D.ietf-tcpm-generalized-ecn] proposes an experiment in which all TCP control packets and retransmissions are ECN-capable as long as appropriate ECN feedback is available in each case.

Pros:

Should work e2e: The ECN field generally propagates end-to-end across the Internet without being wiped or mangled, at least over fixed networks. Unlike the DSCP, the setting of the ECN field is at least meant to be forwarded unchanged by networks that do not support ECN.

Should work in tunnels: The L4S identifiers work across and within

any tunnel that propagates the ECN field in any of the variant ways it has been defined since ECN-tunneling was first specified in the year 2001 [RFC3168]. However, it is likely that some tunnels still do not implement ECN propagation at all.

Should work for many link technologies: At most, but not all, path bottlenecks there is IP-awareness, so that L4S AQMs can be located where the IP-ECN field can be manipulated. Bottlenecks at lower layer nodes without IP-awareness either have to use drop to signal congestion or a specific congestion notification facility has to be defined for that link technology, including propagation to and from IP-ECN. The programme to define these is progressing and in each case so far the scheme already defined for ECN inherently supports L4S as well (see Section 6.1).

Could migrate to one codepoint: If all Classic ECN senders eventually evolve to use the L4S service, the ECT(0) codepoint could be reused for some future purpose, but only once use of ECT(0) packets had reduced to zero, or near-zero, which might never happen.

L4 not required: Being based on the ECN field, this scheme does not need the network to access transport layer flow identifiers. Nonetheless, it does not preclude solutions that do.

#### Appendix C. Potential Competing Uses for the ECT(1) Codepoint

The ECT(1) codepoint of the ECN field has already been assigned once for the ECN nonce [RFC3540], which has now been categorized as historic [RFC8311]. ECN is probably the only remaining field in the Internet Protocol that is common to IPv4 and IPv6 and still has potential to work end-to-end, with tunnels and with lower layers. Therefore, ECT(1) should not be reassigned to a different experimental use (L4S) without carefully assessing competing potential uses. These fall into the following categories:

##### C.1. Integrity of Congestion Feedback

Receiving hosts can fool a sender into downloading faster by suppressing feedback of ECN marks (or of losses if retransmissions are not necessary or available otherwise).



The historic ECN nonce protocol [RFC3540] proposed that a TCP sender could set either of ECT(0) or ECT(1) in each packet of a flow and remember the sequence it had set. If any packet was lost or congestion marked, the receiver would miss that bit of the sequence. An ECN Nonce receiver had to feed back the least significant bit of the sum, so it could not suppress feedback of a loss or mark without a 50-50 chance of guessing the sum incorrectly.

It is highly unlikely that ECT(1) will be needed for integrity protection in future. The ECN Nonce RFC [RFC3540] has been reclassified as historic, partly because other ways have been developed to protect feedback integrity of TCP and other transports [RFC8311] that do not consume a codepoint in the IP header. For instance:

- \* the sender can test the integrity of the receiver's feedback by occasionally setting the IP-ECN field to a value normally only set by the network. Then it can test whether the receiver's feedback faithfully reports what it expects (see para 2 of Section 20.2 of the ECN spec [RFC3168]). This works for loss and it will work for the accurate ECN feedback [RFC7560] intended for L4S.
- \* A network can enforce a congestion response to its ECN markings (or packet losses) by auditing congestion exposure (ConEx) [RFC7713]. Whether the receiver or a downstream network is suppressing congestion feedback or the sender is unresponsive to the feedback, or both, ConEx audit can neutralise any advantage that any of these three parties would otherwise gain.
- \* The TCP authentication option (TCP-AO [RFC5925]) can be used to detect any tampering with TCP congestion feedback (whether malicious or accidental). TCP's congestion feedback fields are immutable end-to-end, so they are amenable to TCP-AO protection, which covers the main TCP header and TCP options by default. However, TCP-AO is often too brittle to use on many end-to-end paths, where middleboxes can make verification fail in their attempts to improve performance or security, e.g. by resegmentation or shifting the sequence space.

## C.2. Notification of Less Severe Congestion than CE

Various researchers have proposed to use ECT(1) as a less severe congestion notification than CE, particularly to enable flows to fill available capacity more quickly after an idle period, when another flow departs or when a flow starts, e.g. VCP [VCP], Queue View (QV) [QV].

Before assigning ECT(1) as an identifier for L4S, we must carefully consider whether it might be better to hold ECT(1) in reserve for future standardisation of rapid flow acceleration, which is an important and enduring problem [RFC6077].

Pre-Congestion Notification (PCN) is another scheme that assigns alternative semantics to the ECN field. It uses ECT(1) to signify a less severe level of pre-congestion notification than CE [RFC6660]. However, the ECN field only takes on the PCN semantics if packets carry a Diffserv codepoint defined to indicate PCN marking within a controlled environment. PCN is required to be applied solely to the outer header of a tunnel across the controlled region in order not to interfere with any end-to-end use of the ECN field. Therefore a PCN region on the path would not interfere with the L4S service identifier defined in Section 3.

#### Authors' Addresses

Koen De Schepper  
Nokia Bell Labs  
Antwerp  
Belgium  
Email: [koen.de\\_schepper@nokia.com](mailto:koen.de_schepper@nokia.com)  
URI: [https://www.bell-labs.com/usr/koen.de\\_schepper](https://www.bell-labs.com/usr/koen.de_schepper)

Bob Briscoe (editor)  
Independent  
United Kingdom  
Email: [ietf@bobbriscoe.net](mailto:ietf@bobbriscoe.net)  
URI: <http://bobbriscoe.net/>

TSVWG  
Internet-Draft  
Updates: 6363 (if approved)  
Intended status: Standards Track  
Expires: July 15, 2019

V. Roca  
INRIA  
A. Begen  
Networked Media  
January 11, 2019

Forward Error Correction (FEC) Framework Extension to Sliding Window  
Codes  
draft-ietf-tsvwg-fecframe-ext-08

Abstract

RFC 6363 describes a framework for using Forward Error Correction (FEC) codes to provide protection against packet loss. The framework supports applying FEC to arbitrary packet flows over unreliable transport and is primarily intended for real-time, or streaming, media. However, FECFRAME as per RFC 6363 is restricted to block FEC codes. This document updates RFC 6363 to support FEC Codes based on a sliding encoding window, in addition to Block FEC Codes, in a backward-compatible way. During multicast/broadcast real-time content delivery, the use of sliding window codes significantly improves robustness in harsh environments, with less repair traffic and lower FEC-related added latency.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 15, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Definitions and Abbreviations . . . . .	4
3. Summary of Architecture Overview . . . . .	7
4. Procedural Overview . . . . .	10
4.1. General . . . . .	10
4.2. Sender Operation with Sliding Window FEC Codes . . . . .	10
4.3. Receiver Operation with Sliding Window FEC Codes . . . . .	13
5. Protocol Specification . . . . .	15
5.1. General . . . . .	15
5.2. FEC Framework Configuration Information . . . . .	16
5.3. FEC Scheme Requirements . . . . .	16
6. Feedback . . . . .	16
7. Transport Protocols . . . . .	17
8. Congestion Control . . . . .	17
9. Implementation Status . . . . .	17
10. Security Considerations . . . . .	17
11. Operations and Management Considerations . . . . .	18
12. IANA Considerations . . . . .	18
13. Acknowledgments . . . . .	18
14. References . . . . .	18
14.1. Normative References . . . . .	18
14.2. Informative References . . . . .	19
Appendix A. About Sliding Encoding Window Management (informational) . . . . .	20
Authors' Addresses . . . . .	21

## 1. Introduction

Many applications need to transport a continuous stream of packetized data from a source (sender) to one or more destinations (receivers) over networks that do not provide guaranteed packet delivery. In particular packets may be lost, which is strictly the focus of this document: we assume that transmitted packets are either lost (e.g., because of a congested router, of a poor signal-to-noise ratio in a wireless network, or because the number of bit errors exceeds the correction capabilities of the physical-layer error correcting code)

or received by the transport protocol without any corruption (i.e., the bit-errors, if any, have been fixed by the physical-layer error correcting code and therefore are hidden to the upper layers).

For these use-cases, Forward Error Correction (FEC) applied within the transport or application layer is an efficient technique to improve packet transmission robustness in presence of packet losses (or "erasures"), without going through packet retransmissions that create a delay often incompatible with real-time constraints. The FEC Building Block defined in [RFC5052] provides a framework for the definition of Content Delivery Protocols (CDPs) that make use of separately-defined FEC schemes. Any CDP defined according to the requirements of the FEC Building Block can then easily be used with any FEC Scheme that is also defined according to the requirements of the FEC Building Block.

Then FECFRAME [RFC6363] provides a framework to define Content Delivery Protocols (CDPs) that provide FEC protection for arbitrary packet flows over an unreliable datagram service transport such as UDP. It is primarily intended for real-time or streaming media applications, using broadcast, multicast, or on-demand delivery.

However, [RFC6363] only considers block FEC schemes defined in accordance with the FEC Building Block [RFC5052] (e.g., [RFC6681], [RFC6816] or [RFC6865]). These codes require the input flow(s) to be segmented into a sequence of blocks. Then FEC encoding (at a sender or an encoding middlebox) and decoding (at a receiver or a decoding middlebox) are both performed on a per-block basis. For instance, if the current block encompasses the 100's to 119's source symbols (i.e., a block of size 20 symbols) of an input flow, encoding (and decoding) will be performed on this block independently of other blocks. This approach has major impacts on FEC encoding and decoding delays. The data packets of continuous media flow(s) may be passed to the transport layer immediately, without delay. But the block creation time, that depends on the number of source symbols in this block, impacts both the FEC encoding delay (since encoding requires that all source symbols be known), and mechanically the packet loss recovery delay at a receiver (since no repair symbol for the current block can be generated and therefore received before that time). Therefore a good value for the block size is necessarily a balance between the maximum FEC decoding latency at the receivers (which must be in line with the most stringent real-time requirement of the protected flow(s), hence an incentive to reduce the block size), and the desired robustness against long loss bursts (which increases with the block size, hence an incentive to increase this size).

This document updates [RFC6363] in order to also support FEC codes based on a sliding encoding window (A.K.A. convolutional codes)

[RFC8406]. This encoding window, either of fixed or variable size, slides over the set of source symbols. FEC encoding is launched whenever needed, from the set of source symbols present in the sliding encoding window at that time. This approach significantly reduces FEC-related latency, since repair symbols can be generated and passed to the transport layer on-the-fly, at any time, and can be regularly received by receivers to quickly recover packet losses. Using sliding window FEC codes is therefore highly beneficial to real-time flows, one of the primary targets of FECFRAME. [RLC-ID] provides an example of such FEC Scheme for FECFRAME, built upon the simple sliding window Random Linear Codes (RLC).

This document is fully backward compatible with [RFC6363]. Indeed:

- o this FECFRAME update does not prevent nor compromise in any way the support of block FEC codes. Both types of codes can nicely co-exist, just like different block FEC schemes can co-exist;
- o each sliding window FEC Scheme is associated to a specific FEC Encoding ID subject to IANA registration, just like block FEC Schemes;
- o any receiver, for instance a legacy receiver that only supports block FEC schemes, can easily identify the FEC Scheme used in a FECFRAME session. Indeed, the FEC Encoding ID that identifies the FEC Scheme is carried in the FEC Framework Configuration Information (see section 5.5 of [RFC6363]). For instance, when the Session Description Protocol (SDP) is used to carry the FEC Framework Configuration Information, the FEC Encoding ID can be communicated in the "encoding-id=" parameter of a "fec-repair-flow" attribute [RFC6364]. This mechanism is the basic approach for a FECFRAME receiver to determine whether or not it supports the FEC Scheme used in a given FECFRAME session;

This document leverages on [RFC6363] and re-uses its structure. It proposes new sections specific to sliding window FEC codes whenever required. The only exception is Section 3 that provides a quick summary of FECFRAME in order to facilitate the understanding of this document to readers not familiar with the concepts and terminology.

## 2. Definitions and Abbreviations

The following list of definitions and abbreviations is copied from [RFC6363], adding only the Block/sliding window FEC Code and Encoding/Decoding Window definitions (tagged with "ADDED"):

**Application Data Unit (ADU):** The unit of source data provided as payload to the transport layer. For instance, it can be a

payload containing the result of the RTP packetization of a compressed video frame.

**ADU Flow:** A sequence of ADUs associated with a transport-layer flow identifier (such as the standard 5-tuple {source IP address, source port, destination IP address, destination port, transport protocol}).

**AL-FEC:** Application-layer Forward Error Correction.

**Application Protocol:** Control protocol used to establish and control the source flow being protected, e.g., the Real-Time Streaming Protocol (RTSP).

**Content Delivery Protocol (CDP):** A complete application protocol specification that, through the use of the framework defined in this document, is able to make use of FEC schemes to provide FEC capabilities.

**FEC Code:** An algorithm for encoding data such that the encoded data flow is resilient to data loss. Note that, in general, FEC codes may also be used to make a data flow resilient to corruption, but that is not considered in this document.

**Block FEC Code: (ADDED)** An FEC Code that operates on blocks, i.e., for which the input flow MUST be segmented into a sequence of blocks, FEC encoding and decoding being performed independently on a per-block basis.

**Sliding Window FEC Code: (ADDED)** An FEC Code that can generate repair symbols on-the-fly, at any time, from the set of source symbols present in the sliding encoding window at that time. These codes are also known as convolutional codes.

**FEC Framework:** A protocol framework for the definition of Content Delivery Protocols using FEC, such as the framework defined in this document.

**FEC Framework Configuration Information:** Information that controls the operation of the FEC Framework.

**FEC Payload ID:** Information that identifies the contents and provides positional information of a packet with respect to the FEC Scheme.

**FEC Repair Packet:** At a sender (respectively, at a receiver), a payload submitted to (respectively, received from) the transport

protocol containing one or more repair symbols along with a Repair FEC Payload ID and possibly an RTP header.

**FEC Scheme:** A specification that defines the additional protocol aspects required to use a particular FEC code with the FEC Framework.

**FEC Source Packet:** At a sender (respectively, at a receiver), a payload submitted to (respectively, received from) the transport protocol containing an ADU along with an optional Explicit Source FEC Payload ID.

**Repair Flow:** The packet flow carrying FEC data.

**Repair FEC Payload ID:** A FEC Payload ID specifically for use with repair packets.

**Source Flow:** The packet flow to which FEC protection is to be applied. A source flow consists of ADUs.

**Source FEC Payload ID:** A FEC Payload ID specifically for use with source packets.

**Source Protocol:** A protocol used for the source flow being protected, e.g., RTP.

**Transport Protocol:** The protocol used for the transport of the source and repair flows, using an unreliable datagram service such as UDP.

**Encoding Window:** (ADDED) Set of Source Symbols available at the sender/coding node that are used to generate a repair symbol, with a Sliding Window FEC Code.

**Decoding Window:** (ADDED) Set of received or decoded source and repair symbols available at a receiver that are used to decode erased source symbols, with a Sliding Window FEC Code.

**Code Rate:** The ratio between the number of source symbols and the number of encoding symbols. By definition, the code rate is such that  $0 < \text{code rate} \leq 1$ . A code rate close to 1 indicates that a small number of repair symbols have been produced during the encoding process.

**Encoding Symbol:** Unit of data generated by the encoding process. With systematic codes, source symbols are part of the encoding symbols.



**Packet Erasure Channel:** A communication path where packets are either lost (e.g., in our case, by a congested router, or because the number of transmission errors exceeds the correction capabilities of the physical-layer code) or received. When a packet is received, it is assumed that this packet is not corrupted (i.e., in our case, the bit-errors, if any, are fixed by the physical-layer code and therefore hidden to the upper layers).

**Repair Symbol:** Encoding symbol that is not a source symbol.

**Source Block:** Group of ADUs that are to be FEC protected as a single block. This notion is restricted to Block FEC Codes.

**Source Symbol:** Unit of data used during the encoding process.

**Systematic Code:** FEC code in which the source symbols are part of the encoding symbols.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

### 3. Summary of Architecture Overview

The architecture of [RFC6363], Section 3, equally applies to this FECFRAME extension and is not repeated here. However, we provide hereafter a quick summary to facilitate the understanding of this document to readers not familiar with the concepts and terminology.

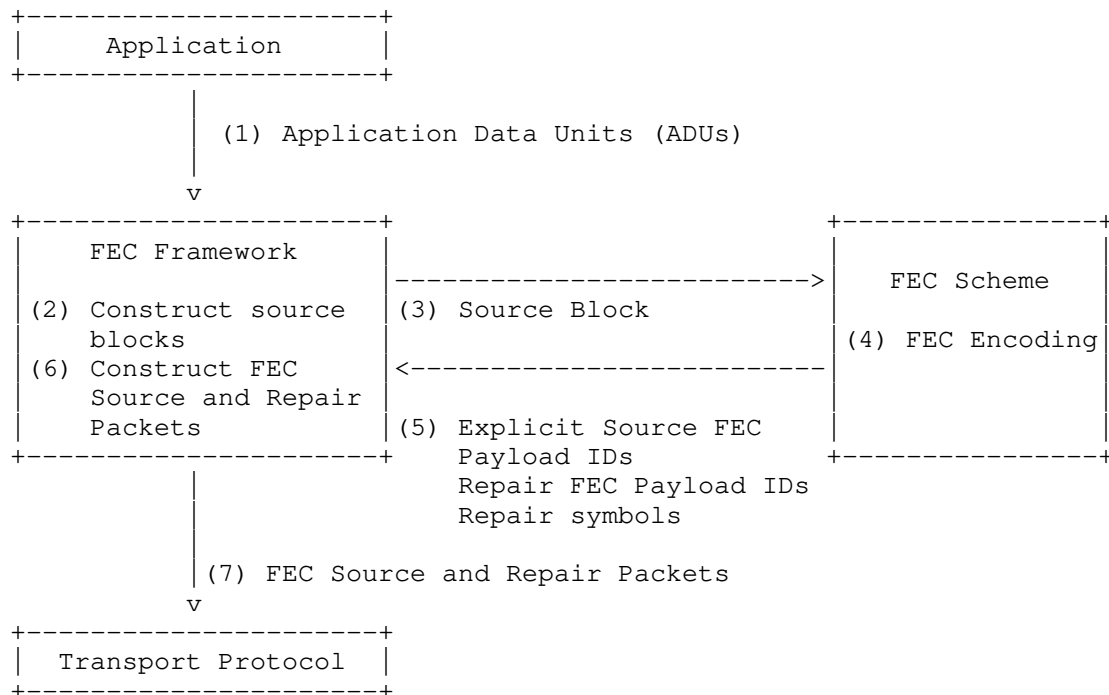


Figure 1: FECFRAME architecture at a sender.

The FECFRAME architecture is illustrated in Figure 1 from the sender's point of view, in case of a block FEC Scheme. It shows an application generating an ADU flow (other flows, from other applications, may co-exist). These ADUs, of variable size, must be somehow mapped to source symbols of fixed size (this fixed size is a requirement of all FEC Schemes that comes from the way mathematical operations are applied to symbols content). This is the goal of an ADU-to-symbols mapping process that is FEC-Scheme specific (see below). Once the source block is built, taking into account both the FEC Scheme constraints (e.g., in terms of maximum source block size) and the application's flow constraints (e.g., in terms of real-time constraints), the associated source symbols are handed to the FEC Scheme in order to produce an appropriate number of repair symbols. FEC Source Packets (containing ADUs) and FEC Repair Packets (containing one or more repair symbols each) are then generated and sent using an appropriate transport protocol (more precisely [RFC6363], Section 7, requires a transport protocol providing an unreliable datagram service, such as UDP). In practice FEC Source Packets may be passed to the transport layer as soon as available, without having to wait for FEC encoding to take place. In that case

a copy of the associated source symbols needs to be kept within FECFRAME for future FEC encoding purposes.

At a receiver (not shown), FECFRAME processing operates in a similar way, taking as input the incoming FEC Source and Repair Packets received. In case of FEC Source Packet losses, the FEC decoding of the associated block may recover all (in case of successful decoding) or a subset potentially empty (otherwise) of the missing source symbols. After source-symbol-to-ADU mapping, when lost ADUs are recovered, they are then assigned to their respective flow (see below). ADUs are returned to the application(s), either in their initial transmission order (in that case ADUs received after an erased one will be delayed until FEC decoding has taken place) or not (in that case each ADU is returned as soon as it is received or recovered), depending on the application requirements.

FECFRAME features two subtle mechanisms:

- o ADUs-to-source-symbols mapping: in order to manage variable size ADUs, FECFRAME and FEC Schemes can use small, fixed size symbols and create a mapping between ADUs and symbols. To each ADU this mechanism prepends a length field (plus a flow identifier, see below) and pads the result to a multiple of the symbol size. A small ADU may be mapped to a single source symbol while a large one may be mapped to multiple symbols. The mapping details are FEC-Scheme-dependent and must be defined in the associated document;
- o Assignment of decoded ADUs to flows in multi-flow configurations: when multiple flows are multiplexed over the same FECFRAME instance, a problem is to assign a decoded ADU to the right flow (UDP port numbers and IP addresses traditionally used to map incoming ADUs to flows are not recovered during FEC decoding). To make it possible, at the FECFRAME sending instance, each ADU is prepended with a flow identifier (1 byte) during the ADU-to-source-symbols mapping (see above). The flow identifiers are also shared between all FECFRAME instances as part of the FEC Framework Configuration Information. This (flow identifier + length + application payload + padding), called ADUI, is then FEC protected. Therefore a decoded ADUI contains enough information to assign the ADU to the right flow.

A few aspects are not covered by FECFRAME, namely:

- o [RFC6363] section 8 does not detail any congestion control mechanism, but only provides high level normative requirements;

- o the possibility of having feedbacks from receiver(s) is considered out of scope, although such a mechanism may exist within the application (e.g., through RTCP control messages);
- o flow adaptation at a FECFRAME sender (e.g., how to set the FEC code rate based on transmission conditions) is not detailed, but it needs to comply with the congestion control normative requirements (see above).

#### 4. Procedural Overview

##### 4.1. General

The general considerations of [RFC6363], Section 4.1, that are specific to block FEC codes are not repeated here.

With a Sliding Window FEC Code, the FEC Source Packet MUST contain information to identify the position occupied by the ADU within the source flow, in terms specific to the FEC Scheme. This information is known as the Source FEC Payload ID, and the FEC Scheme is responsible for defining and interpreting it.

With a Sliding Window FEC Code, the FEC Repair Packets MUST contain information that identifies the relationship between the contained repair payloads and the original source symbols used during encoding. This information is known as the Repair FEC Payload ID, and the FEC Scheme is responsible for defining and interpreting it.

The Sender Operation ([RFC6363], Section 4.2.) and Receiver Operation ([RFC6363], Section 4.3) are both specific to block FEC codes and therefore omitted below. The following two sections detail similar operations for Sliding Window FEC codes.

##### 4.2. Sender Operation with Sliding Window FEC Codes

With a Sliding Window FEC Scheme, the following operations, illustrated in Figure 2 for the generic case (non-RTP repair flows), and in Figure 3 for the case of RTP repair flows, describe a possible way to generate compliant source and repair flows:

1. A new ADU is provided by the application.
2. The FEC Framework communicates this ADU to the FEC Scheme.
3. The sliding encoding window is updated by the FEC Scheme. The ADU-to-source-symbols mapping as well as the encoding window management details are both the responsibility of the FEC Scheme

and MUST be detailed there. Appendix A provides non-normative hints about what FEC Scheme designers need to consider;

4. The Source FEC Payload ID information of the source packet is determined by the FEC Scheme. If required by the FEC Scheme, the Source FEC Payload ID is encoded into the Explicit Source FEC Payload ID field and returned to the FEC Framework.
5. The FEC Framework constructs the FEC Source Packet according to [RFC6363] Figure 6, using the Explicit Source FEC Payload ID provided by the FEC Scheme if applicable.
6. The FEC Source Packet is sent using normal transport-layer procedures. This packet is sent using the same ADU flow identification information as would have been used for the original source packet if the FEC Framework were not present (e.g., the source and destination addresses and UDP port numbers on the IP datagram carrying the source packet will be the same whether or not the FEC Framework is applied).
7. When the FEC Framework needs to send one or several FEC Repair Packets (e.g., according to the target Code Rate), it asks the FEC Scheme to create one or several repair packet payloads from the current sliding encoding window along with their Repair FEC Payload ID.
8. The Repair FEC Payload IDs and repair packet payloads are provided back by the FEC Scheme to the FEC Framework.
9. The FEC Framework constructs FEC Repair Packets according to [RFC6363] Figure 7, using the FEC Payload IDs and repair packet payloads provided by the FEC Scheme.
10. The FEC Repair Packets are sent using normal transport-layer procedures. The port(s) and multicast group(s) to be used for FEC Repair Packets are defined in the FEC Framework Configuration Information.

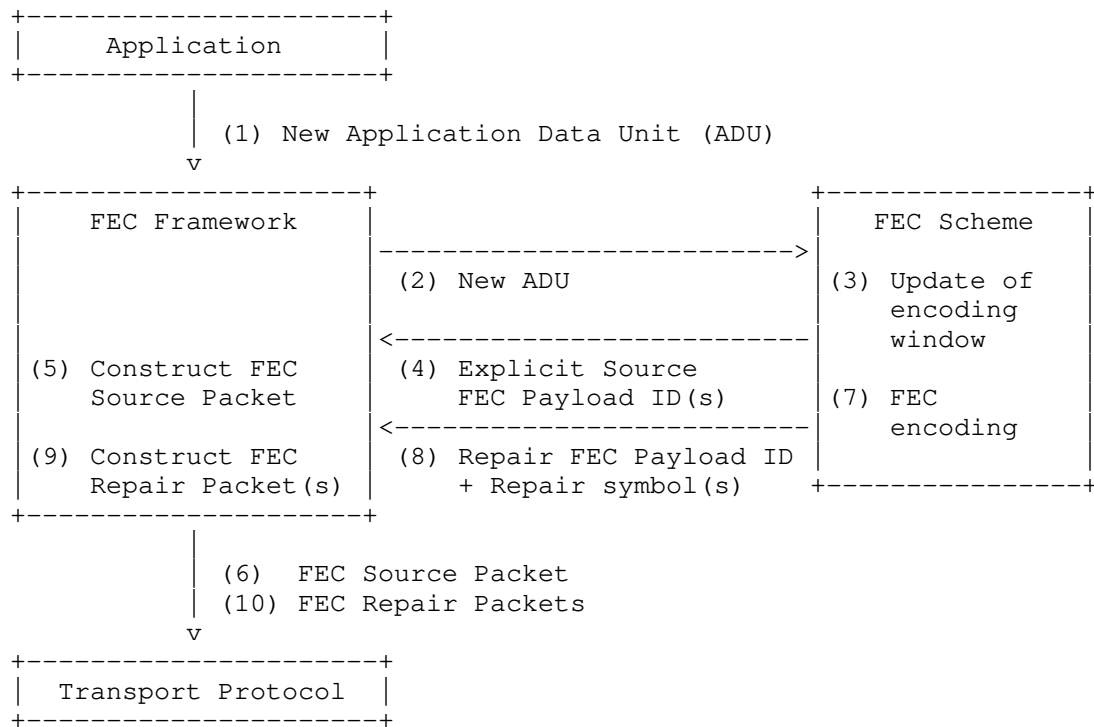


Figure 2: Sender Operation with Sliding Window FEC Codes

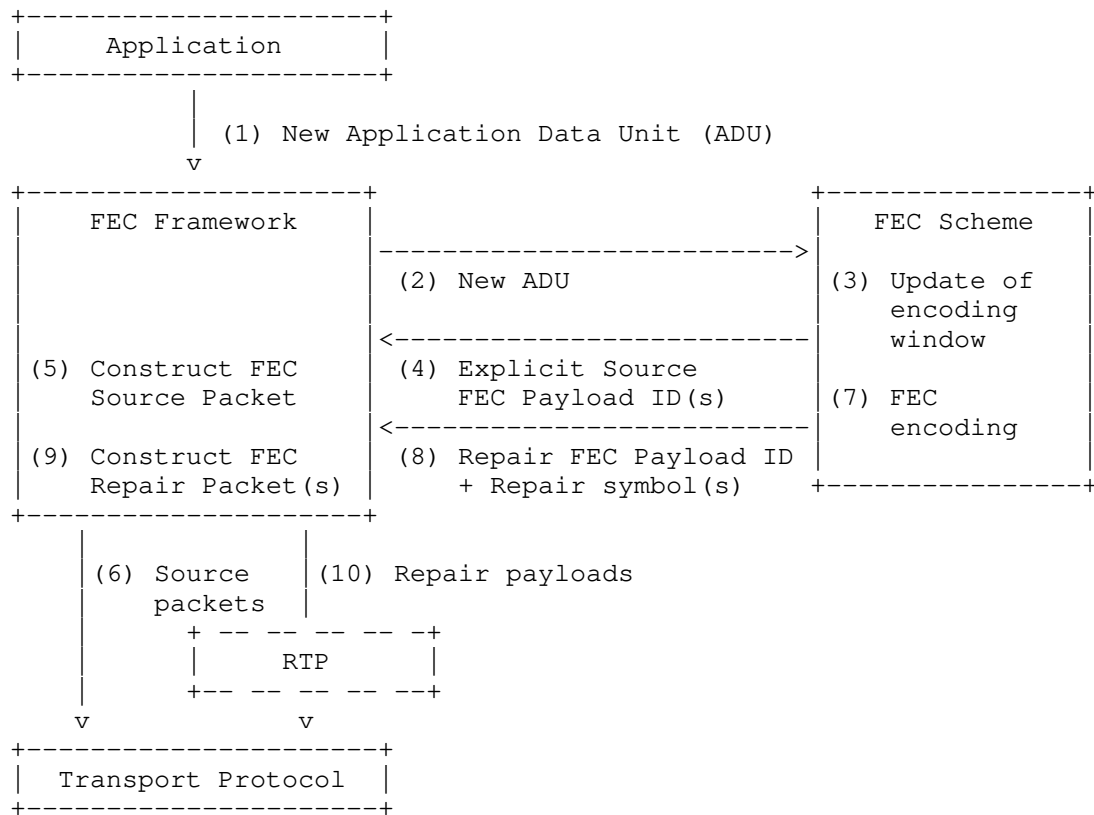


Figure 3: Sender Operation with Sliding Window FEC Codes and RTP Repair Flows

#### 4.3. Receiver Operation with Sliding Window FEC Codes

With a Sliding Window FEC Scheme, the following operations, illustrated in Figure 4 for the generic case (non-RTP repair flows), and in Figure 5 for the case of RTP repair flows. The only differences with respect to block FEC codes lie in steps (4) and (5). Therefore this section does not repeat the other steps of [RFC6363], Section 4.3, "Receiver Operation". The new steps (4) and (5) are:

4. The FEC Scheme uses the received FEC Payload IDs (and derived FEC Source Payload IDs when the Explicit Source FEC Payload ID field is not used) to insert source and repair packets into the decoding window in the right way. If at least one source packet is missing and at least one repair packet has been received, then FEC decoding is attempted to recover missing source payloads. The FEC Scheme determines whether source packets have been lost

and whether enough repair packets have been received to decode any or all of the missing source payloads.

5. The FEC Scheme returns the received and decoded ADUs to the FEC Framework, along with indications of any ADUs that were missing and could not be decoded.

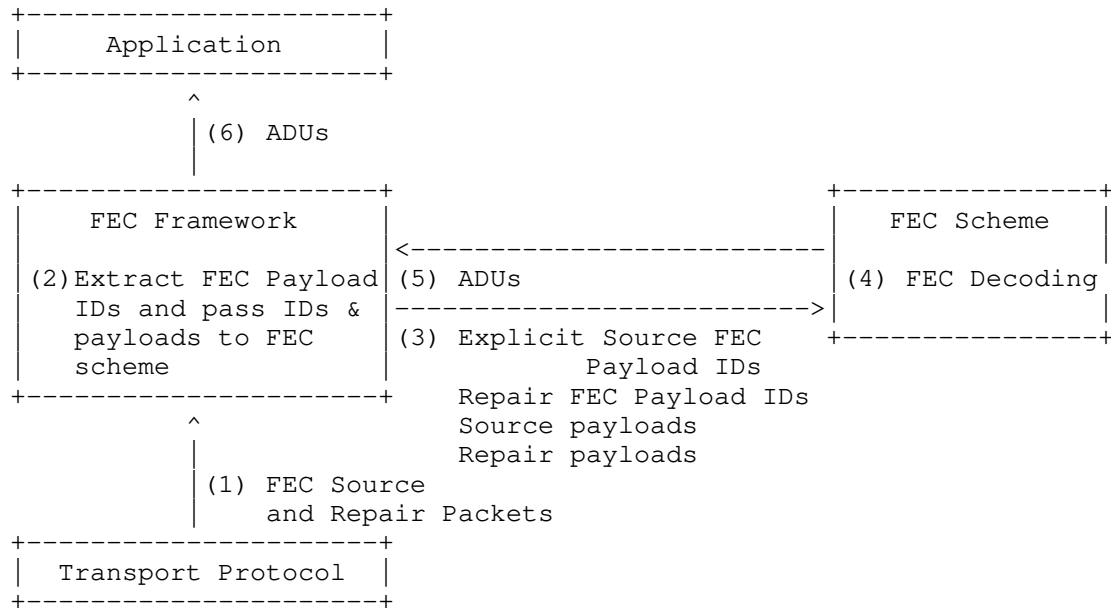


Figure 4: Receiver Operation with Sliding Window FEC Codes



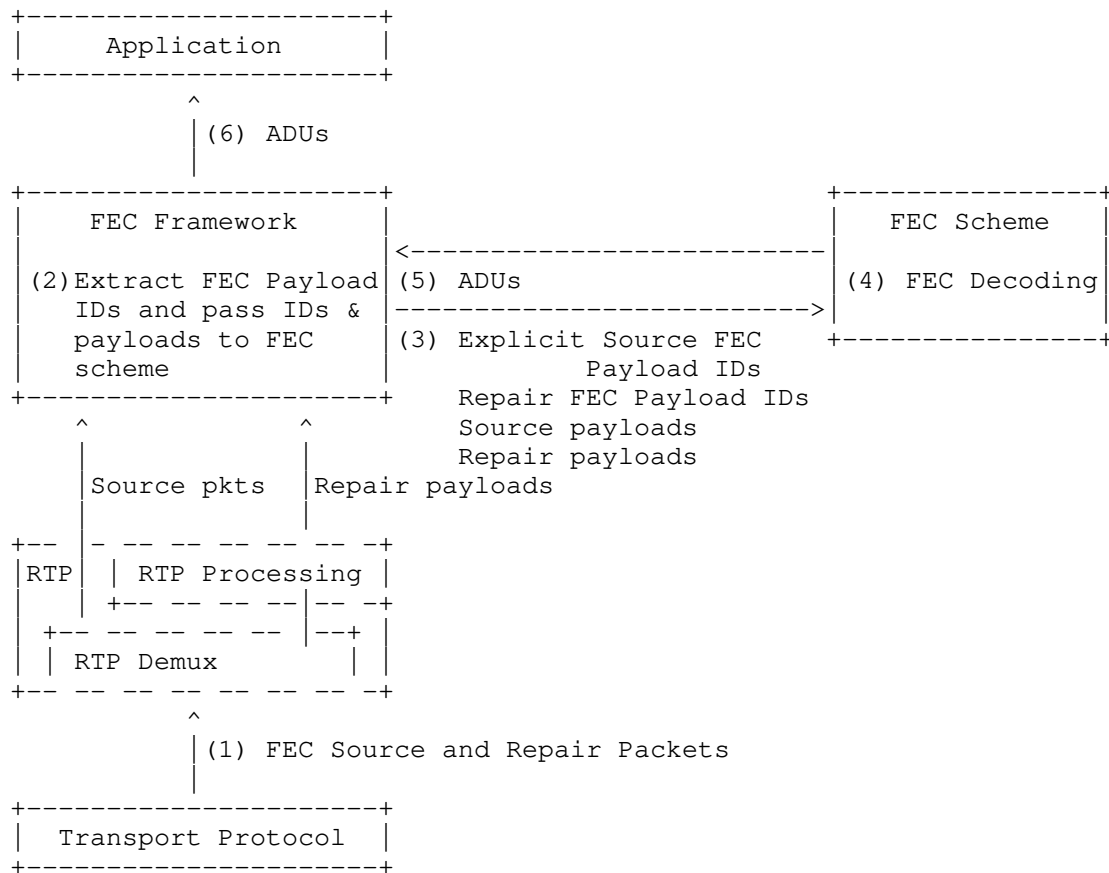


Figure 5: Receiver Operation with Sliding Window FEC Codes and RTP Repair Flows

## 5. Protocol Specification

### 5.1. General

This section discusses the protocol elements for the FEC Framework specific to Sliding Window FEC schemes. The global formats of source data packets (i.e., [RFC6363], Figure 6) and repair data packets (i.e., [RFC6363], Figures 7 and 8) remain the same with Sliding Window FEC codes. They are not repeated here.

## 5.2. FEC Framework Configuration Information

The FEC Framework Configuration Information considerations of [RFC6363], Section 5.5, equally applies to this FECFRAME extension and is not repeated here.

## 5.3. FEC Scheme Requirements

The FEC Scheme requirements of [RFC6363], Section 5.6, mostly apply to this FECFRAME extension and are not repeated here. An exception though is the "full specification of the FEC code", item (4), that is specific to block FEC codes. The following item (4-bis) applies in case of Sliding Window FEC schemes:

### 4-bis. A full specification of the Sliding Window FEC code

This specification MUST precisely define the valid FEC-Scheme-Specific Information values, the valid FEC Payload ID values, and the valid packet payload sizes (where packet payload refers to the space within a packet dedicated to carrying encoding symbols).

Furthermore, given valid values of the FEC-Scheme-Specific Information, a valid Repair FEC Payload ID value, a valid packet payload size, and a valid encoding window (i.e., a set of source symbols), the specification MUST uniquely define the values of the encoding symbol (or symbols) to be included in the repair packet payload with the given Repair FEC Payload ID value.

Additionally, the FEC Scheme associated to a Sliding Window FEC Code:

- o MUST define the relationships between ADUs and the associated source symbols (mapping);
- o MUST define the management of the encoding window that slides over the set of ADUs. Appendix A provides non normative hints about what FEC Scheme designers need to consider;
- o MUST define the management of the decoding window. This usually consists in managing a system of linear equations (in case of a linear FEC code);

## 6. Feedback

The discussion of [RFC6363], Section 6, equally applies to this FECFRAME extension and is not repeated here.

## 7. Transport Protocols

The discussion of [RFC6363], Section 7, equally applies to this FECFRAME extension and is not repeated here.

## 8. Congestion Control

The discussion of [RFC6363], Section 8, equally applies to this FECFRAME extension and is not repeated here.

## 9. Implementation Status

Editor's notes: RFC Editor, please remove this section motivated by RFC 7942 before publishing the RFC. Thanks!

An implementation of FECFRAME extended to Sliding Window codes exists:

- o Organisation: Inria
- o Description: This is an implementation of FECFRAME extended to Sliding Window codes and supporting the RLC FEC Scheme [RLC-ID]. It is based on: (1) a proprietary implementation of FECFRAME, made by Inria and Expway for which interoperability tests have been conducted; and (2) a proprietary implementation of RLC Sliding Window FEC Codes.
- o Maturity: the basic FECFRAME maturity is "production", the FECFRAME extension maturity is "under progress".
- o Coverage: the software implements a subset of [RFC6363], as specialized by the 3GPP eMBMS standard [MBMSTS]. This software also covers the additional features of FECFRAME extended to Sliding Window codes, in particular the RLC FEC Scheme.
- o Licensing: proprietary.
- o Implementation experience: maximum.
- o Information update date: March 2018.
- o Contact: vincent.roca@inria.fr

## 10. Security Considerations

This FECFRAME extension does not add any new security consideration. All the considerations of [RFC6363], Section 9, apply to this document as well. However, for the sake of completeness, the

following goal can be added to the list provided in Section 9.1 "Problem Statement" of [RFC6363]:

- o Attacks can try to corrupt source flows in order to modify the receiver application's behavior (as opposed to just denying service).

## 11. Operations and Management Considerations

This FECFRAME extension does not add any new Operations and Management Consideration. All the considerations of [RFC6363], Section 10, apply to this document as well.

## 12. IANA Considerations

No IANA actions are required for this document.

A FEC Scheme for use with this FEC Framework is identified via its FEC Encoding ID. It is subject to IANA registration in the "FEC Framework (FECFRAME) FEC Encoding IDs" registry. All the rules of [RFC6363], Section 11, apply and are not repeated here.

## 13. Acknowledgments

The authors would like to thank Christer Holmberg, David Black, Gorrry Fairhurst, and Emmanuel Lochin, Spencer Dawkins, Ben Campbell, Benjamin Kaduk, Eric Rescorla, Adam Roach, and Greg Skinner for their valuable feedback on this document. This document being an extension to [RFC6363], the authors would also like to thank Mark Watson as the main author of that RFC.

## 14. References

### 14.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC6363] Watson, M., Begen, A., and V. Roca, "Forward Error Correction (FEC) Framework", RFC 6363, DOI 10.17487/RFC6363, October 2011, <<https://www.rfc-editor.org/info/rfc6363>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

## 14.2. Informative References

- [MBMSTS] 3GPP, "Multimedia Broadcast/Multicast Service (MBMS); Protocols and codecs", 3GPP TS 26.346, March 2009, <<http://ftp.3gpp.org/specs/html-info/26346.htm>>.
- [RFC5052] Watson, M., Luby, M., and L. Vicisano, "Forward Error Correction (FEC) Building Block", RFC 5052, DOI 10.17487/RFC5052, August 2007, <<https://www.rfc-editor.org/info/rfc5052>>.
- [RFC6364] Begen, A., "Session Description Protocol Elements for the Forward Error Correction (FEC) Framework", RFC 6364, DOI 10.17487/RFC6364, October 2011, <<https://www.rfc-editor.org/info/rfc6364>>.
- [RFC6681] Watson, M., Stockhammer, T., and M. Luby, "Raptor Forward Error Correction (FEC) Schemes for FECFRAME", RFC 6681, DOI 10.17487/RFC6681, August 2012, <<https://www.rfc-editor.org/info/rfc6681>>.
- [RFC6816] Roca, V., Cunche, M., and J. Lacan, "Simple Low-Density Parity Check (LDPC) Staircase Forward Error Correction (FEC) Scheme for FECFRAME", RFC 6816, DOI 10.17487/RFC6816, December 2012, <<https://www.rfc-editor.org/info/rfc6816>>.
- [RFC6865] Roca, V., Cunche, M., Lacan, J., Bouabdallah, A., and K. Matsuzono, "Simple Reed-Solomon Forward Error Correction (FEC) Scheme for FECFRAME", RFC 6865, DOI 10.17487/RFC6865, February 2013, <<https://www.rfc-editor.org/info/rfc6865>>.
- [RFC8406] Adamson, B., Adjih, C., Bilbao, J., Firoiu, V., Fitzek, F., Ghanem, S., Lochin, E., Masucci, A., Montpetit, M-J., Pedersen, M., Peralta, G., Roca, V., Ed., Saxena, P., and S. Sivakumar, "Taxonomy of Coding Techniques for Efficient Network Communications", RFC 8406, DOI 10.17487/RFC8406, June 2018, <<https://www.rfc-editor.org/info/rfc8406>>.
- [RLC-ID] Roca, V. and B. Teibi, "Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC) Scheme for FECFRAME", Work in Progress, Transport Area Working Group (TSVWG) draft-ietf-tsvwg-rlc-fec-scheme (Work in Progress), September 2018, <<https://tools.ietf.org/html/draft-ietf-tsvwg-rlc-fec-scheme>>.

## Appendix A. About Sliding Encoding Window Management (informational)

The FEC Framework does not specify the management of the sliding encoding window which is the responsibility of the FEC Scheme. This annex only provides a few informational hints.

Source symbols are added to the sliding encoding window each time a new ADU is available at the sender, after the ADU-to-source-symbol mapping specific to the FEC Scheme.

Source symbols are removed from the sliding encoding window, for instance:

- o after a certain delay, when an "old" ADU of a real-time flow times out. The source symbol retention delay in the sliding encoding window should therefore be initialized according to the real-time features of incoming flow(s) when applicable;
- o once the sliding encoding window has reached its maximum size (there is usually an upper limit to the sliding encoding window size). In that case the oldest symbol is removed each time a new source symbol is added.

Several considerations can impact the management of this sliding encoding window:

- o at the source flows level: real-time constraints can limit the total time source symbols can remain in the encoding window;
- o at the FEC code level: theoretical or practical limitations (e.g., because of computational complexity) can limit the number of source symbols in the encoding window;
- o at the FEC Scheme level: signaling and window management are intrinsically related. For instance, an encoding window composed of a non-sequential set of source symbols requires an appropriate signaling to inform a receiver of the composition of the encoding window, and the associated transmission overhead can limit the maximum encoding window size. On the opposite, an encoding window always composed of a sequential set of source symbols simplifies signaling: providing the identity of the first source symbol plus their number is sufficient, which creates a fixed and relatively small transmission overhead.

Authors' Addresses

Vincent Roca  
INRIA  
Univ. Grenoble Alpes  
France

EMail: [vincent.roca@inria.fr](mailto:vincent.roca@inria.fr)

Ali Begen  
Networked Media  
Konya  
Turkey

EMail: [ali.begen@networked.media](mailto:ali.begen@networked.media)

Transport Area Working Group  
Internet-Draft  
Intended status: Informational  
Expires: 5 September 2022

B. Briscoe, Ed.  
Independent  
K. De Schepper  
Nokia Bell Labs  
M. Bagnulo Braun  
Universidad Carlos III de Madrid  
G. White  
CableLabs  
4 March 2022

Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service:  
Architecture  
draft-ietf-tsvwg-l4s-arch-17

Abstract

This document describes the L4S architecture, which enables Internet applications to achieve Low queuing Latency, Low Loss, and Scalable throughput (L4S). The insight on which L4S is based is that the root cause of queuing delay is in the congestion controllers of senders, not in the queue itself. With the L4S architecture all Internet applications could (but do not have to) transition away from congestion control algorithms that cause substantial queuing delay, to a new class of congestion controls that induce very little queuing, aided by explicit congestion signalling from the network. This new class of congestion controls can provide low latency for capacity-seeking flows, so applications can achieve both high bandwidth and low latency.

The architecture primarily concerns incremental deployment. It defines mechanisms that allow the new class of L4S congestion controls to coexist with 'Classic' congestion controls in a shared network. These mechanisms aim to ensure that the latency and throughput performance using an L4S-compliant congestion controller is usually much better (and rarely worse) than performance would have been using a 'Classic' congestion controller, and that competing flows continuing to use 'Classic' controllers are typically not impacted by the presence of L4S. These characteristics are important to encourage adoption of L4S congestion control algorithms and L4S compliant network elements.

The L4S architecture consists of three components: network support to isolate L4S traffic from classic traffic; protocol features that allow network elements to identify L4S traffic; and host support for L4S congestion controls.



## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 5 September 2022.

## Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Document Roadmap . . . . .	5
2. L4S Architecture Overview . . . . .	5
3. Terminology . . . . .	7
4. L4S Architecture Components . . . . .	9
4.1. Protocol Mechanisms . . . . .	9
4.2. Network Components . . . . .	10
4.3. Host Mechanisms . . . . .	13
5. Rationale . . . . .	15
5.1. Why These Primary Components? . . . . .	15
5.2. What L4S adds to Existing Approaches . . . . .	18
6. Applicability . . . . .	21
6.1. Applications . . . . .	21
6.2. Use Cases . . . . .	22
6.3. Applicability with Specific Link Technologies . . . . .	24

6.4.	Deployment Considerations . . . . .	24
6.4.1.	Deployment Topology . . . . .	25
6.4.2.	Deployment Sequences . . . . .	26
6.4.3.	L4S Flow but Non-ECN Bottleneck . . . . .	28
6.4.4.	L4S Flow but Classic ECN Bottleneck . . . . .	29
6.4.5.	L4S AQM Deployment within Tunnels . . . . .	29
7.	IANA Considerations (to be removed by RFC Editor) . . . . .	30
8.	Security Considerations . . . . .	30
8.1.	Traffic Rate (Non-)Policing . . . . .	30
8.2.	'Latency Friendliness' . . . . .	31
8.3.	Interaction between Rate Policing and L4S . . . . .	33
8.4.	ECN Integrity . . . . .	34
8.5.	Privacy Considerations . . . . .	34
9.	Acknowledgements . . . . .	35
10.	Informative References . . . . .	35
	Authors' Addresses . . . . .	45

## 1. Introduction

At any one time, it is increasingly common for all of the traffic in a bottleneck link (e.g. a household's Internet access) to come from applications that prefer low delay: interactive Web, Web services, voice, conversational video, interactive video, interactive remote presence, instant messaging, online gaming, remote desktop, cloud-based applications and video-assisted remote control of machinery and industrial processes. In the last decade or so, much has been done to reduce propagation delay by placing caches or servers closer to users. However, queuing remains a major, albeit intermittent, component of latency. For instance spikes of hundreds of milliseconds are not uncommon, even with state-of-the-art active queue management (AQM) [COBALT], [DOCSIS3AQM]. Queuing in access network bottlenecks is typically configured to cause overall network delay to roughly double during a long-running flow, relative to expected base (unloaded) path delay [BufferSize]. Low loss is also important because, for interactive applications, losses translate into even longer retransmission delays.

It has been demonstrated that, once access network bit rates reach levels now common in the developed world, increasing capacity offers diminishing returns if latency (delay) is not addressed [Dukkipati06], [Rajiullah15]. Therefore, the goal is an Internet service with very Low queueing Latency, very Low Loss and Scalable throughput (L4S). Very low queueing latency means less than 1 millisecond (ms) on average and less than about 2 ms at the 99th percentile. This document describes the L4S architecture for achieving these goals.

Differentiated services (Diffserv) offers Expedited Forwarding (EF [RFC3246]) for some packets at the expense of others, but this makes no difference when all (or most) of the traffic at a bottleneck at any one time requires low latency. In contrast, L4S still works well when all traffic is L4S - a service that gives without taking needs none of the configuration or management baggage (traffic policing, traffic contracts) associated with favouring some traffic flows over others.

Queuing delay degrades performance intermittently [Hohlfeld14]. It occurs when a large enough capacity-seeking (e.g. TCP) flow is running alongside the user's traffic in the bottleneck link, which is typically in the access network. Or when the low latency application is itself a large capacity-seeking or adaptive rate (e.g. interactive video) flow. At these times, the performance improvement from L4S must be sufficient that network operators will be motivated to deploy it.

Active Queue Management (AQM) is part of the solution to queuing under load. AQM improves performance for all traffic, but there is a limit to how much queuing delay can be reduced by solely changing the network; without addressing the root of the problem.

The root of the problem is the presence of standard TCP congestion control (Reno [RFC5681]) or compatible variants (e.g. TCP Cubic [RFC8312]). We shall use the term 'Classic' for these Reno-friendly congestion controls. Classic congestion controls induce relatively large saw-tooth-shaped excursions up the queue and down again, which have been growing as flow rate scales [RFC3649]. So if a network operator naively attempts to reduce queuing delay by configuring an AQM to operate at a shallower queue, a Classic congestion control will significantly underutilize the link at the bottom of every saw-tooth.

It has been demonstrated that if the sending host replaces a Classic congestion control with a 'Scalable' alternative, when a suitable AQM is deployed in the network the performance under load of all the above interactive applications can be significantly improved. For instance, queuing delay under heavy load with the example DCTCP/DualQ solution cited below on a DSL or Ethernet link is roughly 1 to 2 milliseconds at the 99th percentile without losing link utilization [DualPI2Linux], [DCTtH19] (for other link types, see Section 6.3). This compares with 5-20 ms on average with a Classic congestion control and current state-of-the-art AQMs such as FQ-CoDel [RFC8290], PIE [RFC8033] or DOCSIS PIE [RFC8034] and about 20-30 ms at the 99th percentile [DualPI2Linux].

L4S is designed for incremental deployment. It is possible to deploy the L4S service at a bottleneck link alongside the existing best efforts service [DualPI2Linux] so that unmodified applications can start using it as soon as the sender's stack is updated. Access networks are typically designed with one link as the bottleneck for each site (which might be a home, small enterprise or mobile device), so deployment at either or both ends of this link should give nearly all the benefit in the respective direction. With some transport protocols, namely TCP and SCTP, the sender has to check for suitably updated receiver feedback, whereas with more recent transport protocols such as QUIC and DCCP, all receivers have always been suitable.

This document presents the L4S architecture, by describing and justifying the component parts and how they interact to provide the scalable, low latency, low loss Internet service. It also details the approach to incremental deployment, as briefly summarized above.

### 1.1. Document Roadmap

This document describes the L4S architecture in three passes. First this brief overview gives the very high level idea and states the main components with minimal rationale. This is only intended to give some context for the terminology definitions that follow in Section 3, and to explain the structure of the rest of the document. Then Section 4 goes into more detail on each component with some rationale, but still mostly stating what the architecture is, rather than why. Finally Section 5 justifies why each element of the solution was chosen (Section 5.1) and why these choices were different from other solutions (Section 5.2).

Having described the architecture, Section 6 clarifies its applicability; that is, the applications and use-cases that motivated the design, the challenges applying the architecture to various link technologies, and various incremental deployment models: including the two main deployment topologies, different sequences for incremental deployment and various interactions with pre-existing approaches. The document ends with the usual tail pieces, including extensive discussion of traffic policing and other security considerations Section 8.

## 2. L4S Architecture Overview

Below we outline the three main components to the L4S architecture; 1) the scalable congestion control on the sending host; 2) the AQM at the network bottleneck; and 3) the protocol between them.

But first, the main point to grasp is that low latency is not provided by the network - low latency results from the careful behaviour of the scalable congestion controllers used by L4S senders. The network does have a role - primarily to isolate the low latency of the carefully behaving L4S traffic from the higher queuing delay needed by traffic with pre-existing Classic behaviour. The network also alters the way it signals queue growth to the transport - It uses the Explicit Congestion Notification (ECN) protocol, but it signals the very start of queue growth - immediately without the smoothing delay typical of Classic AQMs. Because ECN support is essential for L4S, senders use the ECN field as the protocol to identify to the network which packets are L4S and which are Classic.

- 1) Host: Scalable congestion controls already exist. They solve the scaling problem with Classic congestion controls, such as Reno or Cubic. Because flow rate has scaled since TCP congestion control was first designed in 1988, assuming the flow lasts long enough, it now takes hundreds of round trips (and growing) to recover after a congestion signal (whether a loss or an ECN mark) as shown in the examples in Section 5.1 and [RFC3649]. Therefore control of queuing and utilization becomes very slack, and the slightest disturbances (e.g. from new flows starting) prevent a high rate from being attained.

With a scalable congestion control, the average time from one congestion signal to the next (the recovery time) remains invariant as the flow rate scales, all other factors being equal. This maintains the same degree of control over queueing and utilization whatever the flow rate, as well as ensuring that high throughput is more robust to disturbances. The scalable control used most widely (in controlled environments) is Data Center TCP (DCTCP [RFC8257]), which has been implemented and deployed in Windows Server Editions (since 2012), in Linux and in FreeBSD. Although DCTCP as-is functions well over wide-area round trip times, most implementations lack certain safety features that would be necessary for use outside controlled environments like data centres (see Section 6.4.3). So scalable congestion control needs to be implemented in TCP and other transport protocols (QUIC, SCTP, RTP/RTCP, RMCAT, etc.). Indeed, between the present document being drafted and published, the following scalable congestion controls were implemented: TCP Prague [PragueLinux], QUIC Prague, an L4S variant of the RMCAT SCReAM controller [SCReAM] and the L4S ECN part of BBRv2 [BBRv2] intended for TCP and QUIC transports.

- 2) Network: L4S traffic needs to be isolated from the queuing

latency of Classic traffic. One queue per application flow (FQ) is one way to achieve this, e.g. FQ-CoDel [RFC8290]. However, just two queues is sufficient and does not require inspection of transport layer headers in the network, which is not always possible (see Section 5.2). With just two queues, it might seem impossible to know how much capacity to schedule for each queue without inspecting how many flows at any one time are using each. And it would be undesirable to arbitrarily divide access network capacity into two partitions. The Dual Queue Coupled AQM was developed as a minimal complexity solution to this problem. It acts like a 'semi-permeable' membrane that partitions latency but not bandwidth. As such, the two queues are for transition from Classic to L4S behaviour, not bandwidth prioritization.

Section 4 gives a high level explanation of how the per-flow-queue (FQ) and DualQ variants of L4S work, and [I-D.ietf-tsvwg-aqm-dualq-coupled] gives a full explanation of the DualQ Coupled AQM framework. A specific marking algorithm is not mandated for L4S AQMs. Appendices of [I-D.ietf-tsvwg-aqm-dualq-coupled] give non-normative examples that have been implemented and evaluated, and give recommended default parameter settings. It is expected that L4S experiments will improve knowledge of parameter settings and whether the set of marking algorithms needs to be limited.

- 3) Protocol: A host needs to distinguish L4S and Classic packets with an identifier so that the network can classify them into their separate treatments. The L4S identifier spec. [I-D.ietf-tsvwg-ecn-l4s-id] concludes that all alternatives involve compromises, but the ECT(1) and CE codepoints of the ECN field represent a workable solution. As already explained, the network also uses ECN to immediately signal the very start of queue growth to the transport.

### 3. Terminology

Note: The following definitions are copied from the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id] for convenience. If there are accidental differences, those in [I-D.ietf-tsvwg-ecn-l4s-id] take precedence.

**Classic Congestion Control:** A congestion control behaviour that can co-exist with standard Reno [RFC5681] without causing significantly negative impact on its flow rate [RFC5033]. The scaling problem with Classic congestion control is explained, with examples, in Section 5.1 and in [RFC3649].

**Scalable Congestion Control:** A congestion control where the average

time from one congestion signal to the next (the recovery time) remains invariant as the flow rate scales, all other factors being equal. For instance, DCTCP averages 2 congestion signals per round-trip whatever the flow rate, as do other recently developed scalable congestion controls, e.g. Relentless TCP [Mathis09], TCP Prague [I-D.briscoe-iccrp-prague-congestion-control], [PragueLinux], BBRv2 [BBRv2], [I-D.cardwell-iccrp-bbr-congestion-control] and the L4S variant of SCReAM for real-time media [SCReAM], [RFC8298]). See Section 4.3 of [I-D.ietf-tsvwg-ecn-l4s-id] for more explanation.

**Classic service:** The Classic service is intended for all the congestion control behaviours that co-exist with Reno [RFC5681] (e.g. Reno itself, Cubic [RFC8312], Compound [I-D.sridharan-tcpm-ctcp], TFRC [RFC5348]). The term 'Classic queue' means a queue providing the Classic service.

**Low-Latency, Low-Loss Scalable throughput (L4S) service:** The 'L4S' service is intended for traffic from scalable congestion control algorithms, such as the Prague congestion control [I-D.briscoe-iccrp-prague-congestion-control], which was derived from DCTCP [RFC8257]. The L4S service is for more general traffic than just TCP Prague -- it allows the set of congestion controls with similar scaling properties to Prague to evolve, such as the examples listed above (Relentless, SCReAM). The term 'L4S queue' means a queue providing the L4S service.

The terms Classic or L4S can also qualify other nouns, such as 'queue', 'codepoint', 'identifier', 'classification', 'packet', 'flow'. For example: an L4S packet means a packet with an L4S identifier sent from an L4S congestion control.

Both Classic and L4S services can cope with a proportion of unresponsive or less-responsive traffic as well, but in the L4S case its rate has to be smooth enough or low enough not build a queue (e.g. DNS, VoIP, game sync datagrams, etc).

**Reno-friendly:** The subset of Classic traffic that is friendly to the standard Reno congestion control defined for TCP in [RFC5681]. The TFRC spec. [RFC5348] indirectly implies that 'friendly' is defined as "generally within a factor of two of the sending rate of a TCP flow under the same conditions". Reno-friendly is used here in place of 'TCP-friendly', given the latter has become imprecise, because the TCP protocol is now used with so many different congestion control behaviours, and Reno is used in non-TCP transports such as QUIC [RFC9000].

**Classic ECN:** The original Explicit Congestion Notification (ECN)

protocol [RFC3168], which requires ECN signals to be treated as equivalent to drops, both when generated in the network and when responded to by the sender.

L4S uses the ECN field as an identifier [I-D.ietf-tsvwg-ecn-l4s-id] with the names for the four codepoints of the 2-bit IP-ECN field unchanged from those defined in the ECN spec [RFC3168]: Not ECT, ECT(0), ECT(1) and CE, where ECT stands for ECN-Capable Transport and CE stands for Congestion Experienced. A packet marked with the CE codepoint is termed 'ECN-marked' or sometimes just 'marked' where the context makes ECN obvious.

Site: A home, mobile device, small enterprise or campus, where the network bottleneck is typically the access link to the site. Not all network arrangements fit this model but it is a useful, widely applicable generalization.

#### 4. L4S Architecture Components

The L4S architecture is composed of the elements in the following three subsections.

##### 4.1. Protocol Mechanisms

The L4S architecture involves: a) unassignment of an identifier; b) reassignment of the same identifier; and c) optional further identifiers:

- a. An essential aspect of a scalable congestion control is the use of explicit congestion signals. 'Classic' ECN [RFC3168] requires an ECN signal to be treated as equivalent to drop, both when it is generated in the network and when it is responded to by hosts. L4S needs networks and hosts to support a more fine-grained meaning for each ECN signal that is less severe than a drop, so that the L4S signals:

- \* can be much more frequent;
- \* can be signalled immediately, without the significant delay required to smooth out fluctuations in the queue.



To enable L4S, the standards track Classic ECN spec. [RFC3168] has had to be updated to allow L4S packets to depart from the 'equivalent to drop' constraint. [RFC8311] is a standards track update to relax specific requirements in RFC 3168 (and certain other standards track RFCs), which clears the way for the experimental changes proposed for L4S. [RFC8311] also reclassifies the original experimental assignment of the ECT(1) codepoint as an ECN nonce [RFC3540] as historic.

- b. [I-D.ietf-tsvwg-ecn-l4s-id] specifies that ECT(1) is used as the identifier to classify L4S packets into a separate treatment from Classic packets. This satisfies the requirement for identifying an alternative ECN treatment in [RFC4774].

The CE codepoint is used to indicate Congestion Experienced by both L4S and Classic treatments. This raises the concern that a Classic AQM earlier on the path might have marked some ECT(0) packets as CE. Then these packets will be erroneously classified into the L4S queue. Appendix B of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id] explains why five unlikely eventualities all have to coincide for this to have any detrimental effect, which even then would only involve a vanishingly small likelihood of a spurious retransmission.

- c. A network operator might wish to include certain unresponsive, non-L4S traffic in the L4S queue if it is deemed to be smoothly enough paced and low enough rate not to build a queue. For instance, VoIP, low rate datagrams to sync online games, relatively low rate application-limited traffic, DNS, LDAP, etc. This traffic would need to be tagged with specific identifiers, e.g. a low latency Diffserv Codepoint such as Expedited Forwarding (EF [RFC3246]), Non-Queue-Building (NQB [I-D.ietf-tsvwg-nqb]), or operator-specific identifiers.

#### 4.2. Network Components

The L4S architecture aims to provide low latency without the need for per-flow operations in network components. Nonetheless, the architecture does not preclude per-flow solutions. The following bullets describe the known arrangements: a) the DualQ Coupled AQM with an L4S AQM in one queue coupled from a Classic AQM in the other; b) Per-Flow Queues with an instance of a Classic and an L4S AQM in each queue; c) Dual queues with per-flow AQMs, but no per-flow queues:

- a. The Dual Queue Coupled AQM (illustrated in Figure 1) achieves the 'semi-permeable' membrane property mentioned earlier as follows:

- \* Latency isolation: Two separate queues are used to isolate L4S queuing delay from the larger queue that Classic traffic needs to maintain full utilization.
- \* Bandwidth pooling: The two queues act as if they are a single pool of bandwidth in which flows of either type get roughly equal throughput without the scheduler needing to identify any flows. This is achieved by having an AQM in each queue, but the Classic AQM provides a congestion signal to both queues in a manner that ensures a consistent response from the two classes of congestion control. Specifically, the Classic AQM generates a drop/mark probability based on congestion in its own queue, which it uses both to drop/mark packets in its own queue and to affect the marking probability in the L4S queue. The strength of the coupling of the congestion signalling between the two queues is enough to make the L4S flows slow down to leave the right amount of capacity for the Classic flows (as they would if they were the same type of traffic sharing the same queue).

Then the scheduler can serve the L4S queue with priority (denoted by the '1' on the higher priority input), because the L4S traffic isn't offering up enough traffic to use all the priority that it is given. Therefore:

- \* for latency isolation on short time-scales (sub-round-trip) the prioritization of the L4S queue protects its low latency by allowing bursts to dissipate quickly;
- \* but for bandwidth pooling on longer time-scales (round-trip and longer) the Classic queue creates an equal and opposite pressure against the L4S traffic to ensure that neither has priority when it comes to bandwidth - the tension between prioritizing L4S and coupling the marking from the Classic AQM results in approximate per-flow fairness.

To protect against unresponsive traffic taking advantage of the prioritization of the L4S queue and starving the Classic queue, it is advisable for the priority to be conditional, not strict (see Appendix A of the DualQ spec [I-D.ietf-tsvwg-aqm-dualq-coupled]).

When there is no Classic traffic, the L4S queue's own AQM comes into play. It starts congestion marking with a very shallow queue, so L4S traffic maintains very low queuing delay.

If either queue becomes persistently overloaded, drop of ECN-capable packets is introduced, as recommended in Section 7 of the ECN spec [RFC3168] and Section 4.2.1 of the AQM recommendations [RFC7567]. Then both queues introduce the same level of drop (not shown in the figure).

The Dual Queue Coupled AQM has been specified as generically as possible [I-D.ietf-tsvwg-aqm-dualq-coupled] without specifying the particular AQMs to use in the two queues so that designers are free to implement diverse ideas. Informational appendices in that draft give pseudocode examples of two different specific AQM approaches: one called DualPI2 (pronounced Dual PI Squared) [DualPI2Linux] that uses the PI2 variant of PIE, and a zero-config variant of RED called Curvy RED. A DualQ Coupled AQM based on PIE has also been specified and implemented for Low Latency DOCSIS [DOCSIS3.1].

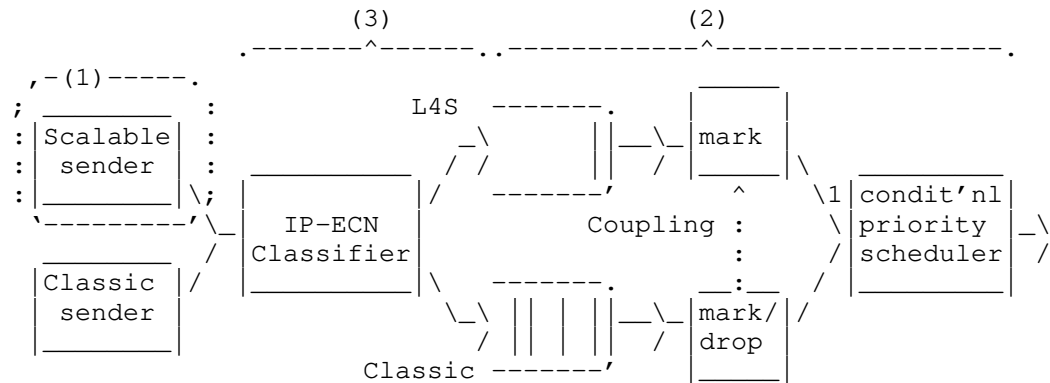


Figure 1: Components of an L4S DualQ Coupled AQM Solution: 1) Scalable Sending Host; 2) Isolation in separate network queues; and 3) Packet Identification Protocol

- b. **Per-Flow Queues and AQMs:** A scheduler with per-flow queues such as FQ-CoDel or FQ-PIE can be used for L4S. For instance within each queue of an FQ-CoDel system, as well as a CoDel AQM, there is typically also the option of ECN marking at an immediate (unsmoothed) shallow threshold to support use in data centres (see Sec.5.2.7 of the FQ-CoDel spec [RFC8290]). In Linux, this has been modified so that the shallow threshold can be solely applied to ECT(1) packets [FQ\_CoDel\_Thresh]. Then if there is a flow of non-ECN or ECT(0) packets in the per-flow-queue, the Classic AQM (e.g. CoDel) is applied; while if there is a flow of ECT(1) packets in the queue, the shallower (typically sub-millisecond) threshold is applied. In addition, ECT(0) and not-ECT packets could potentially be classified into a separate flow-queue from ECT(1) and CE packets to avoid them mixing if they share a common flow-identifier (e.g. in a VPN).
- c. **Dual-queues, but per-flow AQMs:** It should also be possible to use dual queues for isolation, but with per-flow marking to control flow-rates (instead of the coupled per-queue marking of the Dual Queue Coupled AQM). One of the two queues would be for isolating L4S packets, which would be classified by the ECN codepoint. Flow rates could be controlled by flow-specific marking. The policy goal of the marking could be to differentiate flow rates (e.g. [Nadas20], which requires additional signalling of a per-flow 'value'), or to equalize flow-rates (perhaps in a similar way to Approx Fair CoDel [AFCD], [I-D.morton-tsvwg-codel-approx-fair], but with two queues not one).

Note that whenever the term 'DualQ' is used loosely without saying whether marking is per-queue or per-flow, it means a dual queue AQM with per-queue marking.

#### 4.3. Host Mechanisms

The L4S architecture includes two main mechanisms in the end host that we enumerate next:

- a. **Scalable Congestion Control at the sender:** Section 2 defines a scalable congestion control as one where the average time from one congestion signal to the next (the recovery time) remains invariant as the flow rate scales, all other factors being equal. Data Center TCP is the most widely used example. It has been documented as an informational record of the protocol currently in use in controlled environments [RFC8257]. A draft list of safety and performance improvements for a scalable congestion control to be usable on the public Internet has been drawn up (the so-called 'Prague L4S requirements' in Appendix A of

[I-D.ietf-tsvwg-ecn-l4s-id]). The subset that involve risk of harm to others have been captured as normative requirements in Section 4 of [I-D.ietf-tsvwg-ecn-l4s-id]. TCP Prague [I-D.briscoe-iccrp-prague-congestion-control] has been implemented in Linux as a reference implementation to address these requirements [PragueLinux].

Transport protocols other than TCP use various congestion controls that are designed to be friendly with Reno. Before they can use the L4S service, they will need to be updated to implement a scalable congestion response, which they will have to indicate by using the ECT(1) codepoint. Scalable variants are under consideration for more recent transport protocols, e.g. QUIC, and the L4S ECN part of BBRv2 [BBRv2], [I-D.cardwell-iccrp-bbr-congestion-control] is a scalable congestion control intended for the TCP and QUIC transports, amongst others. Also an L4S variant of the RMCAT SCReAM controller [RFC8298] has been implemented [SCReAM] for media transported over RTP.

Section 4.3 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id] defines scalable congestion control in more detail, and specifies that requirements that an L4S scalable congestion control has to comply with.

- b. The ECN feedback in some transport protocols is already sufficiently fine-grained for L4S (specifically DCCP [RFC4340] and QUIC [RFC9000]). But others either require update or are in the process of being updated:
  - \* For the case of TCP, the feedback protocol for ECN embeds the assumption from Classic ECN [RFC3168] that an ECN mark is equivalent to a drop, making it unusable for a scalable TCP. Therefore, the implementation of TCP receivers will have to be upgraded [RFC7560]. Work to standardize and implement more accurate ECN feedback for TCP (AccECN) is in progress [I-D.ietf-tcpm-accurate-ecn], [PragueLinux].
  - \* ECN feedback is only roughly sketched in an appendix of the SCTP specification [RFC4960]. A fuller specification has been proposed in a long-expired draft [I-D.stewart-tsvwg-sctpecn], which would need to be implemented and deployed before SCTCP could support L4S.
  - \* For RTP, sufficient ECN feedback was defined in [RFC6679], but [RFC8888] defines the latest standards track improvements.

## 5. Rationale

### 5.1. Why These Primary Components?

Explicit congestion signalling (protocol): Explicit congestion signalling is a key part of the L4S approach. In contrast, use of drop as a congestion signal creates a tension because drop is both an impairment (less would be better) and a useful signal (more would be better):

- \* Explicit congestion signals can be used many times per round trip, to keep tight control, without any impairment. Under heavy load, even more explicit signals can be applied so the queue can be kept short whatever the load. In contrast, Classic AQMs have to introduce very high packet drop at high load to keep the queue short. By using ECN, an L4S congestion control's sawtooth reduction can be smaller and therefore return to the operating point more often, without worrying that more sawteeth will cause more signals. The consequent smaller amplitude sawteeth fit between an empty queue and a very shallow marking threshold (~1 ms in the public Internet), so queue delay variation can be very low, without risk of under-utilization.
- \* Explicit congestion signals can be emitted immediately to track fluctuations of the queue. L4S shifts smoothing from the network to the host. The network doesn't know the round trip times of any of the flows. So if the network is responsible for smoothing (as in the Classic approach), it has to assume a worst case RTT, otherwise long RTT flows would become unstable. This delays Classic congestion signals by 100-200 ms. In contrast, each host knows its own round trip time. So, in the L4S approach, the host can smooth each flow over its own RTT, introducing no more soothing delay than strictly necessary (usually only a few milliseconds). A host can also choose not to introduce any smoothing delay if appropriate, e.g. during flow start-up.

Neither of the above are feasible if explicit congestion signalling has to be considered 'equivalent to drop' (as was required with Classic ECN [RFC3168]), because drop is an impairment as well as a signal. So drop cannot be excessively frequent, and drop cannot be immediate, otherwise too many drops would turn out to have been due to only a transient fluctuation in the queue that would not have warranted dropping a packet in hindsight. Therefore, in an L4S AQM, the L4S queue uses a new L4S variant of ECN that is not equivalent to drop (see section 5.2 of

the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id]), while the Classic queue uses either Classic ECN [RFC3168] or drop, which are equivalent to each other.

Before Classic ECN was standardized, there were various proposals to give an ECN mark a different meaning from drop. However, there was no particular reason to agree on any one of the alternative meanings, so 'equivalent to drop' was the only compromise that could be reached. RFC 3168 contains a statement that:

"An environment where all end nodes were ECN-Capable could allow new criteria to be developed for setting the CE codepoint, and new congestion control mechanisms for end-node reaction to CE packets. However, this is a research issue, and as such is not addressed in this document."

Latency isolation (network): L4S congestion controls keep queue delay low whereas Classic congestion controls need a queue of the order of the RTT to avoid under-utilization. One queue cannot have two lengths, therefore L4S traffic needs to be isolated in a separate queue (e.g. DualQ) or queues (e.g. FQ).

Coupled congestion notification: Coupling the congestion notification between two queues as in the DualQ Coupled AQM is not necessarily essential, but it is a simple way to allow senders to determine their rate, packet by packet, rather than be overridden by a network scheduler. An alternative is for a network scheduler to control the rate of each application flow (see discussion in Section 5.2).

L4S packet identifier (protocol): Once there are at least two treatments in the network, hosts need an identifier at the IP layer to distinguish which treatment they intend to use.

Scalable congestion notification: A scalable congestion control in the host keeps the signalling frequency from the network high whatever the flow rate, so that queue delay variations can be small when conditions are stable, and rate can track variations in available capacity as rapidly as possible otherwise.

Low loss: Latency is not the only concern of L4S. The 'Low Loss' part of the name denotes that L4S generally achieves zero congestion loss due to its use of ECN. Otherwise, loss would itself cause delay, particularly for short flows, due to retransmission delay [RFC2884].

Scalable throughput: The "Scalable throughput" part of the name

denotes that the per-flow throughput of scalable congestion controls should scale indefinitely, avoiding the imminent scaling problems with Reno-friendly congestion control algorithms [RFC3649]. It was known when TCP congestion avoidance was first developed in 1988 that it would not scale to high bandwidth-delay products (see footnote 6 in [TCP-CA]). Today, regular broadband flow rates over WAN distances are already beyond the scaling range of Classic Reno congestion control. So 'less unscalable' Cubic [RFC8312] and Compound [I-D.sridharan-tcpm-ctcp] variants of TCP have been successfully deployed. However, these are now approaching their scaling limits.

For instance, we will consider a scenario with a maximum RTT of 30 ms at the peak of each sawtooth. As Reno packet rate scales 8x from 1,250 to 10,000 packet/s (from 15 to 120 Mb/s with 1500 B packets), the time to recover from a congestion event rises proportionately by 8x as well, from 422 ms to 3.38 s. It is clearly problematic for a congestion control to take multiple seconds to recover from each congestion event. Cubic [RFC8312] was developed to be less unscalable, but it is approaching its scaling limit; with the same max RTT of 30 ms, at 120 Mb/s Cubic is still fully in its Reno-friendly mode, so it takes about 4.3 s to recover. However, once the flow rate scales by 8x again to 960 Mb/s it enters true Cubic mode, with a recovery time of 12.2 s. From then on, each further scaling by 8x doubles Cubic's recovery time (because the cube root of 8 is 2), e.g. at 7.68 Gb/s the recovery time is 24.3 s. In contrast a scalable congestion control like DCTCP or TCP Prague induces 2 congestion signals per round trip on average, which remains invariant for any flow rate, keeping dynamic control very tight.

For a feel of where the global average lone-flow download sits on this scale at the time of writing (2021), according to [BDPdata] globally averaged fixed access capacity was 103 Mb/s in 2020 and averaged base RTT to a CDN was 25-34ms in 2019. Averaging of per-country data was weighted by Internet user population (data collected globally is necessarily of variable quality, but the paper does double-check that the outcome compares well against a second source). So a lone CUBIC flow would at best take about 200 round trips (5 s) to recover from each of its sawtooth reductions, if the flow even lasted that long. This is described as 'at best' because it assume everyone uses an AQM, whereas in reality most users still have a (probably bloated) tail-drop buffer. In the tail-drop case, likely average recovery time would be at least 4x 5 s, if not more, because RTT under load would be at least double that of an AQM, and recovery time depends on the square of RTT.



Although work on scaling congestion controls tends to start with TCP as the transport, the above is not intended to exclude other transports (e.g. SCTP, QUIC) or less elastic algorithms (e.g. RMCAT), which all tend to adopt the same or similar developments.

## 5.2. What L4S adds to Existing Approaches

All the following approaches address some part of the same problem space as L4S. In each case, it is shown that L4S complements them or improves on them, rather than being a mutually exclusive alternative:

**Diffserv:** Diffserv addresses the problem of bandwidth apportionment for important traffic as well as queuing latency for delay-sensitive traffic. Of these, L4S solely addresses the problem of queuing latency. Diffserv will still be necessary where important traffic requires priority (e.g. for commercial reasons, or for protection of critical infrastructure traffic) - see [I-D.briscoe-tsvwg-l4s-diffserv]. Nonetheless, the L4S approach can provide low latency for all traffic within each Diffserv class (including the case where there is only the one default Diffserv class).

Also, Diffserv can only provide a latency benefit if a small subset of the traffic on a bottleneck link requests low latency. As already explained, it has no effect when all the applications in use at one time at a single site (home, small business or mobile device) require low latency. In contrast, because L4S works for all traffic, it needs none of the management baggage (traffic policing, traffic contracts) associated with favouring some packets over others. This lack of management baggage ought to give L4S a better chance of end-to-end deployment.

In particular, because networks tend not to trust end systems to identify which packets should be favoured over others, where networks assign packets to Diffserv classes they tend to use packet inspection of application flow identifiers or deeper inspection of application signatures. Thus, nowadays, Diffserv doesn't always sit well with encryption of the layers above IP [RFC8404]. So users have to choose between privacy and QoS.

As with Diffserv, the L4S identifier is in the IP header. But, in contrast to Diffserv, the L4S identifier does not convey a want or a need for a certain level of quality. Rather, it promises a certain behaviour (scalable congestion response), which networks can objectively verify if they need to. This is because low delay depends on collective host behaviour, whereas bandwidth priority depends on network behaviour.

State-of-the-art AQMs: AQMs such as PIE and FQ-CoDel give a significant reduction in queuing delay relative to no AQM at all. L4S is intended to complement these AQMs, and should not distract from the need to deploy them as widely as possible. Nonetheless, AQMs alone cannot reduce queuing delay too far without significantly reducing link utilization, because the root cause of the problem is on the host - where Classic congestion controls use large saw-toothed rate variations. The L4S approach resolves this tension between delay and utilization by enabling hosts to minimize the amplitude of their sawteeth. A single-queue Classic AQM is not sufficient to allow hosts to use small sawteeth for two reasons: i) smaller sawteeth would not get lower delay in an AQM designed for larger amplitude Classic sawteeth, because a queue can only have one length at a time; and ii) much smaller sawteeth implies much more frequent sawteeth, so L4S flows would drive a Classic AQM into a high level of ECN-marking, which would appear as heavy congestion to Classic flows, which in turn would greatly reduce their rate as a result (see Section 6.4.4).

Per-flow queuing or marking: Similarly, per-flow approaches such as FQ-CoDel or Approx Fair CoDel [AFCD] are not incompatible with the L4S approach. However, per-flow queuing alone is not enough - it only isolates the queuing of one flow from others; not from itself. Per-flow implementations need to have support for scalable congestion control added, which has already been done for FQ-CoDel in Linux (see Sec.5.2.7 of [RFC8290] and [FQ\_CoDel\_Thresh]). Without this simple modification, per-flow AQMs like FQ-CoDel would still not be able to support applications that need both very low delay and high bandwidth, e.g. video-based control of remote procedures, or interactive cloud-based video (see Note 1 below).

Although per-flow techniques are not incompatible with L4S, it is important to have the DualQ alternative. This is because handling end-to-end (layer 4) flows in the network (layer 3 or 2) precludes some important end-to-end functions. For instance:

- a. Per-flow forms of L4S like FQ-CoDel are incompatible with full end-to-end encryption of transport layer identifiers for privacy and confidentiality (e.g. IPSec or encrypted VPN tunnels, as opposed to TLS over UDP), because they require packet inspection to access the end-to-end transport flow identifiers.

In contrast, the DualQ form of L4S requires no deeper inspection than the IP layer. So, as long as operators take the DualQ approach, their users can have both very low queuing delay and full end-to-end encryption [RFC8404].

- b. With per-flow forms of L4S, the network takes over control of the relative rates of each application flow. Some see it as an advantage that the network will prevent some flows running faster than others. Others consider it an inherent part of the Internet's appeal that applications can control their rate while taking account of the needs of others via congestion signals. They maintain that this has allowed applications with interesting rate behaviours to evolve, for instance, variable bit-rate video that varies around an equal share rather than being forced to remain equal at every instant, or e2e scavenger behaviours [RFC6817] that use less than an equal share of capacity [LEDBAT\_AQM].

The L4S architecture does not require the IETF to commit to one approach over the other, because it supports both, so that the 'market' can decide. Nonetheless, in the spirit of 'Do one thing and do it well' [McIlroy78], the DualQ option provides low delay without prejudging the issue of flow-rate control. Then, flow rate policing can be added separately if desired. This allows application control up to a point, but the network can still choose to set the point at which it intervenes to prevent one flow completely starving another.

Note:

1. It might seem that self-inflicted queuing delay within a per-flow queue should not be counted, because if the delay wasn't in the network it would just shift to the sender. However, modern adaptive applications, e.g. HTTP/2 [RFC7540] or some interactive media applications (see Section 6.1), can keep low latency objects at the front of their local send queue by shuffling priorities of other objects dependent on the progress of other transfers (for example see [lowat]). They cannot shuffle objects once they have released them into the network.

Alternative Back-off ECN (ABE): Here again, L4S is not an alternative to ABE but a complement that introduces much lower queuing delay. ABE [RFC8511] alters the host behaviour in response to ECN marking to utilize a link better and give ECN flows faster throughput. It uses ECT(0) and assumes the network still treats ECN and drop the same. Therefore ABE exploits any lower queuing delay that AQMs can provide. But as explained above, AQMs still cannot reduce queuing delay too far without losing link utilization (to allow for other, non-ABE, flows).

BBR: Bottleneck Bandwidth and Round-trip propagation time

(BBR [I-D.cardwell-iccr-g-bbr-congestion-control]) controls queuing delay end-to-end without needing any special logic in the network, such as an AQM. So it works pretty-much on any path. BBR keeps queuing delay reasonably low, but perhaps not quite as low as with state-of-the-art AQMs such as PIE or FQ-CoDel, and certainly nowhere near as low as with L4S. Queuing delay is also not consistently low, due to BBR's regular bandwidth probing spikes and its aggressive flow start-up phase.

L4S complements BBR. Indeed BBRv2 can use L4S ECN where available and a scalable L4S congestion control behaviour in response to any ECN signalling from the path [BBRv2]. The L4S ECN signal complements the delay based congestion control aspects of BBR with an explicit indication that hosts can use, both to converge on a fair rate and to keep below a shallow queue target set by the network. Without L4S ECN, both these aspects need to be assumed or estimated.

## 6. Applicability

### 6.1. Applications

A transport layer that solves the current latency issues will provide new service, product and application opportunities.

With the L4S approach, the following existing applications also experience significantly better quality of experience under load:

- \* Gaming, including cloud based gaming;
- \* VoIP;
- \* Video conferencing;
- \* Web browsing;
- \* (Adaptive) video streaming;
- \* Instant messaging.

The significantly lower queuing latency also enables some interactive application functions to be offloaded to the cloud that would hardly even be usable today:

- \* Cloud based interactive video;
- \* Cloud based virtual and augmented reality.

The above two applications have been successfully demonstrated with L4S, both running together over a 40 Mb/s broadband access link loaded up with the numerous other latency sensitive applications in the previous list as well as numerous downloads - all sharing the same bottleneck queue simultaneously [L4Sdemo16]. For the former, a panoramic video of a football stadium could be swiped and pinched so that, on the fly, a proxy in the cloud could generate a sub-window of the match video under the finger-gesture control of each user. For the latter, a virtual reality headset displayed a viewport taken from a 360 degree camera in a racing car. The user's head movements controlled the viewport extracted by a cloud-based proxy. In both cases, with 7 ms end-to-end base delay, the additional queuing delay of roughly 1 ms was so low that it seemed the video was generated locally.

Using a swiping finger gesture or head movement to pan a video are extremely latency-demanding actions -- far more demanding than VoIP. Because human vision can detect extremely low delays of the order of single milliseconds when delay is translated into a visual lag between a video and a reference point (the finger or the orientation of the head sensed by the balance system in the inner ear -- the vestibular system).

Without the low queuing delay of L4S, cloud-based applications like these would not be credible without significantly more access bandwidth (to deliver all possible video that might be viewed) and more local processing, which would increase the weight and power consumption of head-mounted displays. When all interactive processing can be done in the cloud, only the data to be rendered for the end user needs to be sent.

Other low latency high bandwidth applications such as:

- \* Interactive remote presence;
- \* Video-assisted remote control of machinery or industrial processes.

are not credible at all without very low queuing delay. No amount of extra access bandwidth or local processing can make up for lost time.

## 6.2. Use Cases

The following use-cases for L4S are being considered by various interested parties:

- \* Where the bottleneck is one of various types of access network:  
e.g. DSL, Passive Optical Networks (PON), DOCSIS cable, mobile, satellite (see Section 6.3 for some technology-specific details)
- \* Private networks of heterogeneous data centres, where there is no single administrator that can arrange for all the simultaneous changes to senders, receivers and network needed to deploy DCTCP:
  - a set of private data centres interconnected over a wide area with separate administrations, but within the same company
  - a set of data centres operated by separate companies interconnected by a community of interest network (e.g. for the finance sector)
  - multi-tenant (cloud) data centres where tenants choose their operating system stack (Infrastructure as a Service - IaaS)
- \* Different types of transport (or application) congestion control:
  - elastic (TCP/SCTP);
  - real-time (RTP, RMCAT);
  - query (DNS/LDAP).
- \* Where low delay quality of service is required, but without inspecting or intervening above the IP layer [RFC8404]:
  - mobile and other networks have tended to inspect higher layers in order to guess application QoS requirements. However, with growing demand for support of privacy and encryption, L4S offers an alternative. There is no need to select which traffic to favour for queuing, when L4S can give favourable queuing to all traffic.
- \* If queuing delay is minimized, applications with a fixed delay budget can communicate over longer distances, or via a longer chain of service functions [RFC7665] or onion routers.
- \* If delay jitter is minimized, it is possible to reduce the dejitter buffers on the receive end of video streaming, which should improve the interactive experience

### 6.3. Applicability with Specific Link Technologies

Certain link technologies aggregate data from multiple packets into bursts, and buffer incoming packets while building each burst. WiFi, PON and cable all involve such packet aggregation, whereas fixed Ethernet and DSL do not. No sender, whether L4S or not, can do anything to reduce the buffering needed for packet aggregation. So an AQM should not count this buffering as part of the queue that it controls, given no amount of congestion signals will reduce it.

Certain link technologies also add buffering for other reasons, specifically:

- \* Radio links (cellular, WiFi, satellite) that are distant from the source are particularly challenging. The radio link capacity can vary rapidly by orders of magnitude, so it is considered desirable to hold a standing queue that can utilize sudden increases of capacity;
- \* Cellular networks are further complicated by a perceived need to buffer in order to make hand-overs imperceptible;

L4S cannot remove the need for all these different forms of buffering. However, by removing 'the longest pole in the tent' (buffering for the large sawteeth of Classic congestion controls), L4S exposes all these 'shorter poles' to greater scrutiny.

Until now, the buffering needed for these additional reasons tended to be over-specified - with the excuse that none were 'the longest pole in the tent'. But having removed the 'longest pole', it becomes worthwhile to minimize them, for instance reducing packet aggregation burst sizes and MAC scheduling intervals.

### 6.4. Deployment Considerations

L4S AQMs, whether DualQ [I-D.ietf-tsvwg-aqm-dualq-coupled] or FQ, e.g. [RFC8290] are, in themselves, an incremental deployment mechanism for L4S - so that L4S traffic can coexist with existing Classic (Reno-friendly) traffic. Section 6.4.1 explains why only deploying an L4S AQM in one node at each end of the access link will realize nearly all the benefit of L4S.

L4S involves both end systems and the network, so Section 6.4.2 suggests some typical sequences to deploy each part, and why there will be an immediate and significant benefit after deploying just one part.

Section 6.4.3 and Section 6.4.4 describe the converse incremental deployment case where there is no L4S AQM at the network bottleneck, so any L4S flow traversing this bottleneck has to take care in case it is competing with Classic traffic.

#### 6.4.1. Deployment Topology

L4S AQMs will not have to be deployed throughout the Internet before L4S can benefit anyone. Operators of public Internet access networks typically design their networks so that the bottleneck will nearly always occur at one known (logical) link. This confines the cost of queue management technology to one place.

The case of mesh networks is different and will be discussed later in this section. But the known bottleneck case is generally true for Internet access to all sorts of different 'sites', where the word 'site' includes home networks, small- to medium-sized campus or enterprise networks and even cellular devices (Figure 2). Also, this known-bottleneck case tends to be applicable whatever the access link technology; whether xDSL, cable, PON, cellular, line of sight wireless or satellite.

Therefore, the full benefit of the L4S service should be available in the downstream direction when an L4S AQM is deployed at the ingress to this bottleneck link. And similarly, the full upstream service will be available once an L4S AQM is deployed at the ingress into the upstream link. (Of course, multi-homed sites would only see the full benefit once all their access links were covered.)

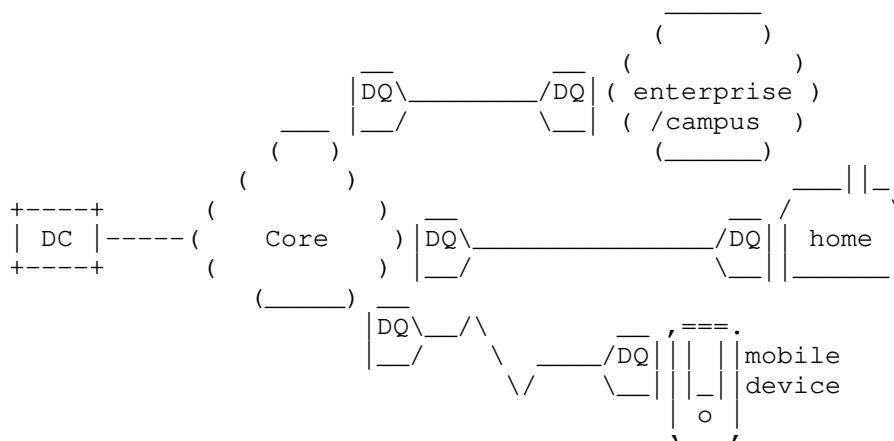


Figure 2: Likely location of DualQ (DQ) Deployments in common access topologies



Deployment in mesh topologies depends on how overbooked the core is. If the core is non-blocking, or at least generously provisioned so that the edges are nearly always the bottlenecks, it would only be necessary to deploy an L4S AQM at the edge bottlenecks. For example, some data-centre networks are designed with the bottleneck in the hypervisor or host NICs, while others bottleneck at the top-of-rack switch (both the output ports facing hosts and those facing the core).

An L4S AQM would often next be needed where the WiFi links in a home sometimes become the bottleneck. And an L4S AQM would eventually also need to be deployed at any other persistent bottlenecks such as network interconnections, e.g. some public Internet exchange points and the ingress and egress to WAN links interconnecting data-centres.

#### 6.4.2. Deployment Sequences

For any one L4S flow to provide benefit, it requires three (or sometimes two) parts to have been deployed: i) the congestion control at the sender; ii) the AQM at the bottleneck; and iii) older transports (namely TCP) need upgraded receiver feedback too. This was the same deployment problem that ECN faced [RFC8170] so we have learned from that experience.

Firstly, L4S deployment exploits the fact that DCTCP already exists on many Internet hosts (Windows, FreeBSD and Linux); both servers and clients. Therefore, an L4S AQM can be deployed at a network bottleneck to immediately give a working deployment of all the L4S parts for testing, as long as the ECT(0) codepoint is switched to ECT(1). DCTCP needs some safety concerns to be fixed for general use over the public Internet (see Section 4.3 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id]), but DCTCP is not on by default, so these issues can be managed within controlled deployments or controlled trials.

Secondly, the performance improvement with L4S is so significant that it enables new interactive services and products that were not previously possible. It is much easier for companies to initiate new work on deployment if there is budget for a new product trial. If, in contrast, there were only an incremental performance improvement (as with Classic ECN), spending on deployment tends to be much harder to justify.

Thirdly, the L4S identifier is defined so that initially network operators can enable L4S exclusively for certain customers or certain applications. But this is carefully defined so that it does not compromise future evolution towards L4S as an Internet-wide service. This is because the L4S identifier is defined not only as the end-to-

end ECN field, but it can also optionally be combined with any other packet header or some status of a customer or their access link (see section 5.4 of [I-D.ietf-tsvwg-ecn-l4s-id]). Operators could do this anyway, even if it were not blessed by the IETF. However, it is best for the IETF to specify that, if they use their own local identifier, it must be in combination with the IETF's identifier. Then, if an operator has opted for an exclusive local-use approach, later they only have to remove this extra rule to make the service work Internet-wide - it will already traverse middleboxes, peerings, etc.

	Servers or proxies	Access link	Clients
0	DCTCP (existing)		DCTCP (existing)
1	Add L4S AQM downstream WORKS DOWNSTREAM FOR CONTROLLED DEPLOYMENTS/TRIALS		
2	Upgrade DCTCP to TCP Prague	FULLY WORKS DOWNSTREAM	Replace DCTCP feedb'k with AccECN
3	Add L4S AQM upstream FULLY WORKS UPSTREAM AND DOWNSTREAM		Upgrade DCTCP to TCP Prague

Figure 3: Example L4S Deployment Sequence

Figure 3 illustrates some example sequences in which the parts of L4S might be deployed. It consists of the following stages:

1. Here, the immediate benefit of a single AQM deployment can be seen, but limited to a controlled trial or controlled deployment. In this example downstream deployment is first, but in other scenarios the upstream might be deployed first. If no AQM at all was previously deployed for the downstream access, an L4S AQM greatly improves the Classic service (as well as adding the L4S service). If an AQM was already deployed, the Classic service will be unchanged (and L4S will add an improvement on top).
2. In this stage, the name 'TCP Prague' [I-D.briscoe-iccrp-prague-congestion-control] is used to represent a variant of DCTCP that is designed to be used in a production Internet environment (assuming it complies with the requirements in Section 4 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id]). If the application is

primarily unidirectional, 'TCP Prague' at one end will provide all the benefit needed. For TCP transports, Accurate ECN feedback (AccECN) [I-D.ietf-tcpm-accurate-ecn] is needed at the other end, but it is a generic ECN feedback facility that is already planned to be deployed for other purposes, e.g. DCTCP, BBR. The two ends can be deployed in either order, because, in TCP, an L4S congestion control only enables itself if it has negotiated the use of AccECN feedback with the other end during the connection handshake. Thus, deployment of TCP Prague on a server enables L4S trials to move to a production service in one direction, wherever AccECN is deployed at the other end. This stage might be further motivated by the performance improvements of TCP Prague relative to DCTCP (see Appendix A.2 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id]).

Unlike TCP, from the outset, QUIC ECN feedback [RFC9000] has supported L4S. Therefore, if the transport is QUIC, one-ended deployment of a Prague congestion control at this stage is simple and sufficient.

3. This is a two-move stage to enable L4S upstream. An L4S AQM or TCP Prague can be deployed in either order as already explained. To motivate the first of two independent moves, the deferred benefit of enabling new services after the second move has to be worth it to cover the first mover's investment risk. As explained already, the potential for new interactive services provides this motivation. An L4S AQM also improves the upstream Classic service - significantly if no other AQM has already been deployed.

Note that other deployment sequences might occur. For instance: the upstream might be deployed first; a non-TCP protocol might be used end-to-end, e.g. QUIC, RTP; a body such as the 3GPP might require L4S to be implemented in 5G user equipment, or other random acts of kindness.

#### 6.4.3. L4S Flow but Non-ECN Bottleneck

If L4S is enabled between two hosts, the L4S sender is required to coexist safely with Reno in response to any drop (see Section 4.3 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id]).

Unfortunately, as well as protecting Classic traffic, this rule degrades the L4S service whenever there is any loss, even if the cause is not persistent congestion at a bottleneck, e.g.:

- \* congestion loss at other transient bottlenecks, e.g. due to bursts in shallower queues;

- \* transmission errors, e.g. due to electrical interference;
- \* rate policing.

Three complementary approaches are in progress to address this issue, but they are all currently research:

- \* In Prague congestion control, ignore certain losses deemed unlikely to be due to congestion (using some ideas from BBR [I-D.cardwell-iccr-g-bbr-congestion-control] regarding isolated losses). This could mask any of the above types of loss while still coexisting with drop-based congestion controls.
- \* A combination of RACK, L4S and link retransmission without resequencing could repair transmission errors without the head of line blocking delay usually associated with link-layer retransmission [UnorderedLTE], [I-D.ietf-tsvwg-ecn-l4s-id];
- \* Hybrid ECN/drop rate policers (see Section 8.3).

L4S deployment scenarios that minimize these issues (e.g. over wireline networks) can proceed in parallel to this research, in the expectation that research success could continually widen L4S applicability.

#### 6.4.4. L4S Flow but Classic ECN Bottleneck

Classic ECN support is starting to materialize on the Internet as an increased level of CE marking. It is hard to detect whether this is all due to the addition of support for ECN in implementations of FQ-CoDel and/or FQ-COBALT, which is not generally problematic, because flow-queue (FQ) scheduling inherently prevents a flow from exceeding the 'fair' rate irrespective of its aggressiveness. However, some of this Classic ECN marking might be due to single-queue ECN deployment. This case is discussed in Section 4.3 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id].

#### 6.4.5. L4S AQM Deployment within Tunnels

An L4S AQM uses the ECN field to signal congestion. So, in common with Classic ECN, if the AQM is within a tunnel or at a lower layer, correct functioning of ECN signalling requires correct propagation of the ECN field up the layers [RFC6040], [I-D.ietf-tsvwg-rfc6040update-shim], [I-D.ietf-tsvwg-ecn-encap-guidelines].

## 7. IANA Considerations (to be removed by RFC Editor)

This specification contains no IANA considerations.

## 8. Security Considerations

### 8.1. Traffic Rate (Non-)Policing

In the current Internet, scheduling usually enforces separation between 'sites' (e.g. households, businesses or mobile users [RFC0970]) and various techniques like redirection to traffic scrubbing facilities deal with flooding attacks. However, there has never been a universal need to police the rate of individual application flows - the Internet has generally always relied on self-restraint of congestion controls at senders for sharing intra-'site' capacity.

As explained in Section 5.2, the DualQ variant of L4S provides low delay without prejudging the issue of flow-rate control. Then, if flow-rate control is needed, per-flow-queuing (FQ) can be used instead, or flow rate policing can be added as a modular addition to a DualQ.

Because the L4S service reduces delay without increasing the delay of Classic traffic, it should not be necessary to rate-police access to the L4S service. In contrast, Section 5.2 explains how Diffserv only makes a difference if some packets get less favourable treatment than others, which typically requires traffic rate policing, which can, in turn, lead to further complexity such as traffic contracts at trust boundaries. Because L4S avoids this management complexity, it is more likely to work end-to-end.

During early deployment (and perhaps always), some networks will not offer the L4S service. In general, these networks should not need to police L4S traffic. They are required (by both the ECN spec [RFC3168] and the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id]) not to change the L4S identifier, which would interfere with end-to-end congestion control. Instead they can merely treat L4S traffic as Not-ECT, as they might already treat all ECN traffic today. At a bottleneck, such networks will introduce some queuing and dropping. When a scalable congestion control detects a drop it will have to respond safely with respect to Classic congestion controls (as required in Section 4.3 of [I-D.ietf-tsvwg-ecn-l4s-id]). This will degrade the L4S service to be no better (but never worse) than Classic best efforts, whenever a non-ECN bottleneck is encountered on a path (see Section 6.4.3).

In cases that are expected to be rare, networks that solely support Classic ECN [RFC3168] in a single queue bottleneck might opt to police L4S traffic so as to protect competing Classic ECN traffic (for instance, see Section 6.1.3 of the L4S operational guidance [I-D.ietf-tsvwg-l4sops]). However, Section 4.3 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id] recommends that the sender adapts its congestion response to properly coexist with Classic ECN flows, i.e. reverting to the self-restraint approach.

Certain network operators might choose to restrict access to the L4S class, perhaps only to selected premium customers as a value-added service. Their packet classifier (item 2 in Figure 1) could identify such customers against some other field (e.g. source address range) as well as classifying on the ECN field. If only the ECN L4S identifier matched, but not the source address (say), the classifier could direct these packets (from non-premium customers) into the Classic queue. Explaining clearly how operators can use an additional local classifiers (see section 5.4 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id]) is intended to remove any motivation to clear the L4S identifier. Then at least the L4S ECN identifier will be more likely to survive end-to-end even though the service may not be supported at every hop. Such local arrangements would only require simple registered/not-registered packet classification, rather than the managed, application-specific traffic policing against customer-specific traffic contracts that Diffserv uses.

## 8.2. 'Latency Friendliness'

Like the Classic service, the L4S service relies on self-restraint - limiting rate in response to congestion. In addition, the L4S service requires self-restraint in terms of limiting latency (burstiness). It is hoped that self-interest and guidance on dynamic behaviour (especially flow start-up, which might need to be standardized) will be sufficient to prevent transports from sending excessive bursts of L4S traffic, given the application's own latency will suffer most from such behaviour.

Whether burst policing becomes necessary remains to be seen. Without it, there will be potential for attacks on the low latency of the L4S service.

If needed, various arrangements could be used to address this concern:

Local bottleneck queue protection: A per-flow (5-tuple) queue

protection function [I-D.briscoe-docsis-q-protection] has been developed for the low latency queue in DOCSIS, which has adopted the DualQ L4S architecture. It protects the low latency service from any queue-building flows that accidentally or maliciously classify themselves into the low latency queue. It is designed to score flows based solely on their contribution to queuing (not flow rate in itself). Then, if the shared low latency queue is at risk of exceeding a threshold, the function redirects enough packets of the highest scoring flow(s) into the Classic queue to preserve low latency.

**Distributed traffic scrubbing:** Rather than policing locally at each bottleneck, it may only be necessary to address problems reactively, e.g. punitively target any deployments of new bursty malware, in a similar way to how traffic from flooding attack sources is rerouted via scrubbing facilities.

**Local bottleneck per-flow scheduling:** Per-flow scheduling should inherently isolate non-bursty flows from bursty (see Section 5.2 for discussion of the merits of per-flow scheduling relative to per-flow policing).

**Distributed access subnet queue protection:** Per-flow queue protection could be arranged for a queue structure distributed across a subnet inter-communicating using lower layer control messages (see Section 2.1.4 of [QDyn]). For instance, in a radio access network, user equipment already sends regular buffer status reports to a radio network controller, which could use this information to remotely police individual flows.

**Distributed Congestion Exposure to Ingress Policers:** The Congestion Exposure (ConEx) architecture [RFC7713] which uses egress audit to motivate senders to truthfully signal path congestion in-band where it can be used by ingress policers. An edge-to-edge variant of this architecture is also possible.

**Distributed Domain-edge traffic conditioning:** An architecture similar to Diffserv [RFC2475] may be preferred, where traffic is proactively conditioned on entry to a domain, rather than reactively policed only if it leads to queuing once combined with other traffic at a bottleneck.

**Distributed core network queue protection:** The policing function

could be divided between per-flow mechanisms at the network ingress that characterize the burstiness of each flow into a signal carried with the traffic, and per-class mechanisms at bottlenecks that act on these signals if queuing actually occurs once the traffic converges. This would be somewhat similar to [Nadas20], which is in turn similar to the idea behind core stateless fair queuing.

None of these possible queue protection capabilities are considered a necessary part of the L4S architecture, which works without them (in a similar way to how the Internet works without per-flow rate policing). Indeed, even where latency policers are deployed, under normal circumstances they would not intervene, and if operators found they were not necessary they could disable them. Part of the L4S experiment will be to see whether such a function is necessary, and which arrangements are most appropriate to the size of the problem.

### 8.3. Interaction between Rate Policing and L4S

As mentioned in Section 5.2, L4S should remove the need for low latency Diffserv classes. However, those Diffserv classes that give certain applications or users priority over capacity, would still be applicable in certain scenarios (e.g. corporate networks). Then, within such Diffserv classes, L4S would often be applicable to give traffic low latency and low loss as well. Within such a Diffserv class, the bandwidth available to a user or application is often limited by a rate policer. Similarly, in the default Diffserv class, rate policers are used to partition shared capacity.

A classic rate policer drops any packets exceeding a set rate, usually also giving a burst allowance (variants exist where the policer re-marks non-compliant traffic to a discard-eligible Diffserv codepoint, so they can be dropped elsewhere during contention). Whenever L4S traffic encounters one of these rate policers, it will experience drops and the source will have to fall back to a Classic congestion control, thus losing the benefits of L4S (Section 6.4.3). So, in networks that already use rate policers and plan to deploy L4S, it will be preferable to redesign these rate policers to be more friendly to the L4S service.

L4S-friendly rate policing is currently a research area (note that this is not the same as latency policing). It might be achieved by setting a threshold where ECN marking is introduced, such that it is just under the policed rate or just under the burst allowance where drop is introduced. For instance the two-rate three-colour marker [RFC2698] or a PCN threshold and excess-rate marker [RFC5670] could mark ECN at the lower rate and drop at the higher. Or an existing rate policer could have congestion-rate policing added,



e.g. using the 'local' (non-ConEx) variant of the ConEx aggregate congestion policer [I-D.briscoe-conex-policing]. It might also be possible to design scalable congestion controls to respond less catastrophically to loss that has not been preceded by a period of increasing delay.

The design of L4S-friendly rate policers will require a separate dedicated document. For further discussion of the interaction between L4S and Diffserv, see [I-D.briscoe-tsvwg-l4s-diffserv].

#### 8.4. ECN Integrity

Receiving hosts can fool a sender into downloading faster by suppressing feedback of ECN marks (or of losses if retransmissions are not necessary or available otherwise). Various ways to protect transport feedback integrity have been developed. For instance:

- \* The sender can test the integrity of the receiver's feedback by occasionally setting the IP-ECN field to the congestion experienced (CE) codepoint, which is normally only set by a congested link. Then the sender can test whether the receiver's feedback faithfully reports what it expects (see 2nd para of Section 20.2 of the Classic ECN spec [RFC3168]).
- \* A network can enforce a congestion response to its ECN markings (or packet losses) by auditing congestion exposure (ConEx) [RFC7713].
- \* Transport layer authentication such as the TCP authentication option (TCP-AO [RFC5925]) or QUIC's use of TLS [RFC9001] can detect any tampering with congestion feedback.
- \* The ECN Nonce [RFC3540] was proposed to detect tampering with congestion feedback, but it has been reclassified as historic [RFC8311].

Appendix C.1 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id] gives more details of these techniques including their applicability and pros and cons.

#### 8.5. Privacy Considerations

As discussed in Section 5.2, the L4S architecture does not preclude approaches that inspect end-to-end transport layer identifiers. For instance, L4S support has been added to FQ-CoDel, which classifies by application flow ID in the network. However, the main innovation of L4S is the DualQ AQM framework that does not need to inspect any deeper than the outermost IP header, because the L4S identifier is in

the IP-ECN field.

Thus, the L4S architecture enables very low queuing delay without requiring inspection of information above the IP layer. This means that users who want to encrypt application flow identifiers, e.g. in IPSec or other encrypted VPN tunnels, don't have to sacrifice low delay [RFC8404].

Because L4S can provide low delay for a broad set of applications that choose to use it, there is no need for individual applications or classes within that broad set to be distinguishable in any way while traversing networks. This removes much of the ability to correlate between the delay requirements of traffic and other identifying features [RFC6973]. There may be some types of traffic that prefer not to use L4S, but the coarse binary categorization of traffic reveals very little that could be exploited to compromise privacy.

## 9. Acknowledgements

Thanks to Richard Scheffenegger, Wes Eddy, Karen Nielsen, David Black, Jake Holland, Vidhi Goel, Ermin Sakic, Praveen Balasubramanian, Gorrry Fairhurst, Mirja Kuehlewind, Philip Eardley, Neal Cardwell and Pete Heist for their useful review comments.

Bob Briscoe and Koen De Schepper were part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The contribution of Koen De Schepper was also part-funded by the 5Growth and DAEMON EU H2020 projects. Bob Briscoe was also part-funded by the Research Council of Norway through the TimeIn project, partly by CableLabs and partly by the Comcast Innovation Fund. The views expressed here are solely those of the authors.

## 10. Informative References

- [AFCD]     Xue, L., Kumar, S., Cui, C., Kondikoppa, P., Chiu, C-H., and S-J. Park, "Towards fair and low latency next generation high speed networks: AFCD queuing", Journal of Network and Computer Applications 70:183--193, July 2016, <<https://doi.org/10.1016/j.jnca.2016.03.021>>.
- [BBRv2]    Cardwell, N., "TCP BBR v2 Alpha/Preview Release", github repository; Linux congestion control module, <<https://github.com/google/bbr/blob/v2alpha/README.md>>.

- [BDPdata] Briscoe, B., "PI2 Parameters", Technical Report TR-BB-2021-001 arXiv:2107.01003 [cs.NI], July 2021, <<https://arxiv.org/abs/2107.01003>>.
- [BufferSize] Appenzeller, G., Keslassy, I., and N. McKeown, "Sizing Router Buffers", In Proc. SIGCOMM'04 34(4):281--292, September 2004, <<https://doi.org/10.1145/1015467.1015499>>.
- [COBALT] Palmei, J., Gupta, S., Imputato, P., Morton, J., Tahiliani, M. P., Avallone, S., and D. Täht, "Design and Evaluation of COBALT Queue Discipline", In Proc. IEEE Int'l Symp. Local and Metropolitan Area Networks (LANMAN'19) 2019:1-6, July 2019, <<https://ieeexplore.ieee.org/abstract/document/8847054>>.
- [DCttH19] De Schepper, K., Bondarenko, O., Tilmans, O., and B. Briscoe, "'Data Centre to the Home': Ultra-Low Latency for All", Updated RITE project Technical Report , July 2019, <[https://bobbbriscoe.net/pubs.html#DCttH\\_TR](https://bobbbriscoe.net/pubs.html#DCttH_TR)>.
- [DOCSIS3.1] CableLabs, "MAC and Upper Layer Protocols Interface (MULPI) Specification, CM-SP-MULPIv3.1", Data-Over-Cable Service Interface Specifications DOCSIS® 3.1 Version i17 or later, 21 January 2019, <<https://specification-search.cablelabs.com/CM-SP-MULPIv3.1>>.
- [DOCSIS3AQM] White, G., "Active Queue Management Algorithms for DOCSIS 3.0; A Simulation Study of CoDel, SFQ-CoDel and PIE in DOCSIS 3.0 Networks", CableLabs Technical Report , April 2013, <[http://www.cablelabs.com/wp-content/uploads/2013/11/Active\\_Queue\\_Management\\_Algorithms\\_DOCSIS\\_3\\_0.pdf](http://www.cablelabs.com/wp-content/uploads/2013/11/Active_Queue_Management_Algorithms_DOCSIS_3_0.pdf)>.
- [DualPI2Linux] Albisser, O., De Schepper, K., Briscoe, B., Tilmans, O., and H. Steen, "DUALPI2 - Low Latency, Low Loss and Scalable (L4S) AQM", Proc. Linux Netdev 0x13 , March 2019, <<https://www.netdevconf.org/0x13/session.html?talk-DUALPI2-AQM>>.
- [Dukkipati06] Dukkipati, N. and N. McKeown, "Why Flow-Completion Time is the Right Metric for Congestion Control", ACM CCR 36(1):59--62, January 2006, <<https://dl.acm.org/doi/10.1145/1111322.1111336>>.

[FQ\_CoDel\_Thresh]

Høiland-Jørgensen, T., "fq\_codel: generalise ce\_threshold marking for subset of traffic", Linux Patch Commit ID: dfcb63celde6b10b, 20 October 2021, <<https://git.kernel.org/pub/scm/linux/kernel/git/netdev/net-next.git/commit/?id=dfcb63celde6b10b>>.

[Hohlfeld14]

Hohlfeld, O., Pujol, E., Ciucu, F., Feldmann, A., and P. Barford, "A QoE Perspective on Sizing Network Buffers", Proc. ACM Internet Measurement Conf (IMC'14) hmm, November 2014, <<http://doi.acm.org/10.1145/2663716.2663730>>.

[I-D.briscoe-conex-policing]

Briscoe, B., "Network Performance Isolation using Congestion Policing", Work in Progress, Internet-Draft, draft-briscoe-conex-policing-01, 14 February 2014, <<https://datatracker.ietf.org/doc/html/draft-briscoe-conex-policing-01>>.

[I-D.briscoe-docsis-q-protection]

Briscoe, B. and G. White, "The DOCSIS(r) Queue Protection Algorithm to Preserve Low Latency", Work in Progress, Internet-Draft, draft-briscoe-docsis-q-protection-02, 31 January 2022, <<https://datatracker.ietf.org/doc/html/draft-briscoe-docsis-q-protection-02>>.

[I-D.briscoe-iccrp-prague-congestion-control]

Schepper, K. D., Tilmans, O., and B. Briscoe, "Prague Congestion Control", Work in Progress, Internet-Draft, draft-briscoe-iccrp-prague-congestion-control-00, 9 March 2021, <<https://datatracker.ietf.org/doc/html/draft-briscoe-iccrp-prague-congestion-control-00>>.

[I-D.briscoe-tsvwg-l4s-diffserv]

Briscoe, B., "Interactions between Low Latency, Low Loss, Scalable Throughput (L4S) and Differentiated Services", Work in Progress, Internet-Draft, draft-briscoe-tsvwg-l4s-diffserv-02, 4 November 2018, <<https://datatracker.ietf.org/doc/html/draft-briscoe-tsvwg-l4s-diffserv-02>>.

- [I-D.cardwell-iccr-g-bbr-congestion-control]  
Cardwell, N., Cheng, Y., Yeganeh, S. H., Swett, I., and V. Jacobson, "BBR Congestion Control", Work in Progress, Internet-Draft, draft-cardwell-iccr-g-bbr-congestion-control-01, 7 November 2021, <<https://datatracker.ietf.org/doc/html/draft-cardwell-iccr-g-bbr-congestion-control-01>>.
- [I-D.ietf-tcpm-accurate-ecn]  
Briscoe, B., Kühlewind, M., and R. Scheffenegger, "More Accurate ECN Feedback in TCP", Work in Progress, Internet-Draft, draft-ietf-tcpm-accurate-ecn-16, 3 February 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-accurate-ecn-16>>.
- [I-D.ietf-tsvwg-aqm-dualq-coupled]  
Schepper, K. D., Briscoe, B., and G. White, "DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)", Work in Progress, Internet-Draft, draft-ietf-tsvwg-aqm-dualq-coupled-22, 4 March 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-aqm-dualq-coupled-22>>.
- [I-D.ietf-tsvwg-ecn-encap-guidelines]  
Briscoe, B. and J. Kaippallimalil, "Guidelines for Adding Congestion Notification to Protocols that Encapsulate IP", Work in Progress, Internet-Draft, draft-ietf-tsvwg-ecn-encap-guidelines-16, 25 May 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-ecn-encap-guidelines-16>>.
- [I-D.ietf-tsvwg-ecn-l4s-id]  
Schepper, K. D. and B. Briscoe, "Explicit Congestion Notification (ECN) Protocol for Very Low Queuing Delay (L4S)", Work in Progress, Internet-Draft, draft-ietf-tsvwg-ecn-l4s-id-24, 1 February 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-ecn-l4s-id-24>>.
- [I-D.ietf-tsvwg-l4sops]  
White, G., "Operational Guidance for Deployment of L4S in the Internet", Work in Progress, Internet-Draft, draft-ietf-tsvwg-l4sops-02, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-l4sops-02>>.

[I-D.ietf-tsvwg-nqb]

White, G. and T. Fossati, "A Non-Queue-Building Per-Hop Behavior (NQB PHB) for Differentiated Services", Work in Progress, Internet-Draft, draft-ietf-tsvwg-nqb-10, 4 March 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-nqb-10>>.

[I-D.ietf-tsvwg-rfc6040update-shim]

Briscoe, B., "Propagating Explicit Congestion Notification Across IP Tunnel Headers Separated by a Shim", Work in Progress, Internet-Draft, draft-ietf-tsvwg-rfc6040update-shim-14, 25 May 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-rfc6040update-shim-14>>.

[I-D.morton-tsvwg-codel-approx-fair]

Morton, J. and P. G. Heist, "Controlled Delay Approximate Fairness AQM", Work in Progress, Internet-Draft, draft-morton-tsvwg-codel-approx-fair-01, 9 March 2020, <<https://datatracker.ietf.org/doc/html/draft-morton-tsvwg-codel-approx-fair-01>>.

[I-D.sridharan-tcpm-ctcp]

Sridharan, M., Tan, K., Bansal, D., and D. Thaler, "Compound TCP: A New TCP Congestion Control for High-Speed and Long Distance Networks", Work in Progress, Internet-Draft, draft-sridharan-tcpm-ctcp-02, 11 November 2008, <<https://datatracker.ietf.org/doc/html/draft-sridharan-tcpm-ctcp-02>>.

[I-D.stewart-tsvwg-sctp-ecn]

Stewart, R. R., Tuexen, M., and X. Dong, "ECN for Stream Control Transmission Protocol (SCTP)", Work in Progress, Internet-Draft, draft-stewart-tsvwg-sctp-ecn-05, 15 January 2014, <<https://datatracker.ietf.org/doc/html/draft-stewart-tsvwg-sctp-ecn-05>>.

[L4Sdemo16]

Bondarenko, O., De Schepper, K., Tsang, I., and B. Briscoe, "Ultra-Low Delay for All: Live Experience, Live Analysis", Proc. MMSYS'16 pp33:1--33:4, May 2016, <<http://dl.acm.org/citation.cfm?doid=2910017.2910633>> (videos of demos: <https://riteproject.eu/dctth/#1511dispatchwg> )>.

[LEDBAT\_AQM]

Al-Saadi, R., Armitage, G., and J. But, "Characterising LEDBAT Performance Through Bottlenecks Using PIE, FQ-CoDel

- and FQ-PIE Active Queue Management", Proc. IEEE 42nd Conference on Local Computer Networks (LCN) 278--285, 2017, <<https://ieeexplore.ieee.org/document/8109367>>.
- [lowat] Meenan, P., "Optimizing HTTP/2 prioritization with BBR and tcp\_notsent\_lowat", Cloudflare Blog , 12 October 2018, <<https://blog.cloudflare.com/http-2-prioritization-with-nginx/>>.
- [Mathis09] Mathis, M., "Relentless Congestion Control", PFLDNeT'09 , May 2009, <<https://www.gdt.id.au/~gdt/presentations/2010-07-06-questnet-tcp/reference-materials/papers/mathis-relentless-congestion-control.pdf>>.
- [McIlroy78] McIlroy, M.D., Pinson, E. N., and B. A. Tague, "UNIX Time-Sharing System: Foreword", The Bell System Technical Journal 57:6(1902--1903), July 1978, <<https://archive.org/details/bstj57-6-1899>>.
- [Nadas20] Nádas, S., Gombos, G., Fejes, F., and S. Laki, "A Congestion Control Independent L4S Scheduler", Proc. Applied Networking Research Workshop (ANRW '20) 45--51, July 2020, <<https://doi.org/10.1145/3404868.3406669>>.
- [PragueLinux] Briscoe, B., De Schepper, K., Albisser, O., Misund, J., Tilmans, O., Kühlewind, M., and A.S. Ahmed, "Implementing the 'TCP Prague' Requirements for Low Latency Low Loss Scalable Throughput (L4S)", Proc. Linux Netdev 0x13 , March 2019, <<https://www.netdevconf.org/0x13/session.html?talk-tcp-prague-l4s>>.
- [QDyn] Briscoe, B., "Rapid Signalling of Queue Dynamics", bobbriscoe.net Technical Report TR-BB-2017-001; arXiv:1904.07044 [cs.NI], September 2017, <<https://arxiv.org/abs/1904.07044>>.
- [Rajiullah15] Rajiullah, M., "Towards a Low Latency Internet: Understanding and Solutions", Masters Thesis; Karlstad Uni, Dept of Maths & CS 2015:41, 2015, <<https://www.diva-portal.org/smash/get/diva2:846109/FULLTEXT01.pdf>>.
- [RFC0970] Nagle, J., "On Packet Switches With Infinite Storage", RFC 970, DOI 10.17487/RFC0970, December 1985, <<https://www.rfc-editor.org/info/rfc970>>.

- [RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, DOI 10.17487/RFC2475, December 1998, <<https://www.rfc-editor.org/info/rfc2475>>.
- [RFC2698] Heinanen, J. and R. Guerin, "A Two Rate Three Color Marker", RFC 2698, DOI 10.17487/RFC2698, September 1999, <<https://www.rfc-editor.org/info/rfc2698>>.
- [RFC2884] Hadi Salim, J. and U. Ahmed, "Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks", RFC 2884, DOI 10.17487/RFC2884, July 2000, <<https://www.rfc-editor.org/info/rfc2884>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC3246] Davie, B., Charny, A., Bennet, J.C.R., Benson, K., Le Boudec, J.Y., Courtney, W., Davari, S., Firoiu, V., and D. Stiliadis, "An Expedited Forwarding PHB (Per-Hop Behavior)", RFC 3246, DOI 10.17487/RFC3246, March 2002, <<https://www.rfc-editor.org/info/rfc3246>>.
- [RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, DOI 10.17487/RFC3540, June 2003, <<https://www.rfc-editor.org/info/rfc3540>>.
- [RFC3649] Floyd, S., "HighSpeed TCP for Large Congestion Windows", RFC 3649, DOI 10.17487/RFC3649, December 2003, <<https://www.rfc-editor.org/info/rfc3649>>.
- [RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI 10.17487/RFC4340, March 2006, <<https://www.rfc-editor.org/info/rfc4340>>.
- [RFC4774] Floyd, S., "Specifying Alternate Semantics for the Explicit Congestion Notification (ECN) Field", BCP 124, RFC 4774, DOI 10.17487/RFC4774, November 2006, <<https://www.rfc-editor.org/info/rfc4774>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.



- [RFC5033] Floyd, S. and M. Allman, "Specifying New Congestion Control Algorithms", BCP 133, RFC 5033, DOI 10.17487/RFC5033, August 2007, <<https://www.rfc-editor.org/info/rfc5033>>.
- [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <<https://www.rfc-editor.org/info/rfc5348>>.
- [RFC5670] Eardley, P., Ed., "Metering and Marking Behaviour of PCN-Nodes", RFC 5670, DOI 10.17487/RFC5670, November 2009, <<https://www.rfc-editor.org/info/rfc5670>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/info/rfc5681>>.
- [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <<https://www.rfc-editor.org/info/rfc5925>>.
- [RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <<https://www.rfc-editor.org/info/rfc6040>>.
- [RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", RFC 6679, DOI 10.17487/RFC6679, August 2012, <<https://www.rfc-editor.org/info/rfc6679>>.
- [RFC6817] Shalunov, S., Hazel, G., Iyengar, J., and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)", RFC 6817, DOI 10.17487/RFC6817, December 2012, <<https://www.rfc-editor.org/info/rfc6817>>.
- [RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <<https://www.rfc-editor.org/info/rfc6973>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<https://www.rfc-editor.org/info/rfc7540>>.

- [RFC7560] Kuehlewind, M., Ed., Scheffenegger, R., and B. Briscoe, "Problem Statement and Requirements for Increased Accuracy in Explicit Congestion Notification (ECN) Feedback", RFC 7560, DOI 10.17487/RFC7560, August 2015, <<https://www.rfc-editor.org/info/rfc7560>>.
- [RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <<https://www.rfc-editor.org/info/rfc7567>>.
- [RFC7665] Halpern, J., Ed. and C. Pignataro, Ed., "Service Function Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <<https://www.rfc-editor.org/info/rfc7665>>.
- [RFC7713] Mathis, M. and B. Briscoe, "Congestion Exposure (ConEx) Concepts, Abstract Mechanism, and Requirements", RFC 7713, DOI 10.17487/RFC7713, December 2015, <<https://www.rfc-editor.org/info/rfc7713>>.
- [RFC8033] Pan, R., Natarajan, P., Baker, F., and G. White, "Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem", RFC 8033, DOI 10.17487/RFC8033, February 2017, <<https://www.rfc-editor.org/info/rfc8033>>.
- [RFC8034] White, G. and R. Pan, "Active Queue Management (AQM) Based on Proportional Integral Controller Enhanced PIE) for Data-Over-Cable Service Interface Specifications (DOCSIS) Cable Modems", RFC 8034, DOI 10.17487/RFC8034, February 2017, <<https://www.rfc-editor.org/info/rfc8034>>.
- [RFC8170] Thaler, D., Ed., "Planning for Protocol Adoption and Subsequent Transitions", RFC 8170, DOI 10.17487/RFC8170, May 2017, <<https://www.rfc-editor.org/info/rfc8170>>.
- [RFC8257] Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <<https://www.rfc-editor.org/info/rfc8257>>.
- [RFC8290] Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm", RFC 8290, DOI 10.17487/RFC8290, January 2018, <<https://www.rfc-editor.org/info/rfc8290>>.

- [RFC8298] Johansson, I. and Z. Sarker, "Self-Clocked Rate Adaptation for Multimedia", RFC 8298, DOI 10.17487/RFC8298, December 2017, <<https://www.rfc-editor.org/info/rfc8298>>.
- [RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <<https://www.rfc-editor.org/info/rfc8311>>.
- [RFC8312] Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", RFC 8312, DOI 10.17487/RFC8312, February 2018, <<https://www.rfc-editor.org/info/rfc8312>>.
- [RFC8404] Moriarty, K., Ed. and A. Morton, Ed., "Effects of Pervasive Encryption on Operators", RFC 8404, DOI 10.17487/RFC8404, July 2018, <<https://www.rfc-editor.org/info/rfc8404>>.
- [RFC8511] Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst, "TCP Alternative Backoff with ECN (ABE)", RFC 8511, DOI 10.17487/RFC8511, December 2018, <<https://www.rfc-editor.org/info/rfc8511>>.
- [RFC8888] Sarker, Z., Perkins, C., Singh, V., and M. Ramalho, "RTP Control Protocol (RTCP) Feedback for Congestion Control", RFC 8888, DOI 10.17487/RFC8888, January 2021, <<https://www.rfc-editor.org/info/rfc8888>>.
- [RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <<https://www.rfc-editor.org/info/rfc9000>>.
- [RFC9001] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <<https://www.rfc-editor.org/info/rfc9001>>.
- [SCReAM] Johansson, I., "SCReAM", github repository; , <<https://github.com/ericssonresearch/scream/blob/master/README.md>>.
- [TCP-CA] Jacobson, V. and M.J. Karels, "Congestion Avoidance and Control", Laurence Berkeley Labs Technical Report , November 1988, <<http://ee.lbl.gov/papers/congavoid.pdf>>.

[UnorderedLTE]

Austrheim, M.V., "Implementing immediate forwarding for 4G in a network simulator", Masters Thesis, Uni Oslo , June 2019.

#### Authors' Addresses

Bob Briscoe (editor)  
Independent  
United Kingdom  
Email: [ietf@bobbriscoe.net](mailto:ietf@bobbriscoe.net)  
URI: <http://bobbriscoe.net/>

Koen De Schepper  
Nokia Bell Labs  
Antwerp  
Belgium  
Email: [koen.de\\_schepper@nokia.com](mailto:koen.de_schepper@nokia.com)  
URI: [https://www.bell-labs.com/usr/koen.de\\_schepper](https://www.bell-labs.com/usr/koen.de_schepper)

Marcelo Bagnulo  
Universidad Carlos III de Madrid  
Av. Universidad 30  
Leganes, Madrid 28911  
Spain  
Phone: 34 91 6249500  
Email: [marcelo@it.uc3m.es](mailto:marcelo@it.uc3m.es)  
URI: <http://www.it.uc3m.es>

Greg White  
CableLabs  
United States of America  
Email: [G.White@CableLabs.com](mailto:G.White@CableLabs.com)

Internet Engineering Task Force  
Internet-Draft  
Obsoletes: 3662 (if approved)  
Updates: 4594,8325 (if approved)  
Intended status: Standards Track  
Expires: September 12, 2019

R. Bless  
Karlsruhe Institute of Technology (KIT)  
March 11, 2019

A Lower Effort Per-Hop Behavior (LE PHB) for Differentiated Services  
draft-ietf-tsvwg-le-phb-10

Abstract

This document specifies properties and characteristics of a Lower Effort (LE) per-hop behavior (PHB). The primary objective of this LE PHB is to protect best-effort (BE) traffic (packets forwarded with the default PHB) from LE traffic in congestion situations, i.e., when resources become scarce, best-effort traffic has precedence over LE traffic and may preempt it. Alternatively, packets forwarded by the LE PHB can be associated with a scavenger service class, i.e., they scavenge otherwise unused resources only. There are numerous uses for this PHB, e.g., for background traffic of low precedence, such as bulk data transfers with low priority in time, non time-critical backups, larger software updates, web search engines while gathering information from web servers and so on. This document recommends a standard DSCP value for the LE PHB. This specification obsoletes RFC 3662 and updates the DSCP recommended in RFC 4594 and RFC 8325 to use the DSCP assigned in this specification.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 12, 2019.

## Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

## Table of Contents

1. Introduction . . . . .	3
2. Requirements Language . . . . .	3
3. Applicability . . . . .	3
4. PHB Description . . . . .	6
5. Traffic Conditioning Actions . . . . .	7
6. Recommended DS Codepoint . . . . .	7
7. Deployment Considerations . . . . .	7
8. Remarking to other DSCPs/PHBs . . . . .	8
9. Multicast Considerations . . . . .	9
10. The Update to RFC 4594 . . . . .	10
11. The Update to RFC 8325 . . . . .	12
12. The Update to draft-ietf-tsvwg-rtcweb-qos . . . . .	12
13. IANA Considerations . . . . .	14
14. Security Considerations . . . . .	14
15. References . . . . .	15
15.1. Normative References . . . . .	15
15.2. Informative References . . . . .	15
Appendix A. History of the LE PHB . . . . .	17
Appendix B. Acknowledgments . . . . .	18

Appendix C. Change History . . . . .	18
Appendix D. Note to RFC Editor . . . . .	21
Author's Address . . . . .	21

## 1. Introduction

This document defines a Differentiated Services per-hop behavior [RFC2474] called "Lower Effort" (LE), which is intended for traffic of sufficiently low urgency that all other traffic takes precedence over the LE traffic in consumption of network link bandwidth. Low urgency traffic has a low priority for timely forwarding, which does not necessarily imply that it is generally of minor importance. From this viewpoint, it can be considered as a network equivalent to a background priority for processes in an operating system. There may or may not be memory (buffer) resources allocated for this type of traffic.

Some networks carry packets that ought to consume network resources only when no other traffic is demanding them. In this point of view, packets forwarded by the LE PHB scavenge otherwise unused resources only, which led to the name "scavenger service" in early Internet2 deployments (see Appendix A). Other commonly used names for LE PHB type services are "Lower-than-best-effort" or "Less-than-best-effort". In summary, with the mentioned feature above, the LE PHB has two important properties: it should scavenge residual capacity and it must be preemptable by the default PHB (or other elevated PHBs) in case they need more resources. Consequently, the effect of this type of traffic on all other network traffic is strictly limited ("no harm" property). This is distinct from "best-effort" (BE) traffic since the network makes no commitment to deliver LE packets. In contrast, BE traffic receives an implied "good faith" commitment of at least some available network resources. This document proposes a Lower Effort Differentiated Services per-hop behavior (LE PHB) for handling this "optional" traffic in a differentiated services node.

## 2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119][RFC8174] when, and only when, they appear in all capitals, as shown here.

## 3. Applicability

A Lower Effort PHB is applicable for many applications that otherwise use best-effort delivery. More specifically, it is suitable for traffic and services that can tolerate strongly varying throughput

for their data flows, especially periods of very low throughput or even starvation (i.e., long interruptions due to significant or even complete packet loss). Therefore, an application sending an LE marked flow needs to be able to tolerate short or (even very) long interruptions due to the presence of severe congestion conditions during the transmission of the flow. Thus, there ought to be an expectation that packets of the LE PHB could be excessively delayed or dropped when any other traffic is present. It is application-dependent when a lack of progress is considered being a failure (e.g., if a transport connection fails due to timing out, the application may try several times to re-establish the transport connection in order to resume the application session before finally giving up). The LE PHB is suitable for sending traffic of low urgency across a Differentiated Services (DS) domain or DS region.

Just like best-effort traffic, LE traffic SHOULD be congestion controlled (i.e., use a congestion controlled transport or implement an appropriate congestion control method [RFC2914] [RFC8085]). Since LE traffic could be starved completely for a longer period of time, transport protocols or applications (and their related congestion control mechanisms) SHOULD be able to detect and react to such a starvation situation. An appropriate reaction would be to resume the transfer instead of aborting it, i.e., an LE optimized transport ought to use appropriate retry strategies (e.g., exponential back-off with an upper bound) as well as corresponding retry and timeout limits in order to avoid the loss of the connection due to the mentioned starvation periods. While it is desirable to achieve a quick resumption of the transfer as soon as resources become available again, it may be difficult to achieve this in practice. In lack of a transport protocol and congestion control that are adapted to LE, applications can also use existing common transport protocols and implement session resumption by trying to re-establish failed connections. Congestion control is not only useful to let the flows within the LE behavior aggregate adapt to the available bandwidth that may be highly fluctuating, but is also essential if LE traffic is mapped to the default PHB in DS domains that do not support LE. In this case, use of background transport protocols, e.g., similar to LEDBAT [RFC6817], is expedient.

Use of the LE PHB might assist a network operator in moving certain kinds of traffic or users to off-peak times. Furthermore, packets can be designated for the LE PHB when the goal is to protect all other packet traffic from competition with the LE aggregate while not completely banning LE traffic from the network. An LE PHB SHOULD NOT be used for a customer's "normal Internet" traffic and packets SHOULD NOT be "downgraded" to the LE PHB instead of being dropped, particularly when the packets are unauthorized traffic. The LE PHB



is expected to have applicability in networks that have at least some unused capacity at certain periods.

The LE PHB allows networks to protect themselves from selected types of traffic as a complement to giving preferential treatment to other selected traffic aggregates. LE ought not to be used for the general case of downgraded traffic, but could be used by design, e.g., to protect an internal network from untrusted external traffic sources. In this case there is no way for attackers to preempt internal (non LE) traffic by flooding. Another use case in this regard is forwarding of multicast traffic from untrusted sources. Multicast forwarding is currently enabled within domains only for specific sources within a domain, but not for sources from anywhere in the Internet. A major problem is that multicast routing creates traffic sources at (mostly) unpredictable branching points within a domain, potentially leading to congestion and packet loss. In the case of multicast traffic packets from untrusted sources are forwarded as LE traffic, they will not harm traffic from non-LE behavior aggregates. A further related use case is mentioned in [RFC3754]: preliminary forwarding of non-admitted multicast traffic.

There is no intrinsic reason to limit the applicability of the LE PHB to any particular application or type of traffic. It is intended as an additional traffic engineering tool for network administrators. For instance, it can be used to fill protection capacity of transmission links that is otherwise unused. Some network providers keep link utilization below 50% to ensure that all traffic is forwarded without loss after rerouting caused by a link failure (cf. Section 6 of [RFC3439]). LE marked traffic can utilize the normally unused capacity and will be preempted automatically in case of link failure when 100% of the link capacity is required for all other traffic. Ideally, applications mark their packets as LE traffic, since they know the urgency of flows. Since LE traffic may be starved for longer periods of time it is probably less suitable for real-time and interactive applications.

Example uses for the LE PHB:

- o For traffic caused by world-wide web search engines while they gather information from web servers.
- o For software updates or dissemination of new releases of operating systems.
- o For reporting errors or telemetry data from operating systems or applications.

- o For backup traffic or non-time critical synchronization or mirroring traffic.
- o For content distribution transfers between caches.
- o For preloading or prefetching objects from web sites.
- o For network news and other "bulk mail" of the Internet.
- o For "downgraded" traffic from some other PHB when this does not violate the operational objectives of the other PHB.
- o For multicast traffic from untrusted (e.g., non-local) sources.

#### 4. PHB Description

The LE PHB is defined in relation to the default PHB (best-effort). A packet forwarded with the LE PHB SHOULD have lower precedence than packets forwarded with the default PHB, i.e., in the case of congestion, LE marked traffic SHOULD be dropped prior to dropping any default PHB traffic. Ideally, LE packets would be forwarded only when no packet with any other PHB is awaiting transmission. This means that in case of link resource contention LE traffic can be starved completely, which may not be always desired by the network operator's policy. The used scheduler to implement the LE PHB may reflect this policy accordingly.

A straightforward implementation could be a simple priority scheduler serving the default PHB queue with higher priority than the lower-effort PHB queue. Alternative implementations may use scheduling algorithms that assign a very small weight to the LE class. This, however, could sometimes cause better service for LE packets compared to BE packets in cases when the BE share is fully utilized and the LE share not.

If a dedicated LE queue is not available, an active queue management mechanism within a common BE/LE queue could also be used. This could drop all arriving LE packets as soon as certain queue length or sojourn time thresholds are exceeded.

Since congestion control is also useful within the LE traffic class, Explicit Congestion Notification (ECN) [RFC3168] SHOULD be used for LE packets, too. More specifically, an LE implementation SHOULD also apply CE marking for ECT marked packets and transport protocols used for LE SHOULD support and employ ECN. For more information on the benefits of using ECN see [RFC8087].

## 5. Traffic Conditioning Actions

If possible, packets SHOULD be pre-marked in DS-aware end systems by applications due to their specific knowledge about the particular precedence of packets. There is no incentive for DS domains to distrust this initial marking, because letting LE traffic enter a DS domain causes no harm. Thus, any policing such as limiting the rate of LE traffic is not necessary at the DS boundary.

As for most other PHBs an initial classification and marking can be also performed at the first DS boundary node according to the DS domain's own policies (e.g., as protection measure against untrusted sources). However, non-LE traffic (e.g., BE traffic) SHOULD NOT be remarked to LE. Remarking traffic from another PHB results in that traffic being "downgraded". This changes the way the network treats this traffic and it is important not to violate the operational objectives of the original PHB. See also remarks with respect to downgrading in Section 3 and Section 8.

## 6. Recommended DS Codepoint

The RECOMMENDED codepoint for the LE PHB is '000001'.

Earlier specifications [RFC4594] recommended to use CS1 as codepoint (as mentioned in [RFC3662]). This is problematic since it may cause a priority inversion in Diffserv domains that treat CS1 as originally proposed in [RFC2474], resulting in forwarding LE packets with higher precedence than BE packets. Existing implementations SHOULD transition to use the unambiguous LE codepoint '000001' whenever possible.

This particular codepoint was chosen due to measurements on the currently observable DSCP remarking behavior in the Internet [ietf99-secchi]. Since some network domains set the former IP precedence bits to zero, it is possible that some other standardized DSCPs get mapped to the LE PHB DSCP if it were taken from the DSCP standards action pool 1 (xxxxx0).

## 7. Deployment Considerations

In order to enable LE support, DS nodes typically only need

- o A BA classifier (Behavior Aggregate classifier, see [RFC2475]) that classifies packets according to the LE DSCP
- o A dedicated LE queue
- o A suitable scheduling discipline, e.g., simple priority queueing

Alternatively, implementations could use active queue management mechanisms instead of a dedicated LE queue, e.g., dropping all arriving LE packets when certain queue length or sojourn time thresholds are exceeded.

Internet-wide deployment of the LE PHB is eased by the following properties:

- o No harm to other traffic: since the LE PHB has the lowest forwarding priority it does not consume resources from other PHBs. Deployment across different provider domains with LE support causes no trust issues or attack vectors to existing (non LE) traffic. Thus, providers can trust LE markings from end-systems, i.e., there is no need to police or remark incoming LE traffic.
- o No PHB parameters or configuration of traffic profiles: the LE PHB itself possesses no parameters that need to be set or configured. Similarly, since LE traffic requires no admission or policing, it is not necessary to configure traffic profiles.
- o No traffic conditioning mechanisms: the LE PHB requires no traffic meters, droppers, or shapers. See also Section 5 for further discussion.

Operators of DS domains that cannot or do not want to implement the LE PHB (e.g., because there is no separate LE queue available in the corresponding nodes) SHOULD NOT drop packets marked with the LE DSCP. They SHOULD map packets with this DSCP to the default PHB and SHOULD preserve the LE DSCP marking. DS domains operators that do not implement the LE PHB should be aware that they violate the "no harm" property of LE. See also Section 8 for further discussion of forwarding LE traffic with the default PHB instead.

#### 8. Remarketing to other DSCPs/PHBs

"DSCP bleaching", i.e., setting the DSCP to '000000' (default PHB) is NOT RECOMMENDED for this PHB. This may cause effects that are in contrast to the original intent in protecting BE traffic from LE traffic (no harm property). In the case that a DS domain does not support the LE PHB, its nodes SHOULD treat LE marked packets with the default PHB instead (by mapping the LE DSCP to the default PHB), but they SHOULD do so without remarketing to DSCP '000000'. The reason for this is that later traversed DS domains may then have still the possibility to treat such packets according to the LE PHB.

Operators of DS domains that forward LE traffic within the BE aggregate need to be aware of the implications, i.e., induced congestion situations and quality-of-service degradation of the

original BE traffic. In this case, the LE property of not harming other traffic is no longer fulfilled. To limit the impact in such cases, traffic policing of the LE aggregate MAY be used.

In the case that LE marked packets are effectively carried within the default PHB (i.e., forwarded as best-effort traffic) they get a better forwarding treatment than expected. For some applications and services, it is favorable if the transmission is finished earlier than expected. However, in some cases it may be against the original intention of the LE PHB user to strictly send the traffic only if otherwise unused resources are available. In the case that LE traffic is mapped to the default PHB, LE traffic may compete with BE traffic for the same resources and thus adversely affect the original BE aggregate. Applications that want to ensure the lower precedence compared to BE traffic even in such cases SHOULD use additionally a corresponding Lower-than-Best-Effort transport protocol [RFC6297], e.g., LEDBAT [RFC6817].

A DS domain that still uses DSCP CS1 for marking LE traffic (including Low Priority-Data as defined in [RFC4594] or the old definition in [RFC3662]) SHOULD remark traffic to the LE DSCP '000001' at the egress to the next DS domain. This increases the probability that the DSCP is preserved end-to-end, whereas a CS1 marked packet may be remarked by the default DSCP if the next domain is applying Diffserv-Interconnection [RFC8100].

## 9. Multicast Considerations

Basically, the multicast considerations in [RFC3754] apply. However, using the Lower Effort PHB for multicast requires paying special attention to the way how packets get replicated inside routers. Due to multicast packet replication, resource contention may actually occur even before a packet is forwarded to its output port and in the worst case, these forwarding resources are missing for higher prioritized multicast or even unicast packets.

Several forward error correction coding schemes such as fountain codes (e.g., [RFC5053]) allow reliable data delivery even in environments with a potential high amount of packet loss in transmission. When used for example over satellite links or other broadcast media, this means that receivers that lose 80% of packets in transmission simply need 5 times as long to receive the complete data than those receivers experiencing no loss (without any receiver feedback required).

Superficially viewed, it may sound very attractive to use IP multicast with the LE PHB to build this type of opportunistic reliable distribution in IP networks, but it can only be usefully

deployed with routers that do not experience forwarding/replication resource starvation when a large amount of packets (virtually) need to be replicated to links where the LE queue is full.

Thus, packet replication of LE marked packets should consider the situation at the respective output links: it is a waste of internal forwarding resources if a packet is replicated to output links that have no resources left for LE forwarding. In those cases a packet would have been replicated just to be dropped immediately after finding a filled LE queue at the respective output port. Such behavior could be avoided for example by using a conditional internal packet replication: a packet would then only be replicated in case the output link is not fully used. This conditional replication, however, is probably not widely implemented.

While the resource contention problem caused by multicast packet replication is also true for other Diffserv PHBs, LE forwarding is special, because often it is assumed that LE packets only get forwarded in case of available resources at the output ports. The previously mentioned redundancy data traffic could nicely use the varying available residual bandwidth being utilized by LE PHB, but only if the specific requirements stated above for conditional replication in the internal implementation of the network devices are considered.

#### 10. The Update to RFC 4594

[RFC4594] recommended to use CS1 as codepoint in section 4.10, whereas CS1 was defined in [RFC2474] to have a higher precedence than CS0, i.e., the default PHB. Consequently, Diffserv domains implementing CS1 according to [RFC2474] will cause a priority inversion for LE packets that contradicts with the original purpose of LE. Therefore, every occurrence of the CS1 DSCP is replaced by the LE DSCP.

Changes:

- o This update to RFC 4594 removes the following entry from figure 3:

Low-Priority Data	CS1	001000	Any flow that has no BW assurance
----------------------	-----	--------	--------------------------------------

and replaces this by the following entry:

Low-Priority Data	LE	000001	Any flow that has no BW assurance
----------------------	----	--------	--------------------------------------

- o This update to RFC 4594 extends the Notes text below figure 3 that currently states "Notes for Figure 3: Default Forwarding (DF) and Class Selector 0 (CS0) provide equivalent behavior and use the same DS codepoint, '000000'." to state "Notes for Figure 3: Default Forwarding (DF) and Class Selector 0 (CS0) provide equivalent behavior and use the same DS codepoint, '000000'. The prior recommendation to use the CS1 DSCP for Low-Priority Data has been replaced by the current recommendation to use the LE DSCP, '000001'."
- o This update to RFC 4594 removes the following entry from figure 4:

Low-Priority Data	CS1	Not applicable	RFC3662	Rate	Yes
----------------------	-----	----------------	---------	------	-----

and replaces this by the following entry:

Low-Priority Data	LE	Not applicable	RFCXXXX	Rate	Yes
----------------------	----	----------------	---------	------	-----

- o Section 2.3 of [RFC4594] specifies: "In network segments that use IP precedence marking, only one of the two service classes can be supported, High-Throughput Data or Low-Priority Data. We RECOMMEND that the DSCP value(s) of the unsupported service class be changed to 000xx1 on ingress and changed back to original value(s) on egress of the network segment that uses precedence marking. For example, if Low-Priority Data is mapped to Standard service class, then 000001 DSCP marking MAY be used to distinguish it from Standard marked packets on egress." This document removes this recommendation, because by using the herein defined LE DSCP such remarking is not necessary. So even if Low-Priority Data is unsupported (i.e., mapped to the default PHB) the LE DSCP should be kept across the domain as RECOMMENDED in Section 8. That removed text is replaced by: "In network segments that use IP Precedence marking, the Low-Priority Data service class receives the same Diffserv QoS as the Standard service class when the LE DSCP is used for Low-Priority Data traffic. This is acceptable behavior for the Low-Priority Data service class, although it is not the preferred behavior."

- o This document removes the following line of RFC 4594,  
Section 4.10: "The RECOMMENDED DSCP marking is CS1 (Class Selector 1)." and replaces this with the following text: "The RECOMMENDED DSCP marking is LE (Lower Effort), which replaces the prior recommendation for CS1 (Class Selector 1) marking."

#### 11. The Update to RFC 8325

Section 4.2.10 of RFC 8325 [RFC8325] specifies "[RFC3662] and [RFC4594] both recommend Low-Priority Data be marked CS1 DSCP." which is updated to "[RFC3662] recommends that Low-Priority Data be marked CS1 DSCP. [RFC4594] as updated by [RFCXXXX] recommends Low-Priority Data be marked LE DSCP."

This document removes the following paragraph of RFC 8325, Section 4.2.10 because this document makes the anticipated change: "Note: This marking recommendation may change in the future, as [LE-PHB] defines a Lower Effort (LE) PHB for Low-Priority Data traffic and recommends an additional DSCP for this traffic."

Section 4.2.10 of RFC 8325 [RFC8325] specifies "therefore, it is RECOMMENDED to map Low-Priority Data traffic marked CS1 DSCP to UP 1" which is updated to "therefore, it is RECOMMENDED to map Low-Priority Data traffic marked with LE DSCP or legacy CS1 DSCP to UP 1"

This update to RFC 8325 replaces the following entry from figure 1:

Low-Priority Data	CS1	RFC 3662	1	AC_BK (Background)
-------------------	-----	----------	---	--------------------

by the following entries:

Low-Priority Data	LE	RFCXXXX	1	AC_BK (Background)
Low-Priority Data (legacy)	CS1	RFC 3662	1	AC_BK (Background)

#### 12. The Update to draft-ietf-tsvwg-rtcweb-qos

Section 5 of [I-D.ietf-tsvwg-rtcweb-qos] describes the Recommended DSCP Values for WebRTC Applications



This update to [I-D.ietf-tsvwg-rtcweb-qos] replaces all occurrences of CS1 with LE in Table 1:

Flow Type	Very Low	Low	Medium	High
Audio	LE (1)	DF (0)	EF (46)	EF (46)
Interactive Video with or without Audio	LE (1)	DF (0)	AF42, AF43 (36, 38)	AF41, AF42 (34, 36)
Non-Interactive Video with or without Audio	LE (1)	DF (0)	AF32, AF33 (28, 30)	AF31, AF32 (26, 28)
Data	LE (1)	DF (0)	AF11	AF21

and updates the following paragraph:

"The above table assumes that packets marked with CS1 are treated as "less than best effort", such as the LE behavior described in [RFC3662]. However, the treatment of CS1 is implementation dependent. If an implementation treats CS1 as other than "less than best effort", then the actual priority (or, more precisely, the per-hop-behavior) of the packets may be changed from what is intended. It is common for CS1 to be treated the same as DF, so applications and browsers using CS1 cannot assume that CS1 will be treated differently than DF [RFC7657]. However, it is also possible per [RFC2474] for CS1 traffic to be given better treatment than DF, thus caution should be exercised when electing to use CS1. This is one of the cases where marking packets using these recommendations can make things worse."

as follows:

"The above table assumes that packets marked with LE are treated as lower effort (i.e., "less than best effort"), such as the LE behavior described in [RFCXXXX]. However, the treatment of LE is implementation dependent. If an implementation treats LE as other than "less than best effort", then the actual priority (or, more precisely, the per-hop-behavior) of the packets may be changed from what is intended. It is common for LE to be treated the same as DF, so applications and browsers using LE cannot assume that LE will be treated differently than DF [RFC7657]. During development of this document, the CS1 DSCP was recommended for "very low" application

priority traffic; implementations that followed that recommendation SHOULD be updated to use the LE DSCP instead of the CS1 DSCP."

### 13. IANA Considerations

This document assigns the Differentiated Services Field Codepoint (DSCP) '000001' from the Differentiated Services Field Codepoints (DSCP) registry (<https://www.iana.org/assignments/dscp-registry/dscp-registry.xhtml>) (Pool 3, Codepoint Space xxxx01, Standards Action) to the LE PHB. This document suggests to use a DSCP from Pool 3 in order to avoid problems for other PHB marked flows to become accidentally remarked as LE PHB, e.g., due to partial DSCP bleaching. See [RFC8436] for re-classifying Pool 3 for Standards Action.

IANA is requested to update the registry as follows:

- o Name: LE
- o Value (Binary): 000001
- o Value (Decimal): 1
- o Reference: [RFC number of this memo]

### 14. Security Considerations

There are no specific security exposures for this PHB. Since it defines a new class of low forwarding priority, remarking other traffic as LE traffic may lead to quality-of-service degradation of such traffic. Thus, any attacker that is able to modify the DSCP of a packet to LE may carry out a downgrade attack. See the general security considerations in [RFC2474] and [RFC2475].

With respect to privacy, an attacker could use the information from the DSCP to infer that the transferred (probably even encrypted) content is considered of low priority or low urgency by a user, in case the DSCP was set on the user's request. On the one hand, this disclosed information is useful only if correlation with metadata (such as the user's IP address) and/or other flows reveal user identity. On the other hand, it might help an observer (e.g., a state level actor) who is interested in learning about the user's behavior from observed traffic: LE marked background traffic (such as software downloads, operating system updates, or telemetry data) may be less interesting for surveillance than general web traffic. Therefore, the LE marking may help the observer to focus on potentially more interesting traffic (however, the user may exploit this particular assumption and deliberately hide interesting traffic in the LE aggregate). Apart from such considerations, the impact of

disclosed information by the LE DSCP is likely negligible in most cases given the numerous traffic analysis possibilities and general privacy threats (e.g., see [RFC6973]).

## 15. References

### 15.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <<http://www.rfc-editor.org/info/rfc2474>>.
- [RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, DOI 10.17487/RFC2475, December 1998, <<http://www.rfc-editor.org/info/rfc2475>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

### 15.2. Informative References

- [carlberg-lbe-2001] Carlberg, K., Gevros, P., and J. Crowcroft, "Lower than best effort: a design and implementation", SIGCOMM Computer Communications Review Volume 31, Issue 2 supplement, April 2001, <<https://doi.org/10.1145/844193.844208>>.
- [chown-lbe-2003] Chown, T., Ferrari, T., Leinen, S., Sabatino, R., Simar, N., and S. Venaas, "Less than Best Effort: Application Scenarios and Experimental Results", In Proceedings of the Second International Workshop on Quality of Service in Multiservice IP Networks (QoS-IP 2003), Lecture Notes in Computer Science, vol 2601. Springer, Berlin, Heidelberg Pages 131-144, February 2003, <[https://doi.org/10.1007/3-540-36480-3\\_10](https://doi.org/10.1007/3-540-36480-3_10)>.

- [draft-bless-diffserv-lbe-phb-00]  
Bless, R. and K. Wehrle, "A Lower Than Best-Effort Per-Hop Behavior", draft-bless-diffserv-lbe-phb-00 (work in progress), September 1999, <<https://tools.ietf.org/html/draft-bless-diffserv-lbe-phb-00>>.
- [I-D.ietf-tsvwg-rtcweb-qos]  
Jones, P., Dhesikan, S., Jennings, C., and D. Druta, "DSCP Packet Markings for WebRTC QoS", draft-ietf-tsvwg-rtcweb-qos-18 (work in progress), August 2016.
- [ietf99-secchi]  
Secchi, R., Venne, A., and A. Custura, "Measurements concerning the DSCP for a LE PHB", Presentation held at 99th IETF Meeting, TSVWG, Prague, July 2017, <<https://datatracker.ietf.org/meeting/99/materials/slides-99-tsvwg-sessb-31measurements-concerning-the-dscp-for-a-le-phb-00>>.
- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, DOI 10.17487/RFC2914, September 2000, <<https://www.rfc-editor.org/info/rfc2914>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<http://www.rfc-editor.org/info/rfc3168>>.
- [RFC3439] Bush, R. and D. Meyer, "Some Internet Architectural Guidelines and Philosophy", RFC 3439, DOI 10.17487/RFC3439, December 2002, <<https://www.rfc-editor.org/info/rfc3439>>.
- [RFC3662] Bless, R., Nichols, K., and K. Wehrle, "A Lower Effort Per-Domain Behavior (PDB) for Differentiated Services", RFC 3662, DOI 10.17487/RFC3662, December 2003, <<http://www.rfc-editor.org/info/rfc3662>>.
- [RFC3754] Bless, R. and K. Wehrle, "IP Multicast in Differentiated Services (DS) Networks", RFC 3754, DOI 10.17487/RFC3754, April 2004, <<http://www.rfc-editor.org/info/rfc3754>>.
- [RFC4594] Babiarz, J., Chan, K., and F. Baker, "Configuration Guidelines for DiffServ Service Classes", RFC 4594, DOI 10.17487/RFC4594, August 2006, <<http://www.rfc-editor.org/info/rfc4594>>.

- [RFC5053] Luby, M., Shokrollahi, A., Watson, M., and T. Stockhammer, "Raptor Forward Error Correction Scheme for Object Delivery", RFC 5053, DOI 10.17487/RFC5053, October 2007, <<https://www.rfc-editor.org/info/rfc5053>>.
- [RFC6297] Welzl, M. and D. Ros, "A Survey of Lower-than-Best-Effort Transport Protocols", RFC 6297, DOI 10.17487/RFC6297, June 2011, <<http://www.rfc-editor.org/info/rfc6297>>.
- [RFC6817] Shalunov, S., Hazel, G., Iyengar, J., and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)", RFC 6817, DOI 10.17487/RFC6817, December 2012, <<http://www.rfc-editor.org/info/rfc6817>>.
- [RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <<https://www.rfc-editor.org/info/rfc6973>>.
- [RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <<https://www.rfc-editor.org/info/rfc8085>>.
- [RFC8087] Fairhurst, G. and M. Welzl, "The Benefits of Using Explicit Congestion Notification (ECN)", RFC 8087, DOI 10.17487/RFC8087, March 2017, <<https://www.rfc-editor.org/info/rfc8087>>.
- [RFC8100] Geib, R., Ed. and D. Black, "Diffserv-Interconnection Classes and Practice", RFC 8100, DOI 10.17487/RFC8100, March 2017, <<http://www.rfc-editor.org/info/rfc8100>>.
- [RFC8325] Szigeti, T., Henry, J., and F. Baker, "Mapping Diffserv to IEEE 802.11", RFC 8325, DOI 10.17487/RFC8325, February 2018, <<https://www.rfc-editor.org/info/rfc8325>>.
- [RFC8436] Fairhurst, G., "Update to IANA Registration Procedures for Pool 3 Values in the Differentiated Services Field Codepoints (DSCP) Registry", RFC 8436, DOI 10.17487/RFC8436, August 2018, <<https://www.rfc-editor.org/info/rfc8436>>.

#### Appendix A. History of the LE PHB

A first version of this PHB was suggested by Roland Bless and Klaus Wehrle in September 1999 [draft-bless-diffserv-lbe-phb-00], named "A Lower Than Best-Effort Per-Hop Behavior". After some discussion in

the Diffserv Working Group Brian Carpenter and Kathie Nichols proposed a "bulk handling" per-domain behavior and believed a PHB was not necessary. Eventually, "Lower Effort" was specified as per-domain behavior and finally became [RFC3662]. More detailed information about its history can be found in Section 10 of [RFC3662].

There are several other names in use for this type of PHB or associated service classes. Well-known is the QBone Scavenger Service (QBSS) that was proposed in March 2001 within the Internet2 QoS Working Group. Alternative names are "Lower-than-best-effort" [carlberg-lbe-2001] or "Less-than-best-effort" [chown-lbe-2003].

#### Appendix B. Acknowledgments

Since text is partially borrowed from earlier Internet-Drafts and RFCs the co-authors of previous specifications are acknowledged here: Kathie Nichols and Klaus Wehrle. David Black, Olivier Bonaventure, Spencer Dawkins, Toerless Eckert, Gorrry Fairhurst, Ruediger Geib, and Kyle Rose provided helpful comments and (partially also text) suggestions.

#### Appendix C. Change History

This section briefly lists changes between Internet-Draft versions for convenience.

Changes in Version 10: (incorporated comments from IESG discussion as follows)

- o Appended "for Differentiated Services" to the title as suggested by Alexey.
- o Addressed Deborah Brungard's discuss: changed phrase to "However, non-LE traffic (e.g., BE traffic) SHOULD NOT be remarked to LE." with additional explanation as suggested by Gorrry.
- o Fixed the sentence "An LE PHB SHOULD NOT be used for a customer's "normal Internet" traffic nor should packets be "downgraded" to the LE PHB instead of being dropped, particularly when the packets are unauthorized traffic." according to Alice's and Mirja's comments.
- o Made reference to RFC8174 normative.
- o Added hint for the RFC editor to apply changes from section Section 12 and to delete it afterwards.

- o Incorporated Mirja's and Benjamin's suggestions.
- o Editorial suggested by Gorrry: In case => In the case that

Changes in Version 09:

- o Incorporated comments from IETF Last Call:
  - \* from Olivier Bonaventure: added a bit of text for session resumption and congestion control aspects as well as ECN usage.
  - \* from Kyle Rose: Revised privacy considerations text in Security Considerations Section

Changes in Version 08:

- o revised two sentences as suggested by Spencer Dawkins

Changes in Version 07:

- o revised some text for clarification according to comments from Spencer Dawkins

Changes in Version 06:

- o added Multicast Considerations section with input from Toerless Eckert
- o incorporated suggestions by David Black with respect to better reflect legacy CS1 handling

Changes in Version 05:

- o added scavenger service class into abstract
- o added some more history
- o added reference for "Myth of Over-Provisioning" in RFC3439 and references to presentations w.r.t. codepoint choices
- o added text to update draft-ietf-tsvwg-rtcweb-qos
- o revised text on congestion control in case of remarking to BE
- o added reference to DSCP measurement talk @IETF99
- o small typo fixes

## Changes in Version 04:

- o Several editorial changes according to review from Gorrry Fairhurst
- o Changed the section structure a bit (moved subsections 1.1 and 1.2 into own sections 3 and 7 respectively)
- o updated section 2 on requirements language
- o added updates to RFC 8325
- o tried to be more explicit what changes are required to RFCs 4594 and 8325

## Changes in Version 03:

- o Changed recommended codepoint to 000001
- o Added text to explain the reasons for the DSCP choice
- o Removed LE-min,LE-strict discussion
- o Added one more potential use case: reporting errors or telemetry data from OSs
- o Added privacy considerations to the security section (not worth an own section I think)
- o Changed IANA considerations section

## Changes in Version 02:

- o Applied many editorial suggestions from David Black
- o Added Multicast traffic use case
- o Clarified what is required for deployment in section 1.2 (Deployment Considerations)
- o Added text about implementations using AQMs and ECN usage
- o Updated IANA section according to David Black's suggestions
- o Revised text in the security section
- o Changed copyright Notice to pre5378Trust200902

## Changes in Version 01:



- o Now obsoletes RFC 3662.
- o Tried to be more precise in section 1.1 (Applicability) according to R. Geib's suggestions, so rephrased several paragraphs. Added text about congestion control
- o Change section 2 (PHB Description) according to R. Geib's suggestions.
- o Added RFC 2119 language to several sentences.
- o Detailed the description of remarking implications and recommendations in Section 8.
- o Added Section 10 to explicitly list changes with respect to RFC 4594, because this document will update it.

#### Appendix D. Note to RFC Editor

This section lists actions for the RFC editor during final formatting.

- o Apply the suggested changes of section Section 12 and add a normative reference in draft-ietf-tsvwg-rtcweb-qos to this RFC.
- o Delete Section 12.
- o Please replace the occurrences of RFCXXXX in Section 10 and Section 11 with the assigned RFC number for this document.
- o Delete Appendix C.
- o Delete this section.

#### Author's Address

Roland Bless  
Karlsruhe Institute of Technology (KIT)  
Kaiserstr. 12  
Karlsruhe 76131  
Germany

Phone: +49 721 608 46413  
Email: roland.bless@kit.edu

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: 28 April 2022

R. R. Stewart  
Netflix, Inc.  
M. Tüxen  
I. Rüngeler  
Münster Univ. of Appl. Sciences  
25 October 2021

Stream Control Transmission Protocol (SCTP) Network Address Translation  
Support  
draft-ietf-tsvwg-natsupp-23

Abstract

The Stream Control Transmission Protocol (SCTP) provides a reliable communications channel between two end-hosts in many ways similar to the Transmission Control Protocol (TCP). With the widespread deployment of Network Address Translators (NAT), specialized code has been added to NAT functions for TCP that allows multiple hosts to reside behind a NAT function and yet share a single IPv4 address, even when two hosts (behind a NAT function) choose the same port numbers for their connection. This additional code is sometimes classified as Network Address and Port Translation (NAPT).

This document describes the protocol extensions needed for the SCTP endpoints and the mechanisms for NAT functions necessary to provide similar features of NAPT in the single point and multipoint traversal scenario.

Finally, a YANG module for SCTP NAT is defined.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 28 April 2022.

## Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Conventions . . . . .	5
3. Terminology . . . . .	5
4. Motivation and Overview . . . . .	6
4.1. SCTP NAT Traversal Scenarios . . . . .	6
4.1.1. Single Point Traversal . . . . .	7
4.1.2. Multipoint Traversal . . . . .	7
4.2. Limitations of Classical NAPT for SCTP . . . . .	8
4.3. The SCTP-Specific Variant of NAT . . . . .	8
5. Data Formats . . . . .	13
5.1. Modified Chunks . . . . .	13
5.1.1. Extended ABORT Chunk . . . . .	13
5.1.2. Extended ERROR Chunk . . . . .	14
5.2. New Error Causes . . . . .	14
5.2.1. VTag and Port Number Collision Error Cause . . . . .	14
5.2.2. Missing State Error Cause . . . . .	15
5.2.3. Port Number Collision Error Cause . . . . .	15
5.3. New Parameters . . . . .	16
5.3.1. Disable Restart Parameter . . . . .	16
5.3.2. VTags Parameter . . . . .	17
6. Procedures for SCTP Endpoints and NAT Functions . . . . .	18
6.1. Association Setup Considerations for Endpoints . . . . .	19
6.2. Handling of Internal Port Number and Verification Tag Collisions . . . . .	19
6.2.1. NAT Function Considerations . . . . .	19
6.2.2. Endpoint Considerations . . . . .	20
6.3. Handling of Internal Port Number Collisions . . . . .	20
6.3.1. NAT Function Considerations . . . . .	20
6.3.2. Endpoint Considerations . . . . .	21
6.4. Handling of Missing State . . . . .	21
6.4.1. NAT Function Considerations . . . . .	22
6.4.2. Endpoint Considerations . . . . .	22

6.5.	Handling of Fragmented SCTP Packets by NAT Functions . .	24
6.6.	Multi Point Traversal Considerations for Endpoints . . .	24
7.	SCTP NAT YANG Module . . . . .	24
7.1.	Tree Structure . . . . .	24
7.2.	YANG Module . . . . .	25
8.	Various Examples of NAT Traversals . . . . .	27
8.1.	Single-homed Client to Single-homed Server . . . . .	28
8.2.	Single-homed Client to Multi-homed Server . . . . .	30
8.3.	Multihomed Client and Server . . . . .	32
8.4.	NAT Function Loses Its State . . . . .	35
8.5.	Peer-to-Peer Communications . . . . .	37
9.	Socket API Considerations . . . . .	42
9.1.	Get or Set the NAT Friendliness (SCTP_NAT_FRIENDLY) . . .	43
10.	IANA Considerations . . . . .	43
10.1.	New Chunk Flags for Two Existing Chunk Types . . . . .	43
10.2.	Three New Error Causes . . . . .	45
10.3.	Two New Chunk Parameter Types . . . . .	46
10.4.	One New URI . . . . .	46
10.5.	One New YANG Module . . . . .	46
11.	Security Considerations . . . . .	46
12.	Normative References . . . . .	47
13.	Informative References . . . . .	48
	Acknowledgments . . . . .	51
	Authors' Addresses . . . . .	51

## 1. Introduction

Stream Control Transmission Protocol (SCTP) [RFC4960] provides a reliable communications channel between two end-hosts in many ways similar to TCP [RFC0793]. With the widespread deployment of Network Address Translators (NAT), specialized code has been added to NAT functions for TCP that allows multiple hosts to reside behind a NAT function using private-use addresses (see [RFC6890]) and yet share a single IPv4 address, even when two hosts (behind a NAT function) choose the same port numbers for their connection. This additional code is sometimes classified as Network Address and Port Translation (NAPT). Please note that this document focuses on the case where the NAT function maps a single or multiple internal addresses to a single external address and vice versa.

To date, specialized code for SCTP has not yet been added to most NAT functions so that only a translation of IP addresses is supported. The end result of this is that only one SCTP-capable host can successfully operate behind such a NAT function and this host can only be single-homed. The only alternative for supporting legacy NAT functions is to use UDP encapsulation as specified in [RFC6951].

The NAT function in the document refers to NAPT functions described in Section 2.2 of [RFC3022], NAT64 [RFC6146], or DS-Lite AFTR [RFC6333].

This document specifies procedures allowing a NAT function to support SCTP by providing similar features to those provided by a NAPT for TCP (see [RFC5382] and [RFC7857]), UDP (see [RFC4787] and [RFC7857]), and ICMP (see [RFC5508] and [RFC7857]). This document also specifies a set of data formats for SCTP packets and a set of SCTP endpoint procedures to support NAT traversal. An SCTP implementation supporting these procedures can assure that in both single-homed and multi-homed cases a NAT function will maintain the appropriate state without the NAT function needing to change port numbers.

It is possible and desirable to make these changes for a number of reasons:

- \* It is desirable for SCTP internal end-hosts on multiple platforms to be able to share a NAT function's external IP address in the same way that a TCP session can use a NAT function.
- \* If a NAT function does not need to change any data within an SCTP packet, it will reduce the processing burden of NAT'ing SCTP by not needing to execute the CRC32c checksum used by SCTP.
- \* Not having to touch the IP payload makes the processing of ICMP messages by NAT functions easier.

An SCTP-aware NAT function will need to follow these procedures for generating appropriate SCTP packet formats.

When considering SCTP-aware NAT it is possible to have multiple levels of support. At each level, the Internal Host, Remote Host, and NAT function does or does not support the procedures described in this document. The following table illustrates the results of the various combinations of support and if communications can occur between two endpoints.

Internal Host	NAT Function	Remote Host	Communication
Support	Support	Support	Yes
Support	Support	No Support	Limited
Support	No Support	Support	None
Support	No Support	No Support	None
No Support	Support	Support	Limited
No Support	Support	No Support	Limited
No Support	No Support	Support	None
No Support	No Support	No Support	None

Table 1: Communication possibilities

From the table it can be seen that no communication can occur when a NAT function does not support SCTP-aware NAT. This assumes that the NAT function does not handle SCTP packets at all and all SCTP packets sent from behind a NAT function are discarded by the NAT function. In some cases, where the NAT function supports SCTP-aware NAT, but one of the two hosts does not support the feature, communication can possibly occur in a limited way. For example, only one host can have a connection when a collision case occurs.

## 2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 3. Terminology

This document uses the following terms, which are depicted in Figure 1. Familiarity with the terminology used in [RFC4960] and [RFC5061] is assumed.

Internal-Address (Int-Addr)

An internal address that is known to the internal host.

**Internal-Port (Int-Port)**

The port number that is in use by the host holding the Internal-Address.

**Internal-VTag (Int-VTag)**

The SCTP Verification Tag (VTag) (see Section 3.1 of [RFC4960]) that the internal host has chosen for an association. The VTag is a unique 32-bit tag that accompanies any incoming SCTP packet for this association to the Internal-Address.

**Remote-Address (Rem-Addr)**

The address that an internal host is attempting to contact.

**Remote-Port (Rem-Port)**

The port number used by the host holding the Remote-Address.

**Remote-VTag (Rem-VTag)**

The Verification Tag (VTag) (see Section 3.1 of [RFC4960]) that the host holding the Remote-Address has chosen for an association. The VTag is a unique 32-bit tag that accompanies any outgoing SCTP packet for this association to the Remote-Address.

**External-Address (Ext-Addr)**

An external address assigned to the NAT function, that it uses as a source address when sending packets towards a Remote-Address.

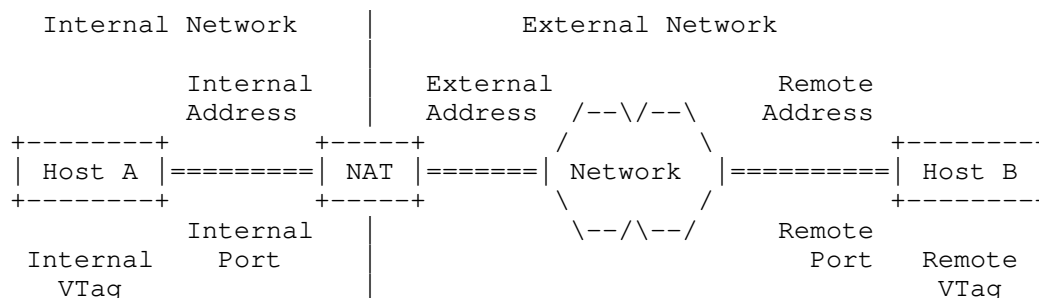


Figure 1: Basic Network Setup

## 4. Motivation and Overview

### 4.1. SCTP NAT Traversal Scenarios

This section defines the notion of single and multipoint NAT traversal.

#### 4.1.1. Single Point Traversal

In this case, all packets in the SCTP association go through a single NAT function, as shown in Figure 2.

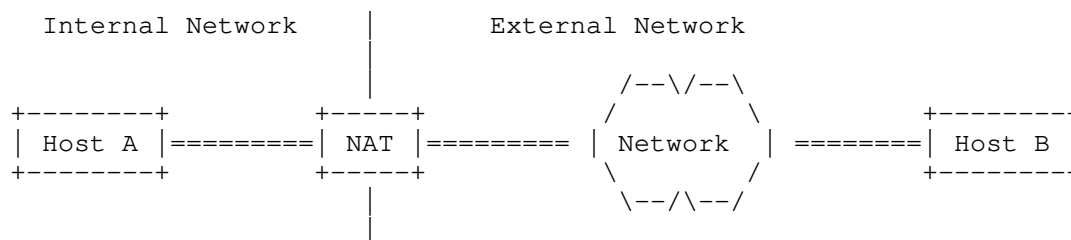


Figure 2: Single NAT Function Scenario

A variation of this case is shown in Figure 3, i.e., multiple NAT functions in the forwarding path between two endpoints.

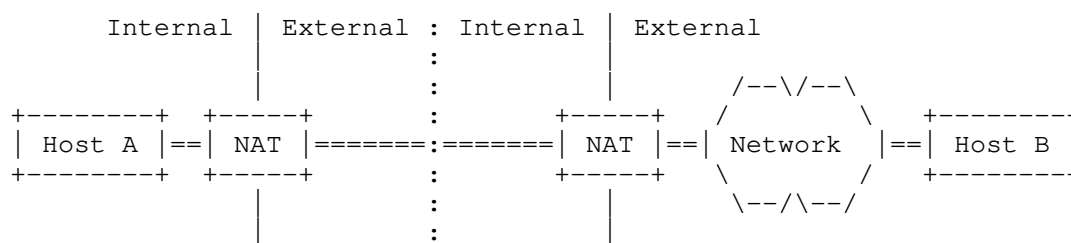


Figure 3: Serial NAT Functions Scenario

Although one of the main benefits of SCTP multi-homing is redundant paths, in the single point traversal scenario the NAT function represents a single point of failure in the path of the SCTP multi-homed association. However, the rest of the path can still benefit from path diversity provided by SCTP multi-homing.

The two SCTP endpoints in this case can be either single-homed or multi-homed. However, the important thing is that the NAT function in this case sees all the packets of the SCTP association.

#### 4.1.2. Multipoint Traversal

This case involves multiple NAT functions and each NAT function only sees some of the packets in the SCTP association. An example is shown in Figure 4.



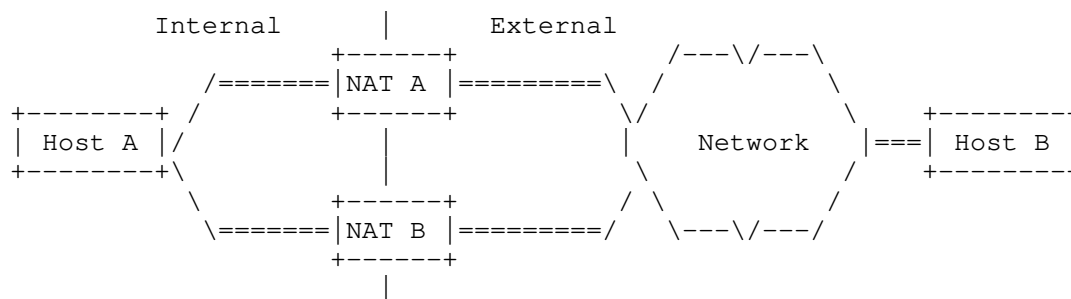


Figure 4: Parallel NAT Functions Scenario

This case does not apply to a single-homed SCTP association (i.e., both endpoints in the association use only one IP address). The advantage here is that the existence of multiple NAT traversal points can preserve the path diversity of a multi-homed association for the entire path. This in turn can improve the robustness of the communication.

#### 4.2. Limitations of Classical NAPT for SCTP

Using classical NAPT possibly results in changing one of the SCTP port numbers during the processing, which requires the recomputation of the transport layer checksum by the NAPT function. Whereas for UDP and TCP this can be done very efficiently, for SCTP the checksum (CRC32c) over the entire packet needs to be recomputed (see Appendix B of [RFC4960] for details of the CRC32c computation). This would considerably add to the NAT computational burden, however hardware support can mitigate this in some implementations.

An SCTP endpoint can have multiple addresses but only has a single port number to use. To make multipoint traversal work, all the NAT functions involved need to recognize the packets they see as belonging to the same SCTP association and perform port number translation in a consistent way. One possible way of doing this is to use a pre-defined table of port numbers and addresses configured within each NAT function. Other mechanisms could make use of NAT to NAT communication. Such mechanisms have not been deployed on a wide scale base and thus are not a preferred solution. Therefore an SCTP variant of NAT function has been developed (see Section 4.3).

#### 4.3. The SCTP-Specific Variant of NAT

In this section it is allowed that there are multiple SCTP capable hosts behind a NAT function that share one External-Address. Furthermore, this section focuses on the single point traversal scenario (see Section 4.1.1).

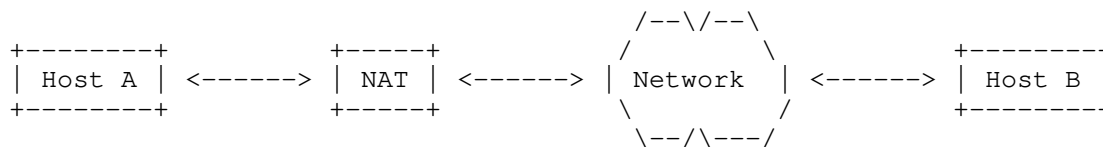
The modification of outgoing SCTP packets sent from an internal host is simple: the source address of the packets has to be replaced with the External-Address. It might also be necessary to establish some state in the NAT function to later handle incoming packets.

Typically, the NAT function has to maintain a NAT binding table of Internal-VTag, Internal-Port, Remote-VTag, Remote-Port, Internal-Address, and whether the restart procedure is disabled or not. An entry in that NAT binding table is called a NAT-State control block. The function Create() obtains the just mentioned parameters and returns a NAT-State control block. A NAT function MAY allow creating NAT-State control blocks via a management interface.

For SCTP packets coming from the external realm of the NAT function the destination address of the packets has to be replaced with the Internal-Address of the host to which the packet has to be delivered, if a NAT state entry is found. The lookup of the Internal-Address is based on the Remote-VTag, Remote-Port, Internal-VTag and the Internal-Port.

The entries in the NAT binding table need to fulfill some uniqueness conditions. There can not be more than one entry NAT binding table with the same pair of Internal-Port and Remote-Port. This rule can be relaxed, if all NAT binding table entries with the same Internal-Port and Remote-Port have the support for the restart procedure disabled (see Section 5.3.1). In this case there can not be no more than one entry with the same Internal-Port, Remote-Port and Remote-VTag and no more than one NAT binding table entry with the same Internal-Port, Remote-Port, and Int-VTag.

The processing of outgoing SCTP packets containing an INIT chunk is illustrated in the following figure. This scenario is valid for all message flows in this section.



```

INIT[Initiate-Tag]
Int-Addr:Int-Port -----> Rem-Addr:Rem-Port
Rem-VTag=0

Create(Initiate-Tag, Int-Port, 0, Rem-Port, Int-Addr,
      IsRestartDisabled)
Returns(NAT-State control block)

```

Translate To:

```

INIT[Initiate-Tag]
Ext-Addr:Int-Port -----> Rem-Addr:Rem-Port
Rem-VTag=0

```

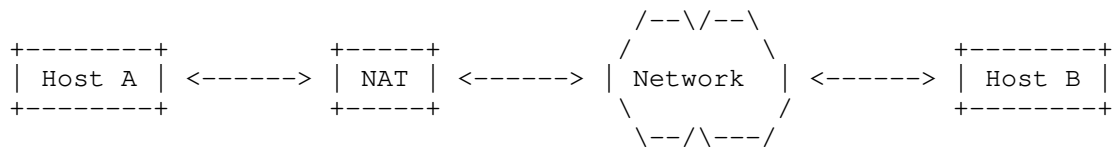
Normally a NAT binding table entry will be created.

However, it is possible that there is already a NAT binding table entry with the same Remote-Port, Internal-Port, and Internal-VTag but different Internal-Address and the restart procedure is disabled. In this case the packet containing the INIT chunk MUST be dropped by the NAT and a packet containing an ABORT chunk SHOULD be sent to the SCTP host that originated the packet with the M bit set and 'VTag and Port Number Collision' error cause (see Section 5.1.1 for the format). The source address of the packet containing the ABORT chunk MUST be the destination address of the packet containing the INIT chunk.

If an outgoing SCTP packet contains an INIT or ASCONF chunk and a matching NAT binding table entry is found, the packet is processed as a normal outgoing packet.

It is also possible that a NAT binding table entry with the same Remote-Port and Internal-Port exists without an Internal-VTag conflict but there exists a NAT binding table entry with the same port numbers but a different Internal-Address and the restart procedure is not disabled. In such a case the packet containing the INIT chunk MUST be dropped by the NAT function and a packet containing an ABORT chunk SHOULD be sent to the SCTP host that originated the packet with the M bit set and 'Port Number Collision' error cause (see Section 5.1.1 for the format).

The processing of outgoing SCTP packets containing no INIT chunks is described in the following figure.

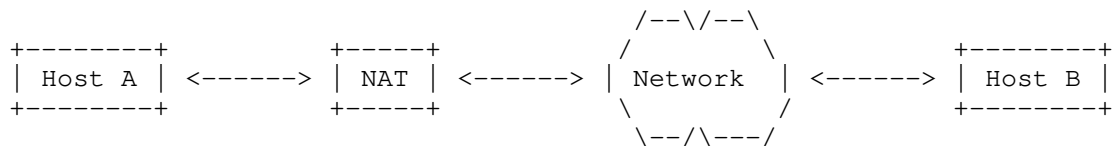


Int-Addr:Int-Port -----> Rem-Addr:Rem-Port  
                                   Rem-VTag

Translate To:

Ext-Addr:Int-Port -----> Rem-Addr:Rem-Port  
                                   Rem-VTag

The processing of incoming SCTP packets containing an INIT ACK chunk is illustrated in the following figure. The Lookup() function has as input the Internal-VTag, Internal-Port, Remote-VTag, and Remote-Port. It returns the corresponding entry of the NAT binding table and updates the Remote-VTag by substituting it with the value of the Initiate-Tag of the INIT ACK chunk. The wildcard character signifies that the parameter's value is not considered in the Lookup() function or changed in the Update() function, respectively.



INIT ACK[Initiate-Tag]  
 Ext-Addr:Int-Port <---- Rem-Addr:Rem-Port  
                                   Int-VTag

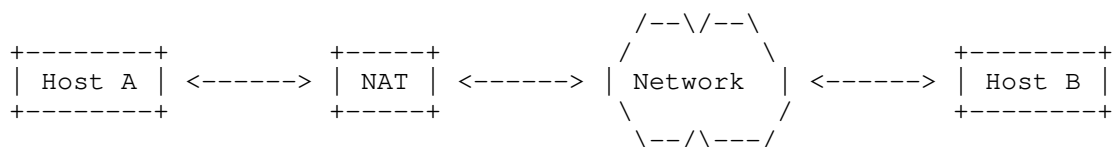
Lookup(Int-VTag, Int-Port, \*, Rem-Port)  
 Update(\*, \*, Initiate-Tag, \*)

Returns(NAT-State control block containing Int-Addr)

INIT ACK[Initiate-Tag]  
 Int-Addr:Int-Port <----- Rem-Addr:Rem-Port  
                                   Int-VTag

In the case where the Lookup function fails because it does not find an entry, the SCTP packet is dropped. If it succeeds, the Update routine inserts the Remote-VTag (the Initiate-Tag of the INIT ACK chunk) in the NAT-State control block.

The processing of incoming SCTP packets containing an ABORT or SHUTDOWN COMPLETE chunk with the T bit set is illustrated in the following figure.



Ext-Addr:Int-Port <----- Rem-Addr:Rem-Port  
Rem-VTag

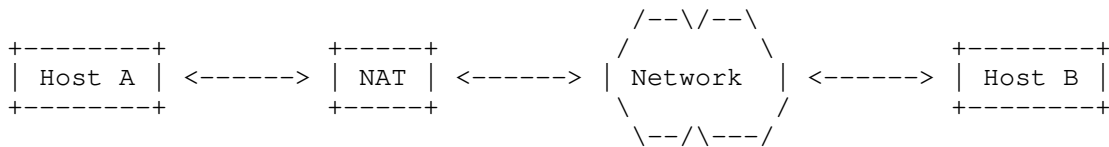
Lookup(\*, Int-Port, Rem-VTag, Rem-Port)

Returns (NAT-State control block containing Int-Addr)

Int-Addr:Int-Port <----- Rem-Addr:Rem-Port  
Rem-VTag

For an incoming packet containing an INIT chunk a table lookup is made only based on the addresses and port numbers. If an entry with a Remote-VTag of zero is found, it is considered a match and the Remote-VTag is updated. If an entry with a non-matching Remote-VTag is found or no entry is found, the incoming packet is silently dropped. If an entry with a matching Remote-VTag is found, the incoming packet is forwarded. This allows the handling of INIT collision through NAT functions.

The processing of other incoming SCTP packets is described in the following figure.



Ext-Addr: Int-Port <----- Rem-Addr: Rem-Port  
Int-VTag

Lookup(Int-VTag, Int-Port, \*, Rem-Port)

Returns(NAT-State control block containing Internal-Address)

Int-Addr: Int-Port <----- Rem-Addr: Rem-Port  
Int-VTag

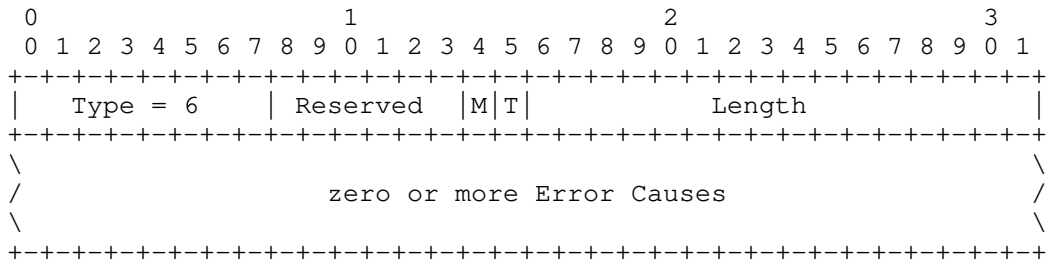
5. Data Formats

This section defines the formats used to support NAT traversal. Section 5.1 and Section 5.2 describe chunks and error causes sent by NAT functions and received by SCTP endpoints. Section 5.3 describes parameters sent by SCTP endpoints and used by NAT functions and SCTP endpoints.

5.1. Modified Chunks

This section presents existing chunks defined in [RFC4960] for which additional flags are specified by this document.

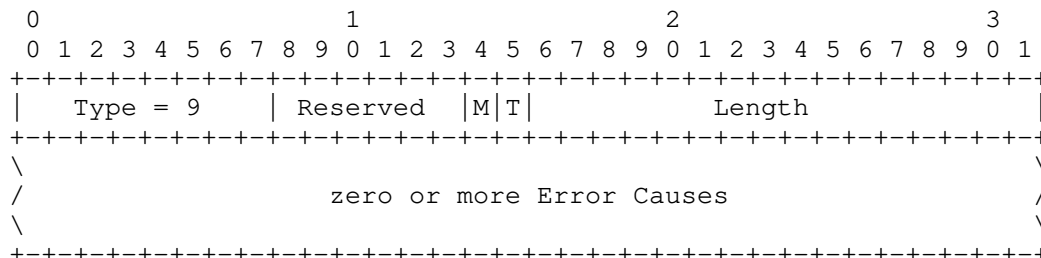
5.1.1. Extended ABORT Chunk



The ABORT chunk is extended to add the new 'M bit'. The M bit indicates to the receiver of the ABORT chunk that the chunk was not generated by the peer SCTP endpoint, but instead by a middle box (e.g., NAT).

[NOTE to RFC-Editor: Assignment of M bit to be confirmed by IANA.]

## 5.1.2. Extended ERROR Chunk



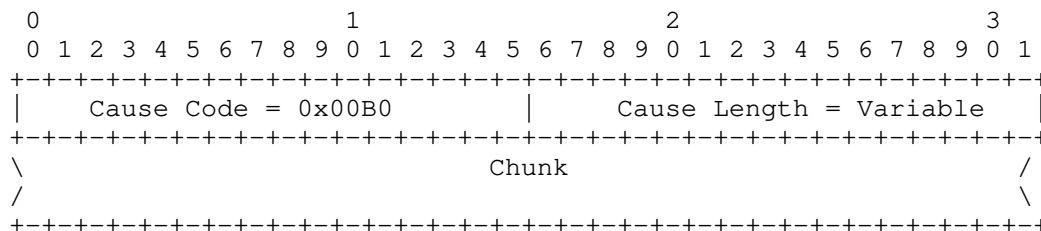
The ERROR chunk defined in [RFC4960] is extended to add the new 'M bit'. The M bit indicates to the receiver of the ERROR chunk that the chunk was not generated by the peer SCTP endpoint, but instead by a middle box.

[NOTE to RFC-Editor: Assignment of M bit to be confirmed by IANA.]

## 5.2. New Error Causes

This section defines the new error causes added by this document.

## 5.2.1. VTag and Port Number Collision Error Cause



Cause Code: 2 bytes (unsigned integer)

This field holds the IANA defined cause code for the 'VTag and Port Number Collision' Error Cause. IANA is requested to assign the value 0x00B0 for this cause code.

Cause Length: 2 bytes (unsigned integer)

This field holds the length in bytes of the error cause. The value MUST be the length of the Cause-Specific Information plus 4.

Chunk: variable length

The Cause-Specific Information is filled with the chunk that caused this error. This can be an INIT, INIT ACK, or ASCONF chunk. Note that if the entire chunk will not fit in the ERROR chunk or ABORT chunk being sent then the bytes that do not fit are truncated.

[NOTE to RFC-Editor: Assignment of cause code to be confirmed by IANA.]

#### 5.2.2. Missing State Error Cause

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
Cause Code = 0x00B1										Cause Length = Variable																													
Original Packet																																							

Cause Code: 2 bytes (unsigned integer)

This field holds the IANA defined cause code for the 'Missing State' Error Cause. IANA is requested to assign the value 0x00B1 for this cause code.

Cause Length: 2 bytes (unsigned integer)

This field holds the length in bytes of the error cause. The value MUST be the length of the Cause-Specific Information plus 4.

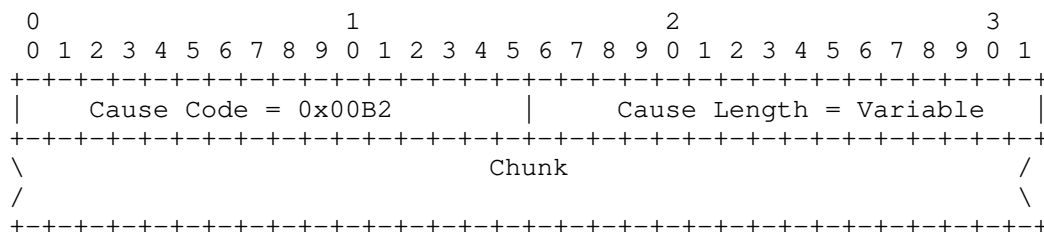
Original Packet: variable length

The Cause-Specific Information is filled with the IPv4 or IPv6 packet that caused this error. The IPv4 or IPv6 header MUST be included. Note that if the packet will not fit in the ERROR chunk or ABORT chunk being sent then the bytes that do not fit are truncated.

[NOTE to RFC-Editor: Assignment of cause code to be confirmed by IANA.]

#### 5.2.3. Port Number Collision Error Cause





Cause Code: 2 bytes (unsigned integer)

This field holds the IANA defined cause code for the 'Port Number Collision' Error Cause. IANA is requested to assign the value 0x00B2 for this cause code.

Cause Length: 2 bytes (unsigned integer)

This field holds the length in bytes of the error cause. The value MUST be the length of the Cause-Specific Information plus 4.

Chunk: variable length

The Cause-Specific Information is filled with the chunk that caused this error. This can be an INIT, INIT ACK, or ASCONF chunk. Note that if the entire chunk will not fit in the ERROR chunk or ABORT chunk being sent then the bytes that do not fit are truncated.

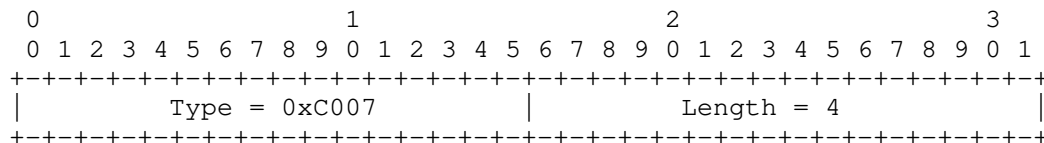
[NOTE to RFC-Editor: Assignment of cause code to be confirmed by IANA.]

### 5.3. New Parameters

This section defines new parameters and their valid appearance defined by this document.

#### 5.3.1. Disable Restart Parameter

This parameter is used to indicate that the restart procedure is requested to be disabled. Both endpoints of an association MUST include this parameter in the INIT chunk and INIT ACK chunk when establishing an association and MUST include it in the ASCONF chunk when adding an address to successfully disable the restart procedure.



Parameter Type: 2 bytes (unsigned integer)

This field holds the IANA defined parameter type for the Disable Restart Parameter. IANA is requested to assign the value 0xC007 for this parameter type.

Parameter Length: 2 bytes (unsigned integer)

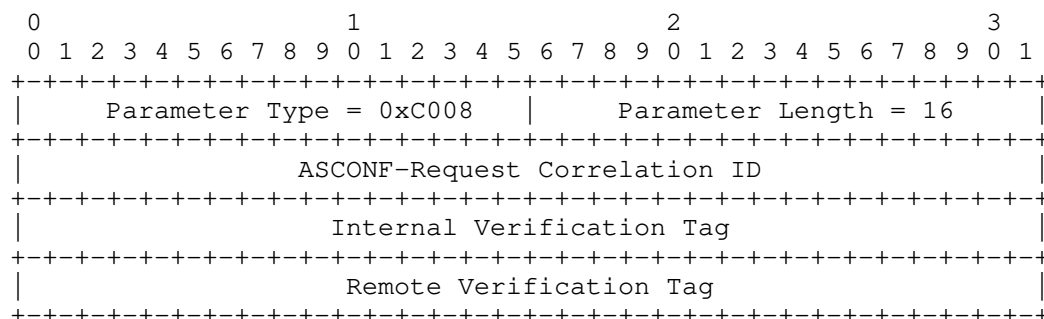
This field holds the length in bytes of the parameter. The value MUST be 4.

[NOTE to RFC-Editor: Assignment of parameter type to be confirmed by IANA.]

The Disable Restart Parameter MAY appear in INIT, INIT ACK and ASCONF chunks and MUST NOT appear in any other chunk.

### 5.3.2. VTags Parameter

This parameter is used to help a NAT function to recover from state loss.



Parameter Type: 2 bytes (unsigned integer)

This field holds the IANA defined parameter type for the VTags Parameter. IANA is requested to assign the value 0xC008 for this parameter type.

Parameter Length: 2 bytes (unsigned integer)

This field holds the length in bytes of the parameter. The value MUST be 16.

ASCONF-Request Correlation ID: 4 bytes (unsigned integer)

This is an opaque integer assigned by the sender to identify each request parameter. The receiver of the ASCONF Chunk will copy this 32-bit value into the ASCONF Response Correlation ID field of the ASCONF ACK response parameter. The sender of the packet containing the ASCONF chunk can use this same value in the ASCONF ACK chunk to find which request the response is for. The receiver MUST NOT change the value of the ASCONF-Request Correlation ID.

Internal Verification Tag: 4 bytes (unsigned integer)

The Verification Tag that the internal host has chosen for the association. The Verification Tag is a unique 32-bit tag that accompanies any incoming SCTP packet for this association to the Internal-Address.

Remote Verification Tag: 4 bytes (unsigned integer)

The Verification Tag that the host holding the Remote-Address has chosen for the association. The VTag is a unique 32-bit tag that accompanies any outgoing SCTP packet for this association to the Remote-Address.

[NOTE to RFC-Editor: Assignment of parameter type to be confirmed by IANA.]

The VTags Parameter MAY appear in ASCONF chunks and MUST NOT appear in any other chunk.

## 6. Procedures for SCTP Endpoints and NAT Functions

If an SCTP endpoint is behind an SCTP-aware NAT, a number of problems can arise as it tries to communicate with its peers:

- \* IP addresses can not be included in the SCTP packet. This is discussed in Section 6.1.
- \* More than one host behind a NAT function could select the same VTag and source port number when communicating with the same peer server. This creates a situation where the NAT function will not be able to tell the two associations apart. This situation is discussed in Section 6.2.
- \* If an SCTP endpoint is a server communicating with multiple peers and the peers are behind the same NAT function, then these peers cannot be distinguished by the server. This case is discussed in Section 6.3.
- \* A restart of a NAT function during a conversation could cause a loss of its state. This problem and its solution is discussed in Section 6.4.
- \* NAT functions need to deal with SCTP packets being fragmented at the IP layer. This is discussed in Section 6.5.
- \* An SCTP endpoint can be behind two NAT functions in parallel providing redundancy. The method to set up this scenario is discussed in Section 6.6.

The mechanisms to solve these problems require additional chunks and parameters, defined in this document, and modified handling procedures from those specified in [RFC4960] as described below.

#### 6.1. Association Setup Considerations for Endpoints

The association setup procedure defined in [RFC4960] allows multi-homed SCTP endpoints to exchange its IP-addresses by using IPv4 or IPv6 address parameters in the INIT and INIT ACK chunks. However, this does not work when NAT functions are present.

Every association setup from a host behind a NAT function MUST NOT use multiple internal addresses. The INIT chunk MUST NOT contain an IPv4 Address parameter, IPv6 Address parameter, or Supported Address Types parameter. The INIT ACK chunk MUST NOT contain any IPv4 Address parameter or IPv6 Address parameter using non-global addresses. The INIT chunk and the INIT ACK chunk MUST NOT contain any Host Name parameters.

If the association is intended to be finally multi-homed, the procedure in Section 6.6 MUST be used.

The INIT and INIT ACK chunk SHOULD contain the Disable Restart parameter defined in Section 5.3.1.

#### 6.2. Handling of Internal Port Number and Verification Tag Collisions

Consider the case where two hosts in the Internal-Address space want to set up an SCTP association with the same service provided by some remote hosts. This means that the Remote-Port is the same. If they both choose the same Internal-Port and Internal-VTag, the NAT function cannot distinguish between incoming packets anymore. However, this is unlikely. The Internal-VTags are chosen at random and if the Internal-Ports are also chosen from the ephemeral port range at random (see [RFC6056]) this gives a 46-bit random number that has to match.

The same can happen with the Remote-VTag when a packet containing an INIT ACK chunk or an ASCONF chunk is processed by the NAT function.

##### 6.2.1. NAT Function Considerations

If the NAT function detects a collision of internal port numbers and verification tags, it SHOULD send a packet containing an ABORT chunk with the M bit set if the collision is triggered by a packet containing an INIT or INIT ACK chunk. If such a collision is triggered by a packet containing an ASCONF chunk, it SHOULD send a packet containing an ERROR chunk with the M bit. The M bit is a new

bit defined by this document to express to SCTP that the source of this packet is a "middle" box, not the peer SCTP endpoint (see Section 5.1.1). If a packet containing an INIT ACK chunk triggers the collision, the corresponding packet containing the ABORT chunk MUST contain the same source and destination address and port numbers as the packet containing the INIT ACK chunk. If a packet containing an INIT chunk or an ASCONF chunk, the source and destination address and port numbers MUST be swapped.

The sender of the packet containing an ERROR or ABORT chunk MUST include the error cause with cause code 'VTag and Port Number Collision' (see Section 5.2.1).

#### 6.2.2. Endpoint Considerations

The sender of the packet containing the INIT chunk or the receiver of a packet containing the INIT ACK chunk, upon reception of a packet containing an ABORT chunk with M bit set and the appropriate error cause code for colliding NAT binding table state is included, SHOULD reinitiate the association setup procedure after choosing a new initiate tag, if the association is in COOKIE-WAIT state. In any other state, the SCTP endpoint MUST NOT respond.

The sender of the packet containing the ASCONF chunk, upon reception of a packet containing an ERROR chunk with M bit set, MUST stop adding the path to the association.

#### 6.3. Handling of Internal Port Number Collisions

When two SCTP hosts are behind an SCTP-aware NAT it is possible that two SCTP hosts in the Internal-Address space will want to set up an SCTP association with the same server running on the same remote host. If the two hosts choose the same internal port, this is considered an internal port number collision.

For the NAT function, appropriate tracking can be performed by assuring that the VTags are unique between the two hosts.

##### 6.3.1. NAT Function Considerations

The NAT function, when processing the packet containing the INIT ACK chunk, SHOULD note in its NAT binding table if the association supports the disable restart extension. This note is used when establishing future associations (i.e. when processing a packet containing an INIT chunk from an internal host) to decide if the connection can be allowed. The NAT function does the following when processing a packet containing an INIT chunk:

- \* If the packet containing the INIT chunk is originating from an internal port to a remote port for which the NAT function has no matching NAT binding table entry, it MUST allow the packet containing the INIT chunk creating an NAT binding table entry.
- \* If the packet containing the INIT chunk matches an existing NAT binding table entry, it MUST validate that the disable restart feature is supported and, if it does, allow the packet containing the INIT chunk to be forwarded.
- \* If the disable restart feature is not supported, the NAT function SHOULD send a packet containing an ABORT chunk with the M bit set.

The 'Port Number Collision' error cause (see Section 5.2.3) MUST be included in the ABORT chunk sent in response to the packet containing an INIT chunk.

If the collision is triggered by a packet containing an ASCONF chunk, a packet containing an ERROR chunk with the 'Port Number Collision' error cause SHOULD be sent in response to the packet containing the ASCONF chunk.

#### 6.3.2. Endpoint Considerations

For the remote SCTP server this means that the Remote-Port and the Remote-Address are the same. If they both have chosen the same Internal-Port the server cannot distinguish between both associations based on the address and port numbers. For the server it looks like the association is being restarted. To overcome this limitation the client sends a Disable Restart parameter in the INIT chunk.

When the server receives this parameter it does the following:

- \* It MUST include a Disable Restart parameter in the INIT ACK to inform the client that it will support the feature.
- \* It MUST disable the restart procedures defined in [RFC4960] for this association.

Servers that support this feature will need to be capable of maintaining multiple connections to what appears to be the same peer (behind the NAT function) differentiated only by the VTags.

#### 6.4. Handling of Missing State

#### 6.4.1. NAT Function Considerations

If the NAT function receives a packet from the internal network for which the lookup procedure does not find an entry in the NAT binding table, a packet containing an ERROR chunk SHOULD be sent back with the M bit set. The source address of the packet containing the ERROR chunk MUST be the destination address of the packet received from the internal network. The verification tag is reflected and the T bit is set. Such a packet containing an ERROR chunk SHOULD NOT be sent if the received packet contains an ASCONF chunk with the VTags parameter or an ABORT, SHUTDOWN COMPLETE or INIT ACK chunk. A packet containing an ERROR chunk MUST NOT be sent if the received packet contains an ERROR chunk with the M bit set. In any case, the packet SHOULD NOT be forwarded to the remote address.

If the NAT function receives a packet from the internal network for which it has no NAT binding table entry and the packet contains an ASCONF chunk with the VTags parameter, the NAT function MUST update its NAT binding table according to the verification tags in the VTags parameter and, if present, the Disable Restart parameter.

When sending a packet containing an ERROR chunk, the error cause 'Missing State' (see Section 5.2.2) MUST be included and the M bit of the ERROR chunk MUST be set (see Section 5.1.2).

#### 6.4.2. Endpoint Considerations

Upon reception of this packet containing the ERROR chunk by an SCTP endpoint the receiver takes the following actions:

- \* It SHOULD validate that the verification tag is reflected by looking at the VTag that would have been included in an outgoing packet. If the validation fails, discard the received packet containing the ERROR chunk.
- \* It SHOULD validate that the peer of the SCTP association supports the dynamic address extension. If the validation fails, discard the received packet containing the ERROR chunk.
- \* It SHOULD generate a packet containing a new ASCONF chunk containing the VTags parameter (see Section 5.3.2) and the Disable Restart parameter (see Section 5.3.1) if the association is using the disable restart feature. By processing this packet the NAT function can recover the appropriate state. The procedures for generating an ASCONF chunk can be found in [RFC5061].

The peer SCTP endpoint receiving such a packet containing an ASCONF chunk SHOULD add the address and respond with an acknowledgment if the address is new to the association (following all procedures defined in [RFC5061]). If the address is already part of the association, the SCTP endpoint MUST NOT respond with an error, but instead SHOULD respond with a packet containing an ASCONF ACK chunk acknowledging the address and take no action (since the address is already in the association).

Note that it is possible that upon receiving a packet containing an ASCONF chunk containing the VTags parameter the NAT function will realize that it has an 'Internal Port Number and Verification Tag collision'. In such a case the NAT function SHOULD send a packet containing an ERROR chunk with the error cause code set to 'VTag and Port Number Collision' (see Section 5.2.1).

If an SCTP endpoint receives a packet containing an ERROR chunk with 'Internal Port Number and Verification Tag collision' as the error cause and the packet in the Error Chunk contains an ASCONF with the VTags parameter, careful examination of the association is necessary. The endpoint does the following:

- \* It MUST validate that the verification tag is reflected by looking at the VTag that would have been included in the outgoing packet. If the validation fails, it MUST discard the packet.
- \* It MUST validate that the peer of the SCTP association supports the dynamic address extension. If the peer does not support this extension, it MUST discard the received packet containing the ERROR chunk.
- \* If the association is attempting to add an address (i.e. following the procedures in Section 6.6) then the endpoint MUST NOT consider the address part of the association and SHOULD make no further attempt to add the address (i.e. cancel any ASCONF timers and remove any record of the path), since the NAT function has a VTag collision and the association cannot easily create a new VTag (as it would if the error occurred when sending a packet containing an INIT chunk).
- \* If the endpoint has no other path, i.e. the procedure was executed due to missing a state in the NAT function, then the endpoint MUST abort the association. This would occur only if the local NAT function restarted and accepted a new association before attempting to repair the missing state (Note that this is no different than what happens to all TCP connections when a NAT function loses its state).



### 6.5. Handling of Fragmented SCTP Packets by NAT Functions

SCTP minimizes the use of IP-level fragmentation. However, it can happen that using IP-level fragmentation is needed to continue an SCTP association. For example, if the path MTU is reduced and there are still some DATA chunk in flight, which require packets larger than the new path MTU. If IP-level fragmentation can not be used, the SCTP association will be terminated in a non-graceful way. See [RFC8900] for more information about IP fragmentation.

Therefore, a NAT function MUST be able to handle IP-level fragmented SCTP packets. The fragments MAY arrive in any order.

When an SCTP packet can not be forwarded by the NAT function due to MTU issues and the IP header forbids fragmentation, the NAT MUST send back a "Fragmentation needed and DF set" ICMPv4 or PTB ICMPv6 message to the internal host. This allows for a faster recovery from this packet drop.

### 6.6. Multi Point Traversal Considerations for Endpoints

If a multi-homed SCTP endpoint behind a NAT function connects to a peer, it MUST first set up the association single-homed with only one address causing the first NAT function to populate its state. Then it SHOULD add each IP address using packets containing ASCONF chunks sent via their respective NAT functions. The address used in the Add IP address parameter is the wildcard address (0.0.0.0 or ::0) and the address parameter in the ASCONF chunk SHOULD also contain the VTags parameter and optionally the Disable Restart parameter.

## 7. SCTP NAT YANG Module

This section defines a YANG module for SCTP NAT.

The terminology for describing YANG data models is defined in [RFC7950]. The meaning of the symbols in tree diagrams is defined in [RFC8340].

### 7.1. Tree Structure

This module augments NAT YANG module [RFC8512] with SCTP specifics. The module supports both classical SCTP NAT (that is, rewrite port numbers) and SCTP-specific variant where the ports numbers are not altered. The YANG "feature" is used to indicate whether SCTP-specific variant is supported.

The tree structure of the SCTP NAT YANG module is provided below:

```

module: ietf-nat-sctp
  augment /nat:nat/nat:instances/nat:instance
    /nat:policy/nat:timers:
      +--rw sctp-timeout?  uint32
  augment /nat:nat/nat:instances/nat:instance
    /nat:mapping-table/nat:mapping-entry:
      +--rw int-VTag?      uint32 {sctp-nat}?
      +--rw rem-VTag?      uint32 {sctp-nat}?

```

Concretely, the SCTP NAT YANG module augments the NAT YANG module (policy, in particular) with the following:

- \* The sctp-timeout is used to control the SCTP inactivity timeout. That is, the time an SCTP mapping will stay active without SCTP packets traversing the NAT. This timeout can be set only for SCTP. Hence, `"/nat:nat/nat:instances/nat:instance/nat:policy/nat:transport-protocols/nat:protocol-id"` MUST be set to `'132'` (SCTP).

In addition, the SCTP NAT YANG module augments the mapping entry with the following parameters defined in Section 3. These parameters apply only for SCTP NAT mapping entries (i.e., `"/nat/instances/instance/mapping-table/mapping-entry/transport-protocol"` MUST be set to `'132'`);

- \* The Internal Verification Tag (Int-VTag)
- \* The Remote Verification Tag (Rem-VTag)

## 7.2. YANG Module

```

<CODE BEGINS> file "ietf-nat-sctp@2020-11-02.yang"
module ietf-nat-sctp {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-nat-sctp";
  prefix nat-sctp;

  import ietf-nat {
    prefix nat;
    reference
      "RFC 8512: A YANG Module for Network Address Translation
       (NAT) and Network Prefix Translation (NPT)";
  }

  organization
    "IETF TSVWG Working Group";
  contact
    "WG Web:  <https://datatracker.ietf.org/wg/tsvwg/>

```

WG List: <mailto:tsvwg@ietf.org>

Author: Mohamed Boucadair  
<mailto:mohamed.boucadair@orange.com>;

description

"This module augments NAT YANG module with Stream Control Transmission Protocol (SCTP) specifics. The extension supports both a classical SCTP NAT (that is, rewrite port numbers) and a, SCTP-specific variant where the ports numbers are not altered.

Copyright (c) 2020 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices.";

```
revision 2019-11-18 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: Stream Control Transmission Protocol (SCTP)
      Network Address Translation Support";
}

feature sctp-nat {
  description
    "This feature means that SCTP-specific variant of NAT
      is supported. That is, avoid rewriting port numbers.";
  reference
    "Section 4.3 of RFC XXXX.";
}

augment "/nat:nat/nat:instances/nat:instance"
  + "/nat:policy/nat:timers" {
  when "/nat:nat/nat:instances/nat:instance"
    + "/nat:policy/nat:transport-protocols"
    + "/nat:protocol-id = 132";
  description
    "Extends NAT policy with a timeout for SCTP mapping
      entries.";
```

```
    leaf sctp-timeout {
      type uint32;
      units "seconds";
      description
        "SCTP inactivity timeout. That is, the time an SCTP
        mapping entry will stay active without packets
        traversing the NAT.";
    }
  }

  augment "/nat:nat/nat:instances/nat:instance"
    + "/nat:mapping-table/nat:mapping-entry" {
    when "nat:transport-protocol = 132";
    if-feature "sctp-nat";
    description
      "Extends the mapping entry with SCTP specifics.";

    leaf int-VTag {
      type uint32;
      description
        "The Internal Verification Tag that the internal
        host has chosen for this communication.";
    }
    leaf rem-VTag {
      type uint32;
      description
        "The Remote Verification Tag that the remote
        peer has chosen for this communication.";
    }
  }
}
<CODE ENDS>
```

## 8. Various Examples of NAT Traversals

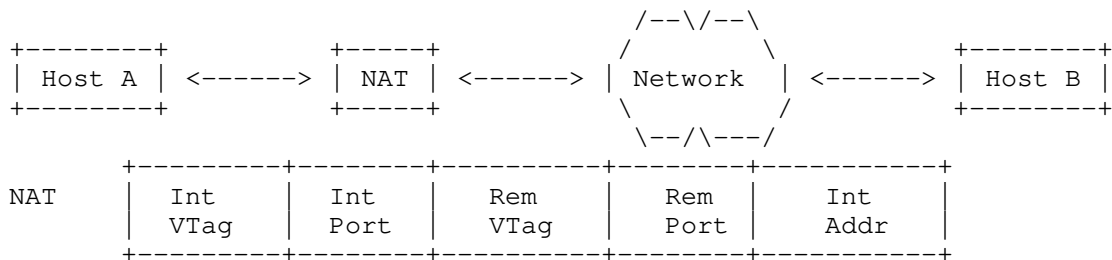
Please note that this section is informational only.

The addresses being used in the following examples are IPv4 addresses for private-use networks and for documentation as specified in [RFC6890]. However, the method described here is not limited to this NAT44 case.

The NAT binding table entries shown in the following examples do not include the flag indicating whether the restart procedure is supported or not. This flag is not relevant for these examples.

## 8.1. Single-homed Client to Single-homed Server

The internal client starts the association with the remote server via a four-way-handshake. Host A starts by sending a packet containing an INIT chunk.



```
INIT[Initiate-Tag = 1234]
10.0.0.1:1 -----> 203.0.113.1:2
    Rem-VTtag = 0
```

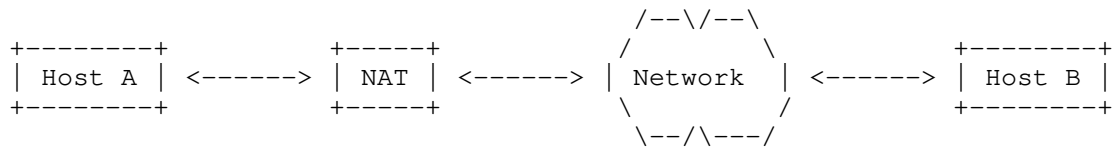
A NAT binding table entry is created, the source address is substituted and the packet is sent on:

NAT function creates entry:

NAT	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	0	2	10.0.0.1

```
INIT[Initiate-Tag = 1234]
192.0.2.1:1 -----> 203.0.113.1:2
    Rem-VTtag = 0
```

Host B receives the packet containing an INIT chunk and sends a packet containing an INIT ACK chunk with the NAT's Remote-address as destination address.



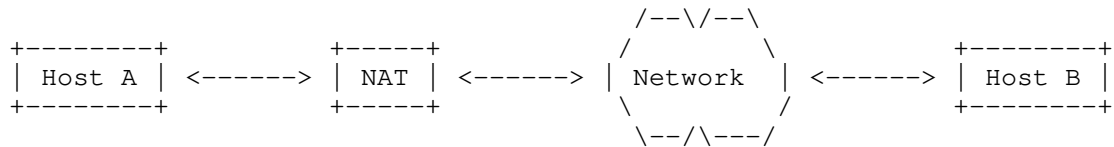
INIT ACK[Initiate-Tag = 5678]  
 192.0.2.1:1 <----- 203.0.113.1:2  
 Int-VTag = 1234

NAT function updates entry:

NAT	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	5678	2	10.0.0.1

INIT ACK[Initiate-Tag = 5678]  
 10.0.0.1:1 <----- 203.0.113.1:2  
 Int-VTag = 1234

The handshake finishes with a COOKIE ECHO acknowledged by a COOKIE ACK.



COOKIE ECHO  
 10.0.0.1:1 -----> 203.0.113.1:2  
 Rem-VTag = 5678

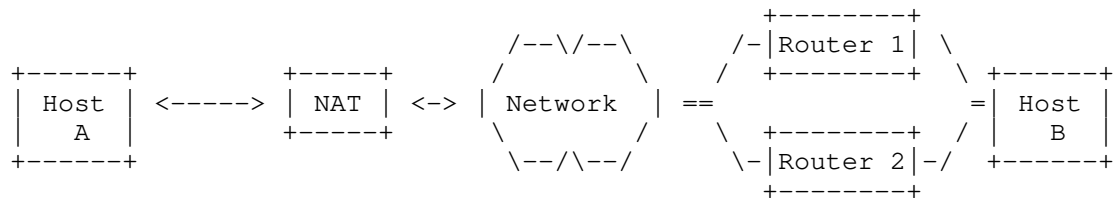
COOKIE ECHO  
 192.0.2.1:1 -----> 203.0.113.1:2  
 Rem-VTag = 5678

COOKIE ACK  
 192.0.2.1:1 <----- 203.0.113.1:2  
 Int-VTag = 1234

COOKIE ACK  
 10.0.0.1:1 <----- 203.0.113.1:2  
 Int-VTag = 1234

## 8.2. Single-homed Client to Multi-homed Server

The internal client is single-homed whereas the remote server is multi-homed. The client (Host A) sends a packet containing an INIT chunk like in the single-homed case.



NAT	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
-----	-------------	-------------	-------------	-------------	-------------

```

INIT[Initiate-Tag = 1234]
10.0.0.1:1 ---> 203.0.113.1:2
    Rem-VTag = 0
  
```

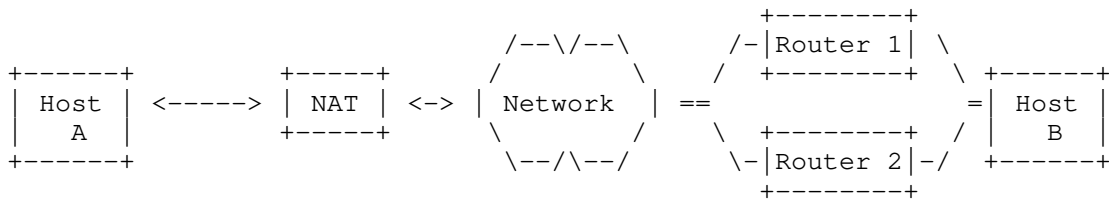
NAT function creates entry:

NAT	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	0	2	10.0.0.1

```

                                INIT[Initiate-Tag = 1234]
192.0.2.1:1 -----> 203.0.113.1:2
                        Rem-VTag = 0
  
```

The server (Host B) includes its two addresses in the INIT ACK chunk.



```
INIT ACK[Initiate-tag = 5678, IP-Addr = 203.0.113.129]
192.0.2.1:1 <----- 203.0.113.1:2
Int-VTag = 1234
```

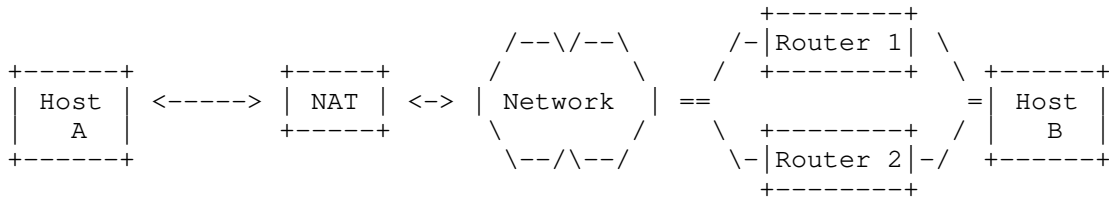
The NAT function does not need to change the NAT binding table for the second address:

NAT	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	5678	2	10.0.0.1

```
INIT ACK[Initiate-Tag = 5678]
10.0.0.1:1 <--- 203.0.113.1:2
Int-VTag = 1234
```

The handshake finishes with a COOKIE ECHO acknowledged by a COOKIE ACK.





COOKIE ECHO  
10.0.0.1:1 ---> 203.0.113.1:2  
Rem-VTag = 5678

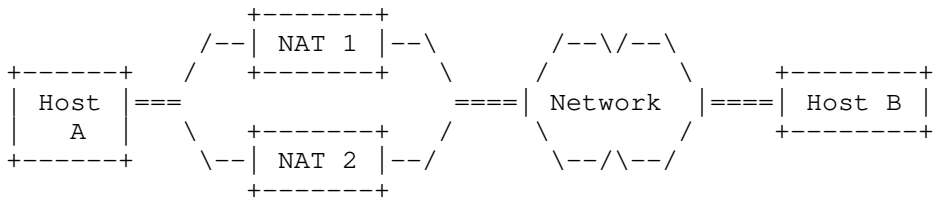
COOKIE ECHO  
192.0.2.1:1 -----> 203.0.113.1:2  
Rem-VTag = 5678

COOKIE ACK  
192.0.2.1:1 <----- 203.0.113.1:2  
Int-VTag = 1234

COOKIE ACK  
10.0.0.1:1 <--- 203.0.113.1:2  
Int-VTag = 1234

8.3. Multihomed Client and Server

The client (Host A) sends a packet containing an INIT chunk to the server (Host B), but does not include the second address.



NAT 1					
	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr

INIT[Initiate-Tag = 1234]  
10.0.0.1:1 -----> 203.0.113.1:2  
Rem-VTag = 0

NAT function 1 creates entry:

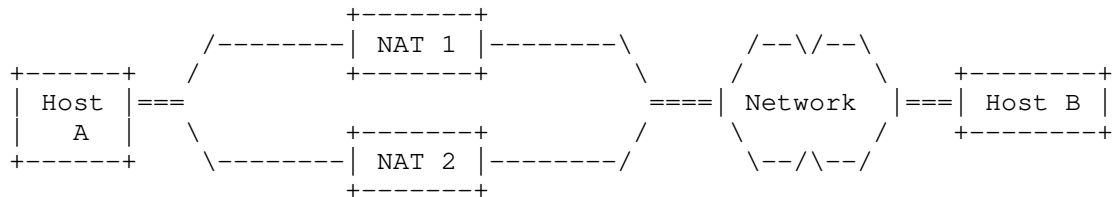
NAT 1	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	0	2	10.0.0.1

```

                                INIT[Initiate-Tag = 1234]
192.0.2.1:1 -----> 203.0.113.1:2
                                Rem-VTag = 0

```

Host B includes its second address in the INIT ACK.



```

INIT ACK[Initiate-Tag = 5678, IP-Addr = 203.0.113.129]
192.0.2.1:1 <----- 203.0.113.1:2
                                Int-VTag = 1234

```

NAT function 1 does not need to update the NAT binding table for the second address:

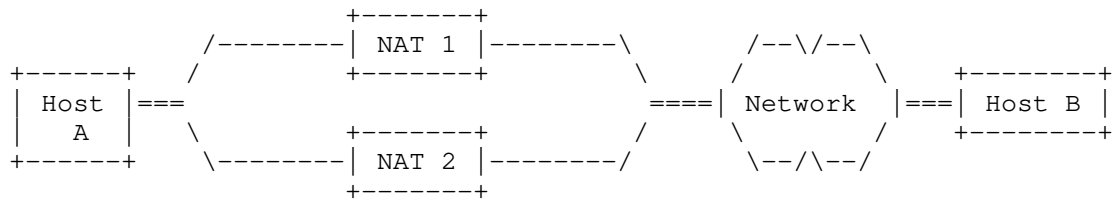
NAT 1	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	5678	2	10.0.0.1

```

INIT ACK[Initiate-Tag = 5678]
10.0.0.1:1 <----- 203.0.113.1:2
                                Int-VTag = 1234

```

The handshake finishes with a COOKIE ECHO acknowledged by a COOKIE ACK.



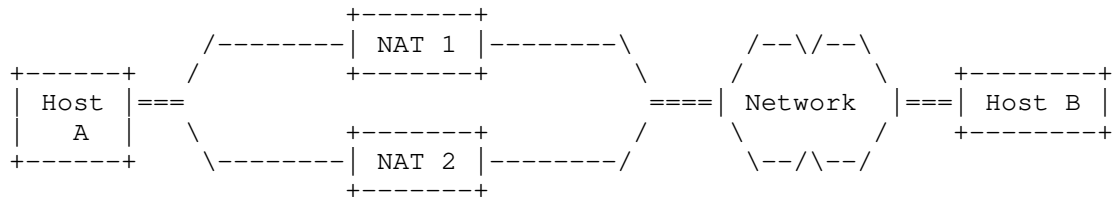
COOKIE ECHO  
 10.0.0.1:1 -----> 203.0.113.1:2  
 Rem-VTag = 5678

COOKIE ECHO  
 192.0.2.1:1 -----> 203.0.113.1:2  
 Rem-VTag = 5678

COOKIE ACK  
 192.0.2.1:1 <----- 203.0.113.1:2  
 Int-VTag = 1234

COOKIE ACK  
 10.0.0.1:1 <----- 203.0.113.1:2  
 Int-VTag = 1234

Host A announces its second address in an ASCONF chunk. The address parameter contains a wildcard address (0.0.0.0 or ::0) to indicate that the source address has to be added. The address parameter within the ASCONF chunk will also contain the pair of VTags (remote and internal) so that the NAT function can populate its NAT binding table entry completely with this single packet.



ASCONF [ADD-IP=0.0.0.0, INT-VTag=1234, Rem-VTag = 5678]  
 10.1.0.1:1 -----> 203.0.113.129:2  
 Rem-VTag = 5678

NAT function 2 creates a complete entry:

NAT 2	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	5678	2	10.1.0.1

```

ASCONF [ADD-IP, Int-VTag=1234, Rem-VTag = 5678]
192.0.2.129:1 -----> 203.0.113.129:2
                        Rem-VTag = 5678

```

```

                        ASCONF ACK
192.0.2.129:1 <----- 203.0.113.129:2
                        Int-VTag = 1234

```

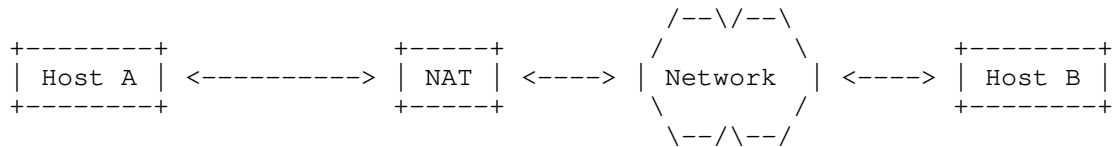
```

                        ASCONF ACK
10.1.0.1:1 <----- 203.0.113.129:2
                        Int-VTag = 1234

```

#### 8.4. NAT Function Loses Its State

Association is already established between Host A and Host B, when the NAT function loses its state and obtains a new external address. Host A sends a DATA chunk to Host B.



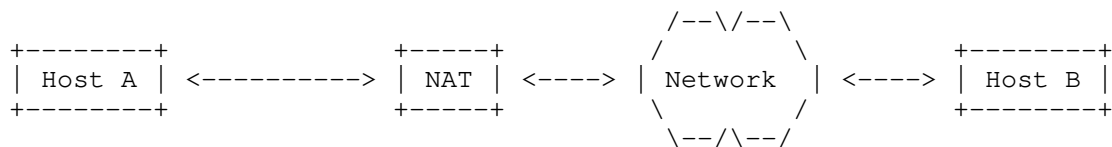
NAT	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr

```

                        DATA
10.0.0.1:1 -----> 203.0.113.1:2
                        Rem-VTag = 5678

```

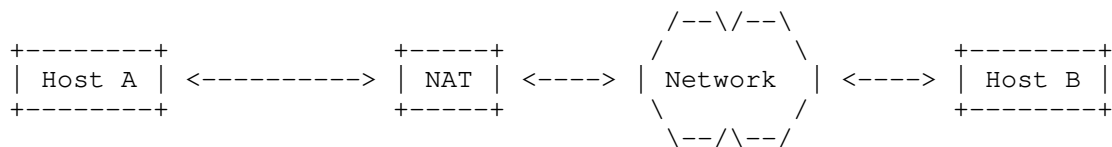
The NAT function cannot find an entry in the NAT binding table for the association. It sends a packet containing an ERROR chunk with the M bit set and the cause "NAT state missing".



```

ERROR [M bit, NAT state missing]
10.0.0.1:1 <----- 203.0.113.1:2
      Rem-VTag = 5678
  
```

On reception of the packet containing the ERROR chunk, Host A sends a packet containing an ASCONF chunk indicating that the former information has to be deleted and the source address of the actual packet added.



```

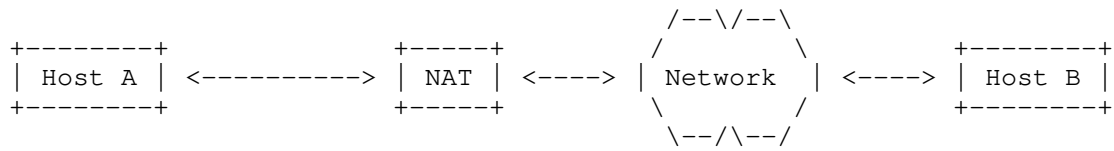
ASCONF [ADD-IP, DELETE-IP, Int-VTag=1234, Rem-VTag = 5678]
10.0.0.1:1 -----> 203.0.113.129:2
      Rem-VTag = 5678
  
```

NAT	+-----+ +-----+ +-----+ +-----+ +-----+				
	Int	Int	Rem	Rem	Int
	VTag	Port	VTag	Port	Addr
	+-----+ +-----+ +-----+ +-----+ +-----+				
	1234	1	5678	2	10.0.0.1
	+-----+ +-----+ +-----+ +-----+ +-----+				

```

ASCONF [ADD-IP, DELETE-IP, Int-VTag=1234, Rem-VTag = 5678]
      192.0.2.2:1 -----> 203.0.113.129:2
      Rem-VTag = 5678
  
```

Host B adds the new source address to this association and deletes all other addresses from this association.



```

                                ASCONF ACK
192.0.2.2:1 <----- 203.0.113.129:2
                        Int-VTag = 1234
  
```

```

                                ASCONF ACK
10.1.0.1:1 <----- 203.0.113.129:2
                        Int-VTag = 1234
  
```

```

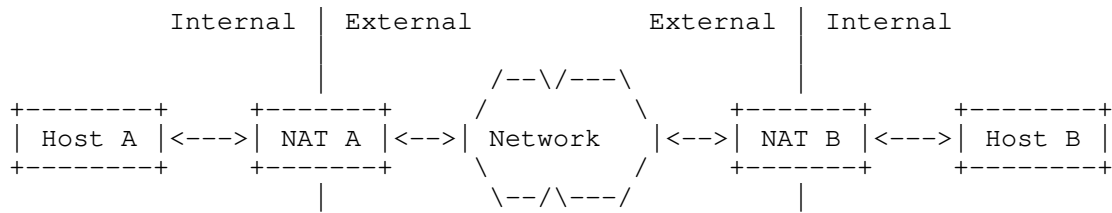
                                DATA
10.0.0.1:1 -----> 203.0.113.1:2
                        Rem-VTag = 5678
  
```

```

                                DATA
192.0.2.2:1 -----> 203.0.113.129:2
                        Rem-VTag = 5678
  
```

#### 8.5. Peer-to-Peer Communications

If two hosts, each of them behind a NAT function, want to communicate with each other, they have to get knowledge of the peer's external address. This can be achieved with a so-called rendezvous server. Afterwards the destination addresses are external, and the association is set up with the help of the INIT collision. The NAT functions create their entries according to their internal peer's point of view. Therefore, NAT function A's Internal-VTag and Internal-Port are NAT function B's Remote-VTag and Remote-Port, respectively. The naming (internal/remote) of the verification tag in the packet flow is done from the sending host's point of view.



## NAT Binding Tables

NAT A	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
-------	-------------	-------------	-------------	-------------	-------------

NAT B	Int v-tag	Int port	Rem v-tag	Rem port	Int Addr
-------	--------------	-------------	--------------	-------------	-------------

```

INIT[Initiate-Tag = 1234]
10.0.0.1:1 --> 203.0.113.1:2
    Rem-VTag = 0
  
```

NAT function A creates entry:

NAT A	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	0	2	10.0.0.1

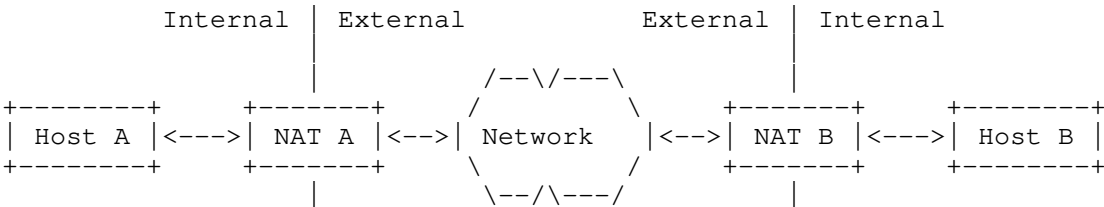
```

INIT[Initiate-Tag = 1234]
192.0.2.1:1 -----> 203.0.113.1:2
    Rem-VTag = 0
  
```

NAT function B processes the packet containing the INIT chunk, but cannot find an entry. The SCTP packet is silently discarded and leaves the NAT binding table of NAT function B unchanged.

NAT B	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
-------	-------------	-------------	-------------	-------------	-------------

Now Host B sends a packet containing an INIT chunk, which is processed by NAT function B. Its parameters are used to create an entry.



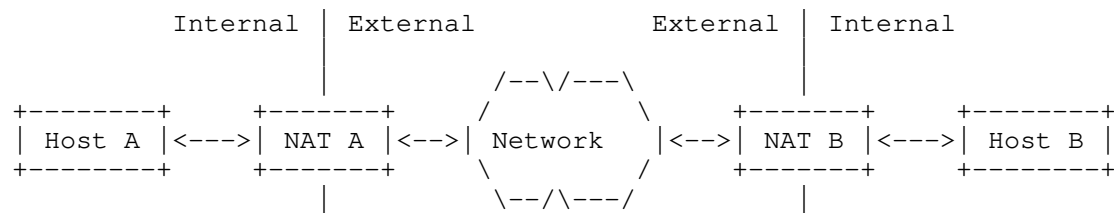
```
INIT[Initiate-Tag = 5678]
192.0.2.1:1 <-- 10.1.0.1:2
Rem-VTag = 0
```

NAT B	Int	Int	Rem	Rem	Int
	VTag	Port	VTag	Port	Addr
	5678	2	0	1	10.1.0.1

```
INIT[Initiate-Tag = 5678]
192.0.2.1:1 <----- 203.0.113.1:2
Rem-VTag = 0
```

NAT function A processes the packet containing the INIT chunk. As the outgoing packet containing an INIT chunk of Host A has already created an entry, the entry is found and updated:



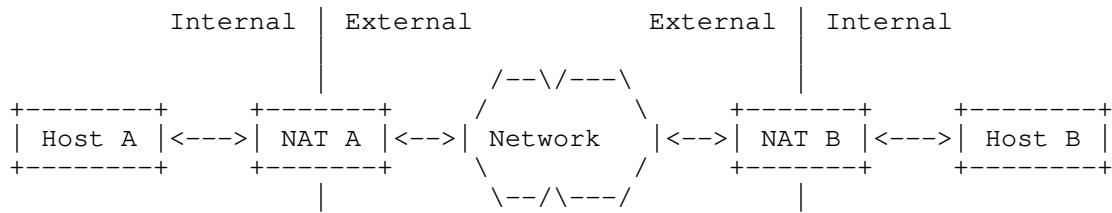


VTag != Int-VTag, but Rem-VTag == 0, find entry.

NAT A	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	5678	2	10.0.0.1

```
INIT[Initiate-tag = 5678]
10.0.0.1:1 <-- 203.0.113.1:2
    Rem-VTag = 0
```

Host A sends a packet containing an INIT ACK chunk, which can pass through NAT function B:



```

INIT ACK[Initiate-Tag = 1234]
10.0.0.1:1 --> 203.0.113.1:2
    Rem-VTag = 5678
  
```

```

          INIT ACK[Initiate-Tag = 1234]
192.0.2.1:1 -----> 203.0.113.1:2
          Rem-VTag = 5678
  
```

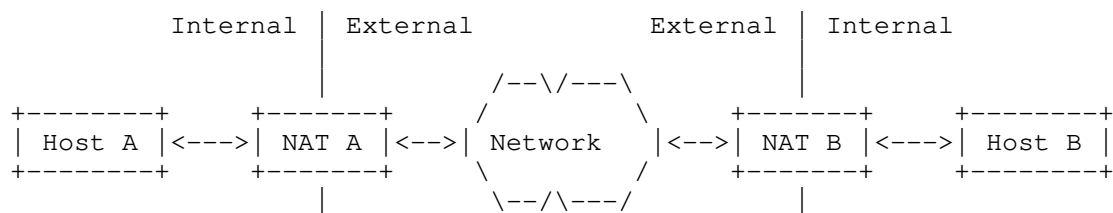
NAT function B updates entry:

NAT B	Int	Int	Rem	Rem	Int
	VTag	Port	VTag	Port	Addr
	5678	2	1234	1	10.1.0.1

```

INIT ACK[Initiate-Tag = 1234]
192.0.2.1:1 --> 10.1.0.1:2
    Rem-VTag = 5678
  
```

The lookup for COOKIE ECHO and COOKIE ACK is successful.



COOKIE ECHO  
 192.0.2.1:1 <-- 10.1.0.1:2  
 Rem-VTag = 1234

COOKIE ECHO  
 192.0.2.1:1 <----- 203.0.113.1:2  
 Rem-VTag = 1234

COOKIE ECHO  
 10.0.0.1:1 <-- 203.0.113.1:2  
 Rem-VTag = 1234

COOKIE ACK  
 10.0.0.1:1 --> 203.0.113.1:2  
 Rem-VTag = 5678

COOKIE ACK  
 192.0.2.1:1 -----> 203.0.113.1:2  
 Rem-VTag = 5678

COOKIE ACK  
 192.0.2.1:1 --> 10.1.0.1:2  
 Rem-VTag = 5678

## 9. Socket API Considerations

This section describes how the socket API defined in [RFC6458] is extended to provide a way for the application to control NAT friendliness.

Please note that this section is informational only.

A socket API implementation based on [RFC6458] is extended by supporting one new read/write socket option.

### 9.1. Get or Set the NAT Friendliness (SCTP\_NAT\_FRIENDLY)

This socket option uses the option\_level IPPROTO\_SCTP and the option\_name SCTP\_NAT\_FRIENDLY. It can be used to enable/disable the NAT friendliness for future associations and retrieve the value for future and specific ones.

```
struct sctp_assoc_value {  
    sctp_assoc_t assoc_id;  
    uint32_t assoc_value;  
};
```

assoc\_id

This parameter is ignored for one-to-one style sockets. For one-to-many style sockets the application can fill in an association identifier or SCTP\_FUTURE\_ASSOC for this query. It is an error to use SCTP\_{CURRENT|ALL}\_ASSOC in assoc\_id.

assoc\_value

A non-zero value indicates a NAT-friendly mode.

## 10. IANA Considerations

[NOTE to RFC-Editor: "RFCXXXX" is to be replaced by the RFC number you assign this document.]

[NOTE to RFC-Editor: The requested values for the chunk type and the chunk parameter types are tentative and to be confirmed by IANA.]

This document (RFCXXXX) is the reference for all registrations described in this section. The requested changes are described below.

### 10.1. New Chunk Flags for Two Existing Chunk Types

As defined in [RFC6096] two chunk flags have to be assigned by IANA for the ERROR chunk. The requested value for the T bit is 0x01 and for the M bit is 0x02.

This requires an update of the "ERROR Chunk Flags" registry for SCTP:

ERROR Chunk Flags

Chunk Flag Value	Chunk Flag Name	Reference
0x01	T bit	[RFCXXXX]
0x02	M bit	[RFCXXXX]
0x04	Unassigned	
0x08	Unassigned	
0x10	Unassigned	
0x20	Unassigned	
0x40	Unassigned	
0x80	Unassigned	

Table 2

As defined in [RFC6096] one chunk flag has to be assigned by IANA for the ABORT chunk. The requested value of the M bit is 0x02.

This requires an update of the "ABORT Chunk Flags" registry for SCTP:

ABORT Chunk Flags

Chunk Flag Value	Chunk Flag Name	Reference
0x01	T bit	[RFC4960]
0x02	M bit	[RFCXXXX]
0x04	Unassigned	
0x08	Unassigned	
0x10	Unassigned	
0x20	Unassigned	
0x40	Unassigned	
0x80	Unassigned	

Table 3

#### 10.2. Three New Error Causes

Three error causes have to be assigned by IANA. It is requested to use the values given below.

This requires three additional lines in the "Error Cause Codes" registry for SCTP:

##### Error Cause Codes

Value	Cause Code	Reference
176	VTag and Port Number Collision	[RFCXXXX]
177	Missing State	[RFCXXXX]
178	Port Number Collision	[RFCXXXX]

Table 4

### 10.3. Two New Chunk Parameter Types

Two chunk parameter types have to be assigned by IANA. IANA is requested to assign these values from the pool of parameters with the upper two bits set to '11' and to use the values given below.

This requires two additional lines in the "Chunk Parameter Types" registry for SCTP:

#### Chunk Parameter Types

ID Value	Chunk Parameter Type	Reference
49159	Disable Restart (0xC007)	[RFCXXXX]
49160	VTags (0xC008)	[RFCXXXX]

Table 5

### 10.4. One New URI

An URI in the "ns" subregistry within the "IETF XML" registry has to be assigned by IANA ([RFC3688]):

URI: urn:ietf:params:xml:ns:yang:ietf-nat-sctp  
 Registrant Contact: The IESG.  
 XML: N/A; the requested URI is an XML namespace.

### 10.5. One New YANG Module

An YANG module in the "YANG Module Names" subregistry within the "YANG Parameters" registry has to be assigned by IANA ([RFC6020]):

Name: ietf-nat-sctp  
 Namespace: urn:ietf:params:xml:ns:yang:ietf-nat-sctp  
 Maintained by IANA: N  
 Prefix: nat-sctp  
 Reference: RFCXXXX

## 11. Security Considerations

State maintenance within a NAT function is always a subject of possible Denial Of Service attacks. This document recommends that at a minimum a NAT function runs a timer on any SCTP state so that old association state can be cleaned up.

Generic issues related to address sharing are discussed in [RFC6269] and apply to SCTP as well.

For SCTP endpoints not disabling the restart procedure, this document does not add any additional security considerations to the ones given in [RFC4960], [RFC4895], and [RFC5061].

SCTP endpoints disabling the restart procedure, need to monitor the status of all associations to mitigate resource exhaustion attacks by establishing a lot of associations sharing the same IP addresses and port numbers.

In any case, SCTP is protected by the verification tags and the usage of [RFC4895] against off-path attackers.

For IP-level fragmentation and reassembly related issues see [RFC4963].

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

All data nodes defined in the YANG module that can be created, modified, and deleted (i.e., config true, which is the default) are considered sensitive. Write operations (e.g., edit-config) applied to these data nodes without proper protection can negatively affect network operations. An attacker who is able to access the SCTP NAT function can undertake various attacks, such as:

- \* Setting a low timeout for SCTP mapping entries to cause failures to deliver incoming SCTP packets.
- \* Instantiating mapping entries to cause NAT collision.

## 12. Normative References



- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC4895] Tuexen, M., Stewart, R., Lei, P., and E. Rescorla, "Authenticated Chunks for the Stream Control Transmission Protocol (SCTP)", RFC 4895, DOI 10.17487/RFC4895, August 2007, <<https://www.rfc-editor.org/info/rfc4895>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.
- [RFC5061] Stewart, R., Xie, Q., Tuexen, M., Maruyama, S., and M. Kozuka, "Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration", RFC 5061, DOI 10.17487/RFC5061, September 2007, <<https://www.rfc-editor.org/info/rfc5061>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6096] Tuexen, M. and R. Stewart, "Stream Control Transmission Protocol (SCTP) Chunk Flags Registration", RFC 6096, DOI 10.17487/RFC6096, January 2011, <<https://www.rfc-editor.org/info/rfc6096>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8512] Boucadair, M., Ed., Sivakumar, S., Jacquenet, C., Vinapamula, S., and Q. Wu, "A YANG Module for Network Address Translation (NAT) and Network Prefix Translation (NPT)", RFC 8512, DOI 10.17487/RFC8512, January 2019, <<https://www.rfc-editor.org/info/rfc8512>>.

### 13. Informative References

- [DOI\_10.1145\_1496091.1496095]  
Hayes, D., But, J., and G. Armitage, "Issues with network address translation for SCTP", ACM SIGCOMM Computer Communication Review Vol. 39, pp. 23-33, DOI 10.1145/1496091.1496095, December 2008, <<https://doi.org/10.1145/1496091.1496095>>.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC3022] Srisuresh, P. and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)", RFC 3022, DOI 10.17487/RFC3022, January 2001, <<https://www.rfc-editor.org/info/rfc3022>>.
- [RFC4787] Audet, F., Ed. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, DOI 10.17487/RFC4787, January 2007, <<https://www.rfc-editor.org/info/rfc4787>>.
- [RFC4963] Heffner, J., Mathis, M., and B. Chandler, "IPv4 Reassembly Errors at High Data Rates", RFC 4963, DOI 10.17487/RFC4963, July 2007, <<https://www.rfc-editor.org/info/rfc4963>>.
- [RFC5382] Guha, S., Ed., Biswas, K., Ford, B., Sivakumar, S., and P. Srisuresh, "NAT Behavioral Requirements for TCP", BCP 142, RFC 5382, DOI 10.17487/RFC5382, October 2008, <<https://www.rfc-editor.org/info/rfc5382>>.
- [RFC5508] Srisuresh, P., Ford, B., Sivakumar, S., and S. Guha, "NAT Behavioral Requirements for ICMP", BCP 148, RFC 5508, DOI 10.17487/RFC5508, April 2009, <<https://www.rfc-editor.org/info/rfc5508>>.
- [RFC6056] Larsen, M. and F. Gont, "Recommendations for Transport-Protocol Port Randomization", BCP 156, RFC 6056, DOI 10.17487/RFC6056, January 2011, <<https://www.rfc-editor.org/info/rfc6056>>.
- [RFC6146] Bagnulo, M., Matthews, P., and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", RFC 6146, DOI 10.17487/RFC6146, April 2011, <<https://www.rfc-editor.org/info/rfc6146>>.

- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC6269] Ford, M., Ed., Boucadair, M., Durand, A., Levis, P., and P. Roberts, "Issues with IP Address Sharing", RFC 6269, DOI 10.17487/RFC6269, June 2011, <<https://www.rfc-editor.org/info/rfc6269>>.
- [RFC6333] Durand, A., Droms, R., Woodyatt, J., and Y. Lee, "Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion", RFC 6333, DOI 10.17487/RFC6333, August 2011, <<https://www.rfc-editor.org/info/rfc6333>>.
- [RFC6458] Stewart, R., Tuexen, M., Poon, K., Lei, P., and V. Yasevich, "Sockets API Extensions for the Stream Control Transmission Protocol (SCTP)", RFC 6458, DOI 10.17487/RFC6458, December 2011, <<https://www.rfc-editor.org/info/rfc6458>>.
- [RFC6890] Cotton, M., Vegoda, L., Bonica, R., Ed., and B. Haberman, "Special-Purpose IP Address Registries", BCP 153, RFC 6890, DOI 10.17487/RFC6890, April 2013, <<https://www.rfc-editor.org/info/rfc6890>>.
- [RFC6951] Tuexen, M. and R. Stewart, "UDP Encapsulation of Stream Control Transmission Protocol (SCTP) Packets for End-Host to End-Host Communication", RFC 6951, DOI 10.17487/RFC6951, May 2013, <<https://www.rfc-editor.org/info/rfc6951>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC7857] Penno, R., Perreault, S., Boucadair, M., Ed., Sivakumar, S., and K. Naito, "Updates to Network Address Translation (NAT) Behavioral Requirements", BCP 127, RFC 7857, DOI 10.17487/RFC7857, April 2016, <<https://www.rfc-editor.org/info/rfc7857>>.

- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [RFC8900] Bonica, R., Baker, F., Huston, G., Hinden, R., Troan, O., and F. Gont, "IP Fragmentation Considered Fragile", BCP 230, RFC 8900, DOI 10.17487/RFC8900, September 2020, <<https://www.rfc-editor.org/info/rfc8900>>.

#### Acknowledgments

The authors wish to thank Mohamed Boucadair, Gorrry Fairhurst, Bryan Ford, David Hayes, Alfred Hines, Karen E. E. Nielsen, Henning Peters, Maksim Proshin, Timo Völker, Dan Wing, and Qiaobing Xie for their invaluable comments.

In addition, the authors wish to thank David Hayes, Jason But, and Grenville Armitage, the authors of [DOI\_10.1145\_1496091.1496095], for their suggestions.

The authors also wish to thank Mohamed Boucadair for contributing the text related to the YANG module.

#### Authors' Addresses

Randall R. Stewart  
Netflix, Inc.  
Chapin, SC 29036  
United States of America

Email: [randall@lakerest.net](mailto:randall@lakerest.net)

Michael Tüxen  
Münster University of Applied Sciences  
Stegerwaldstrasse 39  
48565 Steinfurt  
Germany

Email: [tuexen@fh-muenster.de](mailto:tuexen@fh-muenster.de)

Irene Rüngeler  
Münster University of Applied Sciences  
Stegerwaldstrasse 39  
48565 Steinfurt  
Germany

Email: [i.ruengeler@fh-muenster.de](mailto:i.ruengeler@fh-muenster.de)

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: April 25, 2019

R. Stewart  
Netflix, Inc.  
M. Tuexen  
Muenster Univ. of Appl. Sciences  
M. Proshin  
Ericsson  
October 22, 2018

RFC 4960 Errata and Issues  
draft-ietf-tsvwg-rfc4960-errata-08.txt

## Abstract

This document is a compilation of issues found since the publication of RFC4960 in September 2007 based on experience with implementing, testing, and using SCTP along with the suggested fixes. This document provides deltas to RFC4960 and is organized in a time ordered way. The issues are listed in the order they were brought up. Because some text is changed several times the last delta in the text is the one which should be applied. In addition to the delta a description of the problem and the details of the solution are also provided.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 25, 2019.

## Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Conventions . . . . .	4
3. Corrections to RFC 4960 . . . . .	4
3.1. Path Error Counter Threshold Handling . . . . .	4
3.2. Upper Layer Protocol Shutdown Request Handling . . . . .	5
3.3. Registration of New Chunk Types . . . . .	6
3.4. Variable Parameters for INIT Chunks . . . . .	7
3.5. CRC32c Sample Code on 64-bit Platforms . . . . .	8
3.6. Endpoint Failure Detection . . . . .	9
3.7. Data Transmission Rules . . . . .	10
3.8. T1-Cookie Timer . . . . .	11
3.9. Miscellaneous Typos . . . . .	12
3.10. CRC32c Sample Code . . . . .	19
3.11. partial_bytes_acked after T3-rtx Expiration . . . . .	20
3.12. Order of Adjustments of partial_bytes_acked and cwnd . . . . .	21
3.13. HEARTBEAT ACK and the association error counter . . . . .	22
3.14. Path for Fast Retransmission . . . . .	23
3.15. Transmittal in Fast Recovery . . . . .	24
3.16. Initial Value of ssthresh . . . . .	25
3.17. Automatically Confirmed Addresses . . . . .	26
3.18. Only One Packet after Retransmission Timeout . . . . .	27
3.19. INIT ACK Path for INIT in COOKIE-WAIT State . . . . .	28
3.20. Zero Window Probing and Unreachable Primary Path . . . . .	29
3.21. Normative Language in Section 10 . . . . .	30
3.22. Increase of partial_bytes_acked in Congestion Avoidance . . . . .	33
3.23. Inconsistency in Notifications Handling . . . . .	34
3.24. SACK.Delay Not Listed as a Protocol Parameter . . . . .	40
3.25. Processing of Chunks in an Incoming SCTP Packet . . . . .	42
3.26. CWND Increase in Congestion Avoidance Phase . . . . .	43
3.27. Refresh of cwnd and ssthresh after Idle Period . . . . .	46
3.28. Window Updates After Receiver Window Opens Up . . . . .	47
3.29. Path of DATA and Reply Chunks . . . . .	48
3.30. Outstanding Data, Flightsize and Data In Flight Key Terms . . . . .	50
3.31. CWND Degradation due to Max.Burst . . . . .	52
3.32. Reduction of RTO.Initial . . . . .	53
3.33. Ordering of Bundled SACK and ERROR Chunks . . . . .	55
3.34. Undefined Parameter Returned by RECEIVE Primitive . . . . .	56
3.35. DSCP Changes . . . . .	57

3.36. Inconsistent Handling of ICMPv4 and ICMPv6 Messages . . .	58
3.37. Handling of Soft Errors . . . . .	60
3.38. Honoring CWND . . . . .	60
3.39. Zero Window Probing . . . . .	62
3.40. Updating References Regarding ECN . . . . .	64
3.41. Host Name Address Parameter Deprecated . . . . .	66
3.42. Conflicting Text Regarding the Supported Address Types Parameter . . . . .	70
3.43. Integration of RFC 6096 . . . . .	71
3.44. Integration of RFC 6335 . . . . .	73
3.45. Integration of RFC 7053 . . . . .	75
3.46. CRC32c Code Improvements . . . . .	79
3.47. Clarification of Gap Ack Blocks in SACK Chunks . . . . .	89
3.48. Handling of SSN Wrap Arounds . . . . .	91
3.49. Update RFC 2119 Boilerplate . . . . .	92
3.50. Missed Text Removal . . . . .	93
4. IANA Considerations . . . . .	94
5. Security Considerations . . . . .	94
6. Acknowledgments . . . . .	94
7. References . . . . .	95
7.1. Normative References . . . . .	95
7.2. Informative References . . . . .	95
Authors' Addresses . . . . .	96

## 1. Introduction

This document contains a compilation of all defects found up until the publication of this document for [RFC4960] specifying the Stream Control Transmission Protocol (SCTP). These defects may be of an editorial or technical nature. This document may be thought of as a companion document to be used in the implementation of SCTP to clarify errors in the original SCTP document.

This document provides a history of the changes that will be compiled into a BIS document for [RFC4960]. It is structured similar to [RFC4460].

Each error will be detailed within this document in the form of:

- o The problem description,
- o The text quoted from [RFC4960],
- o The replacement text that should be placed into an upcoming BIS document,
- o A description of the solution.

Note that when reading this document one must use care to assure that a field or item is not updated further on within the document. Since this document is a historical record of the sequential changes that



have been found necessary at various inter-op events and through discussion on the list, the last delta in the text is the one which should be applied.

## 2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 3. Corrections to RFC 4960

[NOTE to RFC-Editor:

References to obsoleted RFCs are in OLD TEXT sections and have the corresponding references to the obsoleting RFCs in the NEW TEXT sections. In addition to this, there are some references to the obsoleted [RFC2960], which are intended.

]

### 3.1. Path Error Counter Threshold Handling

#### 3.1.1. Description of the Problem

The handling of the 'Path.Max.Retrans' parameter is described in Section 8.2 and Section 8.3 of [RFC4960] in an inconsistent way. Whereas Section 8.2 describes that a path is marked inactive when the path error counter exceeds the threshold, Section 8.3 says the path is marked inactive when the path error counter reaches the threshold.

This issue was reported as an Errata for [RFC4960] with Errata ID 1440.

#### 3.1.2. Text Changes to the Document

-----  
Old text: (Section 8.3)  
-----

When the value of this counter reaches the protocol parameter 'Path.Max.Retrans', the endpoint should mark the corresponding destination address as inactive if it is not so marked, and may also optionally report to the upper layer the change of reachability of this destination address. After this, the endpoint should continue HEARTBEAT on this destination address but should stop increasing the counter.

-----  
New text: (Section 8.3)  
-----

When the value of this counter exceeds the protocol parameter 'Path.Max.Retrans', the endpoint SHOULD mark the corresponding destination address as inactive if it is not so marked, and MAY also optionally report to the upper layer the change of reachability of this destination address. After this, the endpoint SHOULD continue HEARTBEAT on this destination address but SHOULD stop increasing the counter.

This text has been modified by multiple errata. It is further updated in Section 3.23.

### 3.1.3. Solution Description

The intended state change should happen when the threshold is exceeded.

## 3.2. Upper Layer Protocol Shutdown Request Handling

### 3.2.1. Description of the Problem

Section 9.2 of [RFC4960] describes the handling of received SHUTDOWN chunks in the SHUTDOWN-RECEIVED state instead of the handling of shutdown requests from its upper layer in this state.

This issue was reported as an Errata for [RFC4960] with Errata ID 1574.

### 3.2.2. Text Changes to the Document

-----  
Old text: (Section 9.2)  
-----

Once an endpoint has reached the SHUTDOWN-RECEIVED state, it MUST NOT send a SHUTDOWN in response to a ULP request, and should discard subsequent SHUTDOWN chunks.

-----  
New text: (Section 9.2)  
-----

Once an endpoint has reached the SHUTDOWN-RECEIVED state, it MUST ignore ULP shutdown requests, but MUST continue responding to SHUTDOWN chunks from its peer.

This text is in final form, and is not further updated in this document.

### 3.2.3. Solution Description

The text never intended the SCTP endpoint to ignore SHUTDOWN chunks from its peer. If it did, the endpoints could never gracefully terminate associations in some cases.

## 3.3. Registration of New Chunk Types

### 3.3.1. Description of the Problem

Section 14.1 of [RFC4960] should deal with new chunk types, however, the text refers to parameter types.

This issue was reported as an Errata for [RFC4960] with Errata ID 2592.

### 3.3.2. Text Changes to the Document

-----  
Old text: (Section 14.1)  
-----

The assignment of new chunk parameter type codes is done through an IETF Consensus action, as defined in [RFC2434]. Documentation of the chunk parameter MUST contain the following information:

-----  
New text: (Section 14.1)  
-----

The assignment of new chunk type codes is done through an IETF Consensus action, as defined in [RFC8126]. Documentation of the chunk type MUST contain the following information:

This text has been modified by multiple errata. It is further updated in Section 3.43.

### 3.3.3. Solution Description

Refer to chunk types as intended and change reference to [RFC8126].

## 3.4. Variable Parameters for INIT Chunks

### 3.4.1. Description of the Problem

Newlines in wrong places break the layout of the table of variable parameters for the INIT chunk in Section 3.3.2 of [RFC4960].

This issue was reported as an Errata for [RFC4960] with Errata ID 3291 and Errata ID 3804.

### 3.4.2. Text Changes to the Document

-----  
 Old text: (Section 3.3.2)  
 -----

Variable Parameters	Status	Type Value
IPv4 Address (Note 1)	Optional	5
IPv6 Address (Note 1)	Optional	6
Cookie Preservative	Optional	9
Reserved for ECN Capable (Note 2)	Optional	32768 (0x8000)
Host Name Address (Note 3)	Optional	11
Supported Address Types (Note 4)	Optional	12

-----  
 New text: (Section 3.3.2)  
 -----

Variable Parameters	Status	Type Value
IPv4 Address (Note 1)	Optional	5
IPv6 Address (Note 1)	Optional	6
Cookie Preservative	Optional	9
Reserved for ECN Capable (Note 2)	Optional	32768 (0x8000)
Host Name Address (Note 3)	Optional	11
Supported Address Types (Note 4)	Optional	12

This text is in final form, and is not further updated in this document.

### 3.4.3. Solution Description

Fix the formatting of the table.

### 3.5. CRC32c Sample Code on 64-bit Platforms

#### 3.5.1. Description of the Problem

The sample code for computing the CRC32c provided in [RFC4960] assumes that a variable of type unsigned long uses 32 bits. This is not true on some 64-bit platforms (for example the ones using LP64).

This issue was reported as an Errata for [RFC4960] with Errata ID 3423.

#### 3.5.2. Text Changes to the Document

-----

Old text: (Appendix C)

-----

```
unsigned long
generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    unsigned long crc32 = ~0L;
```

-----

New text: (Appendix C)

-----

```
unsigned long
generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    unsigned long crc32 = 0xffffffffL;
```

This text has been modified by multiple errata. It is further updated in Section 3.10 and in Section 3.46.

### 3.5.3. Solution Description

Use 0xffffffffL instead of ~0L which gives the same value on platforms using 32 bits or 64 bits for variables of type unsigned long.

## 3.6. Endpoint Failure Detection

### 3.6.1. Description of the Problem

The handling of the association error counter defined in Section 8.1 of [RFC4960] can result in an association failure even if the path used for data transmission is available, but idle.

This issue was reported as an Errata for [RFC4960] with Errata ID 3788.

### 3.6.2. Text Changes to the Document

-----

Old text: (Section 8.1)

-----

An endpoint shall keep a counter on the total number of consecutive retransmissions to its peer (this includes retransmissions to all the destination transport addresses of the peer if it is multi-homed), including unacknowledged HEARTBEAT chunks.

-----

New text: (Section 8.1)

-----

An endpoint SHOULD keep a counter on the total number of consecutive retransmissions to its peer (this includes data retransmissions to all the destination transport addresses of the peer if it is multi-homed), including the number of unacknowledged HEARTBEAT chunks observed on the path which is currently used for data transfer. Unacknowledged HEARTBEAT chunks observed on paths different from the path currently used for data transfer SHOULD NOT increment the association error counter, as this could lead to association closure even if the path which is currently used for data transfer is available (but idle).

This text has been modified by multiple errata. It is further updated in Section 3.23.

### 3.6.3. Solution Description

A more refined handling for the association error counter is defined.

## 3.7. Data Transmission Rules

### 3.7.1. Description of the Problem

When integrating the changes to Section 6.1 A) of [RFC2960] as described in Section 2.15.2 of [RFC4460] some text was duplicated and became the final paragraph of Section 6.1 A) of [RFC4960].

This issue was reported as an Errata for [RFC4960] with Errata ID 4071.

### 3.7.2. Text Changes to the Document

-----  
Old text: (Section 6.1 A)  
-----

The sender MUST also have an algorithm for sending new DATA chunks to avoid silly window syndrome (SWS) as described in [RFC0813]. The algorithm can be similar to the one described in Section 4.2.3.4 of [RFC1122].

However, regardless of the value of `rwnd` (including if it is 0), the data sender can always have one DATA chunk in flight to the receiver if allowed by `cwnd` (see rule B below). This rule allows the sender to probe for a change in `rwnd` that the sender missed due to the SACK having been lost in transit from the data receiver to the data sender.

-----  
New text: (Section 6.1 A)  
-----

The sender MUST also have an algorithm for sending new DATA chunks to avoid silly window syndrome (SWS) as described in [RFC1122]. The algorithm can be similar to the one described in Section 4.2.3.4 of [RFC1122].

This text is in final form, and is not further updated in this document.

### 3.7.3. Solution Description

Last paragraph of Section 6.1 A) removed as intended in Section 2.15.2 of [RFC4460].

### 3.8. T1-Cookie Timer

#### 3.8.1. Description of the Problem

Figure 4 of [RFC4960] illustrates the SCTP association setup. However, it incorrectly shows that the `T1-init` timer is used in the `COOKIE-ECHOED` state whereas the `T1-cookie` timer should have been used instead.

This issue was reported as an Errata for [RFC4960] with Errata ID 4400.



## 3.8.2. Text Changes to the Document

-----  
 Old text: (Section 5.1.6, Figure 4)  
 -----

```

COOKIE ECHO [Cookie_Z] -----\
(Start T1-init timer)          \
(Enter COOKIE-ECHOED state)     \---> (build TCB enter ESTABLISHED
                                     state)
                                     /---- COOKIE-ACK
                                     /
(Cancel T1-init timer, <-----/
  Enter ESTABLISHED state)

```

-----  
 New text: (Section 5.1.6, Figure 4)  
 -----

```

COOKIE ECHO [Cookie_Z] -----\
(Start T1-cookie timer)         \
(Enter COOKIE-ECHOED state)      \---> (build TCB enter ESTABLISHED
                                     state)
                                     /---- COOKIE-ACK
                                     /
(Cancel T1-cookie timer, <----/
  Enter ESTABLISHED state)

```

This text has been modified by multiple errata. It is further updated in Section 3.9.

## 3.8.3. Solution Description

Change the figure such that the T1-cookie timer is used instead of the T1-init timer.

## 3.9. Miscellaneous Typos

## 3.9.1. Description of the Problem

While processing [RFC4960] some typos were not caught.

One typo was reported as an Errata for [RFC4960] with Errata ID 5003.

## 3.9.2. Text Changes to the Document

-----

Old text: (Section 1.6)

-----

Transmission Sequence Numbers wrap around when they reach  $2^{32} - 1$ . That is, the next TSN a DATA chunk MUST use after transmitting  $TSN = 2^{32} - 1$  is  $TSN = 0$ .

-----

New text: (Section 1.6)

-----

Transmission Sequence Numbers wrap around when they reach  $2^{32} - 1$ . That is, the next TSN a DATA chunk MUST use after transmitting  $TSN = 2^{32} - 1$  is  $TSN = 0$ .

This text is in final form, and is not further updated in this document.

-----

Old text: (Section 3.3.10.9)

-----

No User Data: This error cause is returned to the originator of a DATA chunk if a received DATA chunk has no user data.

-----

New text: (Section 3.3.10.9)

-----

No User Data: This error cause is returned to the originator of a DATA chunk if a received DATA chunk has no user data.

This text is in final form, and is not further updated in this document.

-----  
 Old text: (Section 6.7, Figure 9)  
 -----

```

Endpoint A                                Endpoint Z {App
sends 3 messages; strm 0} DATA [TSN=6,Strm=0,Seq=2] -----
-----> (ack delayed) (Start T3-rtx timer)

DATA [TSN=7,Strm=0,Seq=3] -----> X (lost)

DATA [TSN=8,Strm=0,Seq=4] -----> (gap detected,
                                   immediately send ack)
                                   /----- SACK [TSN Ack=6,Block=1,
                                   /          Start=2,End=2]
                                   <-----/ (remove 6 from out-queue,
and mark 7 as "1" missing report)

```

-----  
 New text: (Section 6.7, Figure 9)  
 -----

```

Endpoint A                                Endpoint Z
{App sends 3 messages; strm 0}
DATA [TSN=6,Strm=0,Seq=2] -----> (ack delayed)
(Start T3-rtx timer)

DATA [TSN=7,Strm=0,Seq=3] -----> X (lost)

DATA [TSN=8,Strm=0,Seq=4] -----> (gap detected,
                                   immediately send ack)
                                   /----- SACK [TSN Ack=6,Block=1,
                                   /          Start=2,End=2]
                                   <-----/
(remove 6 from out-queue,
and mark 7 as "1" missing report)

```

This text is in final form, and is not further updated in this document.

-----

Old text: (Section 6.10)

-----

An endpoint bundles chunks by simply including multiple chunks in one outbound SCTP packet. The total size of the resultant IP datagram, including the SCTP packet and IP headers, MUST be less than or equal to the current Path MTU.

-----

New text: (Section 6.10)

-----

An endpoint bundles chunks by simply including multiple chunks in one outbound SCTP packet. The total size of the resultant IP datagram, including the SCTP packet and IP headers, MUST be less than or equal to the current PMTU.

This text is in final form, and is not further updated in this document.

-----

Old text: (Section 10.1 O))

-----

o Receive Unacknowledged Message

Format: RECEIVE\_UNACKED(data retrieval id, buffer address, buffer size, [,stream id] [, stream sequence number] [,partial flag] [,payload protocol-id])

-----

New text: (Section 10.1 O))

-----

O) Receive Unacknowledged Message

Format: RECEIVE\_UNACKED(data retrieval id, buffer address, buffer size [,stream id] [,stream sequence number] [,partial flag] [,payload protocol-id])

This text is in final form, and is not further updated in this document.

-----  
Old text: (Section 10.1 M)  
-----

M) Set Protocol Parameters

Format: SETPROTOCOLPARAMETERS(association id,  
[,destination transport address,]  
protocol parameter list)

-----  
New text: (Section 10.1 M)  
-----

M) Set Protocol Parameters

Format: SETPROTOCOLPARAMETERS(association id,  
[destination transport address,]  
protocol parameter list)

This text is in final form, and is not further updated in this document.

-----  
Old text: (Appendix C)  
-----

ICMP2) An implementation MAY ignore all ICMPv6 messages where the type field is not "Destination Unreachable", "Parameter Problem", or "Packet Too Big".

-----  
New text: (Appendix C)  
-----

ICMP2) An implementation MAY ignore all ICMPv6 messages where the type field is not "Destination Unreachable", "Parameter Problem", or "Packet Too Big".

This text is in final form, and is not further updated in this document.

-----

Old text: (Appendix C)

-----

ICMP7) If the ICMP message is either a v6 "Packet Too Big" or a v4 "Fragmentation Needed", an implementation MAY process this information as defined for PATH MTU discovery.

-----

New text: (Appendix C)

-----

ICMP7) If the ICMP message is either a v6 "Packet Too Big" or a v4 "Fragmentation Needed", an implementation MAY process this information as defined for PMTU discovery.

This text is in final form, and is not further updated in this document.

-----

Old text: (Section 5.4)

-----

2) For the receiver of the COOKIE ECHO, the only CONFIRMED address is the one to which the INIT-ACK was sent.

-----

New text: (Section 5.4)

-----

2) For the receiver of the COOKIE ECHO, the only CONFIRMED address is the one to which the INIT ACK was sent.

This text is in final form, and is not further updated in this document.

-----  
 Old text: (Section 5.1.6, Figure 4)  
 -----

```

COOKIE ECHO [Cookie_Z] -----\
(Start T1-init timer)          \
(Enter COOKIE-ECHOED state)    \---> (build TCB enter ESTABLISHED
                                   state)
                                   /---- COOKIE-ACK
                                   /
(Cancel T1-init timer, <-----/
  Enter ESTABLISHED state)

```

-----  
 New text: (Section 5.1.6, Figure 4)  
 -----

```

COOKIE ECHO [Cookie_Z] -----\
(Start T1-cookie timer)       \
(Enter COOKIE-ECHOED state)    \---> (build TCB enter ESTABLISHED
                                   state)
                                   /---- COOKIE ACK
                                   /
(Cancel T1-cookie timer, <---/
  Enter ESTABLISHED state)

```

This text has been modified by multiple errata. It includes modifications from Section 3.8. It is in final form, and is not further updated in this document.

-----  
 Old text: (Section 5.2.5)  
 -----

5.2.5. Handle Duplicate COOKIE-ACK.

-----  
 New text: (Section 5.2.5)  
 -----

5.2.5. Handle Duplicate COOKIE ACK.

This text is in final form, and is not further updated in this document.

-----  
Old text: (Section 8.3)  
-----

By default, an SCTP endpoint SHOULD monitor the reachability of the idle destination transport address(es) of its peer by sending a HEARTBEAT chunk periodically to the destination transport address(es). HEARTBEAT sending MAY begin upon reaching the ESTABLISHED state and is discontinued after sending either SHUTDOWN or SHUTDOWN-ACK. A receiver of a HEARTBEAT MUST respond to a HEARTBEAT with a HEARTBEAT-ACK after entering the COOKIE-ECHOED state (INIT sender) or the ESTABLISHED state (INIT receiver), up until reaching the SHUTDOWN-SENT state (SHUTDOWN sender) or the SHUTDOWN-ACK-SENT state (SHUTDOWN receiver).

-----  
New text: (Section 8.3)  
-----

By default, an SCTP endpoint SHOULD monitor the reachability of the idle destination transport address(es) of its peer by sending a HEARTBEAT chunk periodically to the destination transport address(es). HEARTBEAT sending MAY begin upon reaching the ESTABLISHED state and is discontinued after sending either SHUTDOWN or SHUTDOWN ACK. A receiver of a HEARTBEAT MUST respond to a HEARTBEAT with a HEARTBEAT ACK after entering the COOKIE-ECHOED state (INIT sender) or the ESTABLISHED state (INIT receiver), up until reaching the SHUTDOWN-SENT state (SHUTDOWN sender) or the SHUTDOWN-ACK-SENT state (SHUTDOWN receiver).

This text is in final form, and is not further updated in this document.

### 3.9.3. Solution Description

Typos fixed.

### 3.10. CRC32c Sample Code

#### 3.10.1. Description of the Problem

The CRC32c computation is described in Appendix B of [RFC4960]. However, the corresponding sample code and its explanation appears at the end of Appendix C, which deals with ICMP handling.



### 3.10.2. Text Changes to the Document

Move all of Appendix C starting with the following sentence to the end of Appendix B.

The following non-normative sample code is taken from an open-source CRC generator [WILLIAMS93], using the "mirroring" technique and yielding a lookup table for SCTP CRC32c with 256 entries, each 32 bits wide.

This text has been modified by multiple errata. It includes modifications from Section 3.5. It is further updated in Section 3.46.

### 3.10.3. Solution Description

Text moved to the appropriate location.

## 3.11. partial\_bytes\_acked after T3-rtx Expiration

### 3.11.1. Description of the Problem

Section 7.2.3 of [RFC4960] explicitly states that partial\_bytes\_acked should be reset to 0 after packet loss detection from SACK but the same is missed for T3-rtx timer expiration.

### 3.11.2. Text Changes to the Document

-----  
Old text: (Section 7.2.3)  
-----

When the T3-rtx timer expires on an address, SCTP should perform slow start by:

```
ssthresh = max(cwnd/2, 4*MTU)
cwnd = 1*MTU
```

-----  
New text: (Section 7.2.3)  
-----

When the T3-rtx timer expires on an address, SCTP SHOULD perform slow start by:

```
ssthresh = max(cwnd/2, 4*MTU)
cwnd = 1*MTU
partial_bytes_acked = 0
```

This text is in final form, and is not further updated in this document.

### 3.11.3. Solution Description

Specify that `partial_bytes_acked` should be reset to 0 after `T3-rtx` timer expiration.

### 3.12. Order of Adjustments of `partial_bytes_acked` and `cwnd`

#### 3.12.1. Description of the Problem

Section 7.2.2 of [RFC4960] likely implies the wrong order of adjustments applied to `partial_bytes_acked` and `cwnd` in the congestion avoidance phase.

#### 3.12.2. Text Changes to the Document

-----

Old text: (Section 7.2.2)

-----

- o When `partial_bytes_acked` is equal to or greater than `cwnd` and before the arrival of the SACK the sender had `cwnd` or more bytes of data outstanding (i.e., before arrival of the SACK, `flightsize` was greater than or equal to `cwnd`), increase `cwnd` by MTU, and reset `partial_bytes_acked` to `(partial_bytes_acked - cwnd)`.

-----

New text: (Section 7.2.2)

-----

- o When `partial_bytes_acked` is equal to or greater than `cwnd` and before the arrival of the SACK the sender had `cwnd` or more bytes of data outstanding (i.e., before arrival of the SACK, `flightsize` was greater than or equal to `cwnd`), `partial_bytes_acked` is reset to `(partial_bytes_acked - cwnd)`. Next, `cwnd` is increased by `1*MTU`.

This text has been modified by multiple errata. It is further updated in Section 3.26.

#### 3.12.3. Solution Description

The new text defines the exact order of adjustments of `partial_bytes_acked` and `cwnd` in the congestion avoidance phase.

### 3.13. HEARTBEAT ACK and the association error counter

#### 3.13.1. Description of the Problem

Section 8.1 and Section 8.3 of [RFC4960] prescribe that the receiver of a HEARTBEAT ACK must reset the association overall error counter. In some circumstances, e.g. when a router discards DATA chunks but not HEARTBEAT chunks due to the larger size of the DATA chunk, it might be better to not clear the association error counter on reception of the HEARTBEAT ACK and reset it only on reception of the SACK to avoid stalling the association.

#### 3.13.2. Text Changes to the Document

-----  
Old text: (Section 8.1)  
-----

The counter shall be reset each time a DATA chunk sent to that peer endpoint is acknowledged (by the reception of a SACK) or a HEARTBEAT ACK is received from the peer endpoint.

-----  
New text: (Section 8.1)  
-----

The counter MUST be reset each time a DATA chunk sent to that peer endpoint is acknowledged (by the reception of a SACK). When a HEARTBEAT ACK is received from the peer endpoint, the counter SHOULD also be reset. The receiver of the HEARTBEAT ACK MAY choose not to clear the counter if there is outstanding data on the association. This allows for handling the possible difference in reachability based on DATA chunks and HEARTBEAT chunks.

This text is in final form, and is not further updated in this document.

-----  
Old text: (Section 8.3)  
-----

Upon the receipt of the HEARTBEAT ACK, the sender of the HEARTBEAT should clear the error counter of the destination transport address to which the HEARTBEAT was sent, and mark the destination transport address as active if it is not so marked. The endpoint may optionally report to the upper layer when an inactive destination address is marked as active due to the reception of the latest HEARTBEAT ACK. The receiver of the HEARTBEAT ACK must also clear the association overall error count as well (as defined in Section 8.1).

-----  
New text: (Section 8.3)  
-----

Upon the receipt of the HEARTBEAT ACK, the sender of the HEARTBEAT MUST clear the error counter of the destination transport address to which the HEARTBEAT was sent, and mark the destination transport address as active if it is not so marked. The endpoint MAY optionally report to the upper layer when an inactive destination address is marked as active due to the reception of the latest HEARTBEAT ACK. The receiver of the HEARTBEAT ACK SHOULD also clear the association overall error counter (as defined in Section 8.1).

This text has been modified by multiple errata. It is further updated in Section 3.23.

### 3.13.3. Solution Description

The new text provides a possibility to not reset the association overall error counter when a HEARTBEAT ACK is received if there are valid reasons for it.

### 3.14. Path for Fast Retransmission

#### 3.14.1. Description of the Problem

[RFC4960] clearly describes where to retransmit data that is timed out when the peer is multi-homed but the same is not stated for fast retransmissions.

#### 3.14.2. Text Changes to the Document

-----

Old text: (Section 6.4)

-----

Furthermore, when its peer is multi-homed, an endpoint SHOULD try to retransmit a chunk that timed out to an active destination transport address that is different from the last destination address to which the DATA chunk was sent.

-----

New text: (Section 6.4)

-----

Furthermore, when its peer is multi-homed, an endpoint SHOULD try to retransmit a chunk that timed out to an active destination transport address that is different from the last destination address to which the DATA chunk was sent.

When its peer is multi-homed, an endpoint SHOULD send fast retransmissions to the same destination transport address where the original data was sent to. If the primary path has been changed and the original data was sent to the old primary path before the fast retransmit, the implementation MAY send it to the new primary path.

This text is in final form, and is not further updated in this document.

### 3.14.3. Solution Description

The new text clarifies where to send fast retransmissions.

### 3.15. Transmittal in Fast Recovery

#### 3.15.1. Description of the Problem

The Fast Retransmit on Gap Reports algorithm intends that only the very first packet may be sent regardless of cwnd in the Fast Recovery phase but rule 3) of [RFC4960], Section 7.2.4, misses this clarification.

#### 3.15.2. Text Changes to the Document

-----

Old text: (Section 7.2.4)

-----

- 3) Determine how many of the earliest (i.e., lowest TSN) DATA chunks marked for retransmission will fit into a single packet, subject to constraint of the path MTU of the destination transport address to which the packet is being sent. Call this value K. Retransmit those K DATA chunks in a single packet. When a Fast Retransmit is being performed, the sender SHOULD ignore the value of cwnd and SHOULD NOT delay retransmission for this single packet.

-----

New text: (Section 7.2.4)

-----

- 3) If not in Fast Recovery, determine how many of the earliest (i.e., lowest TSN) DATA chunks marked for retransmission will fit into a single packet, subject to constraint of the PMTU of the destination transport address to which the packet is being sent. Call this value K. Retransmit those K DATA chunks in a single packet. When a Fast Retransmit is being performed, the sender SHOULD ignore the value of cwnd and SHOULD NOT delay retransmission for this single packet.

This text is in final form, and is not further updated in this document.

### 3.15.3. Solution Description

The new text explicitly specifies to send only the first packet in the Fast Recovery phase disregarding cwnd limitations.

### 3.16. Initial Value of ssthresh

#### 3.16.1. Description of the Problem

The initial value of ssthresh should be set arbitrarily high. Using the advertised receiver window of the peer is inappropriate if the peer increases its window after the handshake. Furthermore, use a higher requirements level, since not following the advice may result in performance problems.

### 3.16.2. Text Changes to the Document

-----

Old text: (Section 7.2.1)

-----

- o The initial value of ssthresh MAY be arbitrarily high (for example, implementations MAY use the size of the receiver advertised window).

-----

New text: (Section 7.2.1)

-----

- o The initial value of ssthresh SHOULD be arbitrarily high (e.g., the size of the largest possible advertised window).

This text is in final form, and is not further updated in this document.

### 3.16.3. Solution Description

Use the same value as suggested in [RFC5681], Section 3.1, as an appropriate initial value. Furthermore, use the same requirements level.

## 3.17. Automatically Confirmed Addresses

### 3.17.1. Description of the Problem

The Path Verification procedure of [RFC4960] prescribes that any address passed to the sender of the INIT by its upper layer is automatically CONFIRMED. This, however, is unclear if only addresses in the request to initiate association establishment are considered or any addresses provided by the upper layer in any requests (e.g. in 'Set Primary').

### 3.17.2. Text Changes to the Document

-----

Old text: (Section 5.4)

-----

- 1) Any address passed to the sender of the INIT by its upper layer is automatically considered to be CONFIRMED.

-----

New text: (Section 5.4)

-----

- 1) Any addresses passed to the sender of the INIT by its upper layer in the request to initialize an association are automatically considered to be CONFIRMED.

This text is in final form, and is not further updated in this document.

### 3.17.3. Solution Description

The new text clarifies that only addresses provided by the upper layer in the request to initialize an association are automatically confirmed.

## 3.18. Only One Packet after Retransmission Timeout

### 3.18.1. Description of the Problem

[RFC4960] is not completely clear when it describes data transmission after T3-rtx timer expiration. Section 7.2.1 does not specify how many packets are allowed to be sent after T3-rtx timer expiration if more than one packet fit into cwnd. At the same time, Section 7.2.3 has the text without normative language saying that SCTP should ensure that no more than one packet will be in flight after T3-rtx timer expiration until successful acknowledgment. It makes the text inconsistent.

### 3.18.2. Text Changes to the Document



-----

Old text: (Section 7.2.1)

-----

- o The initial cwnd after a retransmission timeout MUST be no more than 1\*MTU.

-----

New text: (Section 7.2.1)

-----

- o The initial cwnd after a retransmission timeout MUST be no more than 1\*MTU and only one packet is allowed to be in flight until successful acknowledgement.

This text is in final form, and is not further updated in this document.

### 3.18.3. Solution Description

The new text clearly specifies that only one packet is allowed to be sent after T3-rtx timer expiration until successful acknowledgement.

## 3.19. INIT ACK Path for INIT in COOKIE-WAIT State

### 3.19.1. Description of the Problem

In case of an INIT received in the COOKIE-WAIT state [RFC4960] prescribes to send an INIT ACK to the same destination address to which the original INIT has been sent. This text does not address the possibility of the upper layer to provide multiple remote IP addresses while requesting the association establishment. If the upper layer has provided multiple IP addresses and only a subset of these addresses are supported by the peer then the destination address of the original INIT may be absent in the incoming INIT and sending INIT ACK to that address is useless.

### 3.19.2. Text Changes to the Document

-----  
Old text: (Section 5.2.1)  
-----

Upon receipt of an INIT in the COOKIE-WAIT state, an endpoint MUST respond with an INIT ACK using the same parameters it sent in its original INIT chunk (including its Initiate Tag, unchanged). When responding, the endpoint MUST send the INIT ACK back to the same address that the original INIT (sent by this endpoint) was sent.

-----  
New text: (Section 5.2.1)  
-----

Upon receipt of an INIT in the COOKIE-WAIT state, an endpoint MUST respond with an INIT ACK using the same parameters it sent in its original INIT chunk (including its Initiate Tag, unchanged). When responding, the following rules MUST be applied:

- 1) The INIT ACK MUST only be sent to an address passed by the upper layer in the request to initialize the association.
- 2) The INIT ACK MUST only be sent to an address reported in the incoming INIT.
- 3) The INIT ACK SHOULD be sent to the source address of the received INIT.

This text is in final form, and is not further updated in this document.

### 3.19.3. Solution Description

The new text requires sending INIT ACK to a destination address that is passed by the upper layer and reported in the incoming INIT. If the source address of the INIT meets these conditions, sending the INIT ACK to the source address of the INIT is the preferred behavior.

## 3.20. Zero Window Probing and Unreachable Primary Path

### 3.20.1. Description of the Problem

Section 6.1 of [RFC4960] states that when sending zero window probes, SCTP should neither increment the association counter nor increment the destination address error counter if it continues to receive new packets from the peer. However, the reception of new packets from the peer does not guarantee the peer's reachability and, if the destination address becomes unreachable during zero window probing,

SCTP cannot get an updated rwnd until it switches the destination address for probes.

### 3.20.2. Text Changes to the Document

-----  
Old text: (Section 6.1)  
-----

If the sender continues to receive new packets from the receiver while doing zero window probing, the unacknowledged window probes should not increment the error counter for the association or any destination transport address. This is because the receiver MAY keep its window closed for an indefinite time. Refer to Section 6.2 on the receiver behavior when it advertises a zero window.

-----  
New text: (Section 6.1)  
-----

If the sender continues to receive SACKs from the peer while doing zero window probing, the unacknowledged window probes SHOULD NOT increment the error counter for the association or any destination transport address. This is because the receiver could keep its window closed for an indefinite time. Section 6.2 describes the receiver behavior when it advertises a zero window.

This text is in final form, and is not further updated in this document.

### 3.20.3. Solution Description

The new text clarifies that if the receiver continues to send SACKs, the sender of probes should not increment the error counter of the association and the destination address even if the SACKs do not acknowledge the probes.

## 3.21. Normative Language in Section 10

### 3.21.1. Description of the Problem

Section 10 of [RFC4960] is informative and, therefore, normative language such as MUST and MAY cannot be used there. However, there are several places in Section 10 where MUST and MAY are used.

## 3.21.2. Text Changes to the Document

-----  
Old text: (Section 10.1 E))  
-----

- o no-bundle flag - instructs SCTP not to bundle this user data with other outbound DATA chunks. SCTP MAY still bundle even when this flag is present, when faced with network congestion.

-----  
New text: (Section 10.1 E))  
-----

- o no-bundle flag - instructs SCTP not to bundle this user data with other outbound DATA chunks. SCTP may still bundle even when this flag is present, when faced with network congestion.

This text is in final form, and is not further updated in this document.

-----  
Old text: (Section 10.1 G))  
-----

- o Stream Sequence Number - the Stream Sequence Number assigned by the sending SCTP peer.
- o partial flag - if this returned flag is set to 1, then this Receive contains a partial delivery of the whole message. When this flag is set, the stream id and Stream Sequence Number MUST accompany this receive. When this flag is set to 0, it indicates that no more deliveries will be received for this Stream Sequence Number.

-----  
New text: (Section 10.1 G))  
-----

- o stream sequence number - the Stream Sequence Number assigned by the sending SCTP peer.
- o partial flag - if this returned flag is set to 1, then this primitive contains a partial delivery of the whole message. When this flag is set, the stream id and stream sequence number must accompany this primitive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.

This text is in final form, and is not further updated in this document.

-----  
Old text: (Section 10.1 N)  
-----

- o Stream Sequence Number - this value is returned indicating the Stream Sequence Number that was associated with the message.
- o partial flag - if this returned flag is set to 1, then this message is a partial delivery of the whole message. When this flag is set, the stream id and Stream Sequence Number MUST accompany this receive. When this flag is set to 0, it indicates that no more deliveries will be received for this Stream Sequence Number.

-----  
New text: (Section 10.1 N)  
-----

- o stream sequence number - this value is returned indicating the Stream Sequence Number that was associated with the message.
- o partial flag - if this returned flag is set to 1, then this message is a partial delivery of the whole message. When this flag is set, the stream id and stream sequence number must accompany this primitive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.

This text is in final form, and is not further updated in this document.

-----  
Old text: (Section 10.1 O)  
-----

- o Stream Sequence Number - this value is returned indicating the Stream Sequence Number that was associated with the message.
- o partial flag - if this returned flag is set to 1, then this message is a partial delivery of the whole message. When this flag is set, the stream id and Stream Sequence Number MUST accompany this receive. When this flag is set to 0, it indicates that no more deliveries will be received for this Stream Sequence Number.

-----  
New text: (Section 10.1 O)  
-----

- o stream sequence number - this value is returned indicating the Stream Sequence Number that was associated with the message.
- o partial flag - if this returned flag is set to 1, then this message is a partial delivery of the whole message. When this flag is set, the stream id and stream sequence number must accompany this primitive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.

This text is in final form, and is not further updated in this document.

### 3.21.3. Solution Description

The normative language is removed from Section 10. In addition, the consistency of the text has been improved.

## 3.22. Increase of partial\_bytes\_acked in Congestion Avoidance

### 3.22.1. Description of the Problem

Two issues have been discovered with the partial\_bytes\_acked handling described in Section 7.2.2 of [RFC4960]:

- o If the Cumulative TSN Ack Point is not advanced but the SACK chunk acknowledges new TSNs in the Gap Ack Blocks, these newly acknowledged TSNs are not considered for partial\_bytes\_acked although these TSNs were successfully received by the peer.

- o Duplicate TSNs are not considered in `partial_bytes_acked` although they confirm that the DATA chunks were successfully received by the peer.

### 3.22.2. Text Changes to the Document

-----

Old text: (Section 7.2.2)

-----

- o Whenever `cwnd` is greater than `ssthresh`, upon each SACK arrival that advances the Cumulative TSN Ack Point, increase `partial_bytes_acked` by the total number of bytes of all new chunks acknowledged in that SACK including chunks acknowledged by the new Cumulative TSN Ack and by Gap Ack Blocks.

-----

New text: (Section 7.2.2)

-----

- o Whenever `cwnd` is greater than `ssthresh`, upon each SACK arrival, increase `partial_bytes_acked` by the total number of bytes of all new chunks acknowledged in that SACK including chunks acknowledged by the new Cumulative TSN Ack, by Gap Ack Blocks and by the number of bytes of duplicated chunks reported in Duplicate TSNs.

This text has been modified by multiple errata. It is further updated in Section 3.26.

### 3.22.3. Solution Description

Now `partial_bytes_acked` is increased by TSNs reported as duplicated as well as TSNs newly acknowledged in Gap Ack Blocks even if the Cumulative TSN Ack Point is not advanced.

## 3.23. Inconsistency in Notifications Handling

### 3.23.1. Description of the Problem

[RFC4960] uses inconsistent normative and non-normative language when describing rules for sending notifications to the upper layer. E.g. Section 8.2 of [RFC4960] says that when a destination address becomes inactive due to an unacknowledged DATA chunk or HEARTBEAT chunk, SCTP SHOULD send a notification to the upper layer while Section 8.3 of [RFC4960] says that when a destination address becomes inactive due to an unacknowledged HEARTBEAT chunk, SCTP may send a notification to the upper layer.

This makes the text inconsistent.

### 3.23.2. Text Changes to the Document

-----

Old text: (Section 8.1)

-----

An endpoint shall keep a counter on the total number of consecutive retransmissions to its peer (this includes retransmissions to all the destination transport addresses of the peer if it is multi-homed), including unacknowledged HEARTBEAT chunks.

-----

New text: (Section 8.1)

-----

An endpoint SHOULD keep a counter on the total number of consecutive retransmissions to its peer (this includes data retransmissions to all the destination transport addresses of the peer if it is multi-homed), including the number of unacknowledged HEARTBEAT chunks observed on the path which currently is used for data transfer. Unacknowledged HEARTBEAT chunks observed on paths different from the path currently used for data transfer SHOULD NOT increment the association error counter, as this could lead to association closure even if the path which currently is used for data transfer is available (but idle). If the value of this counter exceeds the limit indicated in the protocol parameter 'Association.Max.Retrans', the endpoint SHOULD consider the peer endpoint unreachable and SHALL stop transmitting any more data to it (and thus the association enters the CLOSED state). In addition, the endpoint SHOULD report the failure to the upper layer and optionally report back all outstanding user data remaining in its outbound queue. The association is automatically closed when the peer endpoint becomes unreachable.

This text has been modified by multiple errata. It includes modifications from Section 3.6. It is in final form, and is not further updated in this document.



-----  
Old text: (Section 8.2)  
-----

When an outstanding TSN is acknowledged or a HEARTBEAT sent to that address is acknowledged with a HEARTBEAT ACK, the endpoint shall clear the error counter of the destination transport address to which the DATA chunk was last sent (or HEARTBEAT was sent). When the peer endpoint is multi-homed and the last chunk sent to it was a retransmission to an alternate address, there exists an ambiguity as to whether or not the acknowledgement should be credited to the address of the last chunk sent. However, this ambiguity does not seem to bear any significant consequence to SCTP behavior. If this ambiguity is undesirable, the transmitter may choose not to clear the error counter if the last chunk sent was a retransmission.

-----  
New text: (Section 8.2)  
-----

When an outstanding TSN is acknowledged or a HEARTBEAT sent to that address is acknowledged with a HEARTBEAT ACK, the endpoint SHOULD clear the error counter of the destination transport address to which the DATA chunk was last sent (or HEARTBEAT was sent), and SHOULD also report to the upper layer when an inactive destination address is marked as active. When the peer endpoint is multi-homed and the last chunk sent to it was a retransmission to an alternate address, there exists an ambiguity as to whether or not the acknowledgement could be credited to the address of the last chunk sent. However, this ambiguity does not seem to bear any significant consequence to SCTP behavior. If this ambiguity is undesirable, the transmitter MAY choose not to clear the error counter if the last chunk sent was a retransmission.

This text is in final form, and is not further updated in this document.

-----

Old text: (Section 8.3)

-----

When the value of this counter reaches the protocol parameter 'Path.Max.Retrans', the endpoint should mark the corresponding destination address as inactive if it is not so marked, and may also optionally report to the upper layer the change of reachability of this destination address. After this, the endpoint should continue HEARTBEAT on this destination address but should stop increasing the counter.

-----

New text: (Section 8.3)

-----

When the value of this counter exceeds the protocol parameter 'Path.Max.Retrans', the endpoint SHOULD mark the corresponding destination address as inactive if it is not so marked, and SHOULD also report to the upper layer the change of reachability of this destination address. After this, the endpoint SHOULD continue HEARTBEAT on this destination address but SHOULD stop increasing the counter.

This text has been modified by multiple errata. It includes modifications from Section 3.1. It is in final form, and is not further updated in this document.

-----  
Old text: (Section 8.3)  
-----

Upon the receipt of the HEARTBEAT ACK, the sender of the HEARTBEAT should clear the error counter of the destination transport address to which the HEARTBEAT was sent, and mark the destination transport address as active if it is not so marked. The endpoint may optionally report to the upper layer when an inactive destination address is marked as active due to the reception of the latest HEARTBEAT ACK. The receiver of the HEARTBEAT ACK must also clear the association overall error count as well (as defined in Section 8.1).

-----  
New text: (Section 8.3)  
-----

Upon the receipt of the HEARTBEAT ACK, the sender of the HEARTBEAT SHOULD clear the error counter of the destination transport address to which the HEARTBEAT was sent, and mark the destination transport address as active if it is not so marked. The endpoint SHOULD report to the upper layer when an inactive destination address is marked as active due to the reception of the latest HEARTBEAT ACK. The receiver of the HEARTBEAT ACK SHOULD also clear the association overall error counter (as defined in Section 8.1).

This text has been modified by multiple errata. It includes modifications from Section 3.13. It is in final form, and is not further updated in this document.

-----  
Old text: (Section 9.2)  
-----

An endpoint should limit the number of retransmissions of the SHUTDOWN chunk to the protocol parameter 'Association.Max.Retrans'. If this threshold is exceeded, the endpoint should destroy the TCB and MUST report the peer endpoint unreachable to the upper layer (and thus the association enters the CLOSED state).

-----  
New text: (Section 9.2)  
-----

An endpoint SHOULD limit the number of retransmissions of the SHUTDOWN chunk to the protocol parameter 'Association.Max.Retrans'. If this threshold is exceeded, the endpoint SHOULD destroy the TCB and SHOULD report the peer endpoint unreachable to the upper layer (and thus the association enters the CLOSED state).

This text is in final form, and is not further updated in this document.

-----  
Old text: (Section 9.2)  
-----

The sender of the SHUTDOWN ACK should limit the number of retransmissions of the SHUTDOWN ACK chunk to the protocol parameter 'Association.Max.Retrans'. If this threshold is exceeded, the endpoint should destroy the TCB and may report the peer endpoint unreachable to the upper layer (and thus the association enters the CLOSED state).

-----  
New text: (Section 9.2)  
-----

The sender of the SHUTDOWN ACK SHOULD limit the number of retransmissions of the SHUTDOWN ACK chunk to the protocol parameter 'Association.Max.Retrans'. If this threshold is exceeded, the endpoint SHOULD destroy the TCB and SHOULD report the peer endpoint unreachable to the upper layer (and thus the association enters the CLOSED state).

This text is in final form, and is not further updated in this document.

### 3.23.3. Solution Description

The inconsistencies are removed by using consistently SHOULD.

### 3.24. SACK.Delay Not Listed as a Protocol Parameter

#### 3.24.1. Description of the Problem

SCTP as specified in [RFC4960] supports delaying SACKs. The timer value for this is a parameter and Section 6.2 of [RFC4960] specifies a default and maximum value for it. However, defining a name for this parameter and listing it in the table of protocol parameters in Section 15 of [RFC4960] is missing.

This issue was reported as an Errata for [RFC4960] with Errata ID 4656.

#### 3.24.2. Text Changes to the Document

-----

Old text: (Section 6.2)

-----

An implementation MUST NOT allow the maximum delay to be configured to be more than 500 ms. In other words, an implementation MAY lower this value below 500 ms but MUST NOT raise it above 500 ms.

-----

New text: (Section 6.2)

-----

An implementation MUST NOT allow the maximum delay (protocol parameter 'SACK.Delay') to be configured to be more than 500 ms. In other words, an implementation MAY lower the value of SACK.Delay below 500 ms but MUST NOT raise it above 500 ms.

This text is in final form, and is not further updated in this document.

-----  
Old text: (Section 15)  
-----

The following protocol parameters are RECOMMENDED:

RTO.Initial - 3 seconds  
RTO.Min - 1 second  
RTO.Max - 60 seconds  
Max.Burst - 4  
RTO.Alpha - 1/8  
RTO.Beta - 1/4  
Valid.Cookie.Life - 60 seconds  
Association.Max.Retrans - 10 attempts  
Path.Max.Retrans - 5 attempts (per destination address)  
Max.Init.Retransmits - 8 attempts  
HB.interval - 30 seconds  
HB.Max.Burst - 1

-----  
New text: (Section 15)  
-----

The following protocol parameters are RECOMMENDED:

RTO.Initial - 3 seconds  
RTO.Min - 1 second  
RTO.Max - 60 seconds  
Max.Burst - 4  
RTO.Alpha - 1/8  
RTO.Beta - 1/4  
Valid.Cookie.Life - 60 seconds  
Association.Max.Retrans - 10 attempts  
Path.Max.Retrans - 5 attempts (per destination address)  
Max.Init.Retransmits - 8 attempts  
HB.interval - 30 seconds  
HB.Max.Burst - 1  
SACK.Delay - 200 milliseconds

This text has been modified by multiple errata. It is further updated in Section 3.32.

### 3.24.3. Solution Description

The parameter was given a name and added to the list of protocol parameters.

### 3.25. Processing of Chunks in an Incoming SCTP Packet

#### 3.25.1. Description of the Problem

There are a few places in [RFC4960] where the receiver of a packet must discard it while processing the chunks of the packet. It is unclear whether the receiver has to rollback state changes already performed while processing the packet or not.

The intention of [RFC4960] is to process an incoming packet chunk by chunk and not to perform any prescreening of chunks in the received packet. Thus, by discarding one chunk the receiver also causes discarding of all further chunks.

#### 3.25.2. Text Changes to the Document

-----

Old text: (Section 3.2)

-----

- 00 - Stop processing this SCTP packet and discard it, do not process any further chunks within it.
- 01 - Stop processing this SCTP packet and discard it, do not process any further chunks within it, and report the unrecognized chunk in an 'Unrecognized Chunk Type'.

-----

New text: (Section 3.2)

-----

- 00 - Stop processing this SCTP packet, discard the unrecognized chunk and all further chunks.
- 01 - Stop processing this SCTP packet, discard the unrecognized chunk and all further chunks, and report the unrecognized chunk in an 'Unrecognized Chunk Type'.

This text is in final form, and is not further updated in this document.

-----  
Old text: (Section 11.3)  
-----

It is helpful for some firewalls if they can inspect just the first fragment of a fragmented SCTP packet and unambiguously determine whether it corresponds to an INIT chunk (for further information, please refer to [RFC1858]). Accordingly, we stress the requirements, stated in Section 3.1, that (1) an INIT chunk MUST NOT be bundled with any other chunk in a packet, and (2) a packet containing an INIT chunk MUST have a zero Verification Tag. Furthermore, we require that the receiver of an INIT chunk MUST enforce these rules by silently discarding an arriving packet with an INIT chunk that is bundled with other chunks or has a non-zero verification tag and contains an INIT-chunk.

-----  
New text: (Section 11.3)  
-----

It is helpful for some firewalls if they can inspect just the first fragment of a fragmented SCTP packet and unambiguously determine whether it corresponds to an INIT chunk (for further information, please refer to [RFC1858]). Accordingly, we stress the requirements, stated in Section 3.1, that (1) an INIT chunk MUST NOT be bundled with any other chunk in a packet, and (2) a packet containing an INIT chunk MUST have a zero Verification Tag. The receiver of an INIT chunk MUST silently discard the INIT chunk and all further chunks if the INIT chunk is bundled with other chunks or the packet has a non-zero verification tag.

This text is in final form, and is not further updated in this document.

### 3.25.3. Solution Description

The new text makes it clear that chunks can be processed from the beginning to the end and no rollback or pre-screening is required.

### 3.26. CWND Increase in Congestion Avoidance Phase

#### 3.26.1. Description of the Problem

[RFC4960] in Section 7.2.2 prescribes to increase cwnd by 1\*MTU per RTT if the sender has cwnd or more bytes of data outstanding to the corresponding address in the Congestion Avoidance phase. However, this is described without normative language. Moreover, Section 7.2.2 includes an algorithm how an implementation can achieve



this but this algorithm is underspecified and actually allows increasing cwnd by more than 1\*MTU per RTT.

### 3.26.2. Text Changes to the Document

-----

Old text: (Section 7.2.2)

-----

When cwnd is greater than ssthresh, cwnd should be incremented by 1\*MTU per RTT if the sender has cwnd or more bytes of data outstanding for the corresponding transport address.

-----

New text: (Section 7.2.2)

-----

When cwnd is greater than ssthresh, cwnd SHOULD be incremented by 1\*MTU per RTT if the sender has cwnd or more bytes of data outstanding for the corresponding transport address. The basic guidelines for incrementing cwnd during congestion avoidance are:

- o Sctp MAY increment cwnd by 1\*MTU.
- o Sctp SHOULD increment cwnd by one 1\*MTU once per RTT when the sender has cwnd or more bytes of data outstanding for the corresponding transport address.
- o Sctp MUST NOT increment cwnd by more than 1\*MTU per RTT.

This text is in final form, and is not further updated in this document.

-----

Old text: (Section 7.2.2)

-----

- o Whenever `cwnd` is greater than `ssthresh`, upon each SACK arrival that advances the Cumulative TSN Ack Point, increase `partial_bytes_acked` by the total number of bytes of all new chunks acknowledged in that SACK including chunks acknowledged by the new Cumulative TSN Ack and by Gap Ack Blocks.
- o When `partial_bytes_acked` is equal to or greater than `cwnd` and before the arrival of the SACK the sender had `cwnd` or more bytes of data outstanding (i.e., before arrival of the SACK, `flightsize` was greater than or equal to `cwnd`), increase `cwnd` by MTU, and reset `partial_bytes_acked` to `(partial_bytes_acked - cwnd)`.

-----

New text: (Section 7.2.2)

-----

- o Whenever `cwnd` is greater than `ssthresh`, upon each SACK arrival, increase `partial_bytes_acked` by the total number of bytes of all new chunks acknowledged in that SACK including chunks acknowledged by the new Cumulative TSN Ack, by Gap Ack Blocks and by the number of bytes of duplicated chunks reported in Duplicate TSNs.
- o When `partial_bytes_acked` is greater than `cwnd` and before the arrival of the SACK the sender had less than `cwnd` bytes of data outstanding (i.e., before arrival of the SACK, `flightsize` was less than `cwnd`), reset `partial_bytes_acked` to `cwnd`.
- o When `partial_bytes_acked` is equal to or greater than `cwnd` and before the arrival of the SACK the sender had `cwnd` or more bytes of data outstanding (i.e., before arrival of the SACK, `flightsize` was greater than or equal to `cwnd`), `partial_bytes_acked` is reset to `(partial_bytes_acked - cwnd)`. Next, `cwnd` is increased by `1*MTU`.

This text has been modified by multiple errata. It includes modifications from Section 3.12 and Section 3.22. It is in final form, and is not further updated in this document.

### 3.26.3. Solution Description

The basic guidelines for incrementing `cwnd` during the congestion avoidance phase are added into Section 7.2.2. The guidelines include the normative language and are aligned with [RFC5681].

The algorithm from Section 7.2.2 is improved to not allow increasing cwnd by more than 1\*MTU per RTT.

### 3.27. Refresh of cwnd and ssthresh after Idle Period

#### 3.27.1. Description of the Problem

[RFC4960] prescribes to adjust cwnd per RTO if the endpoint does not transmit data on a given transport address. In addition to that, it prescribes to set cwnd to the initial value after a sufficiently long idle period. The latter is excessive. Moreover, it is unclear what is a sufficiently long idle period.

[RFC4960] doesn't specify the handling of ssthresh in the idle case. If ssthresh is reduced due to a packet loss, ssthresh is never recovered. So traffic can end up in Congestion Avoidance all the time, resulting in a low sending rate and bad performance. The problem is even more serious for SCTP because in a multi-homed SCTP association traffic that switches back to the previously failed primary path will also lead to the situation where traffic ends up in Congestion Avoidance.

#### 3.27.2. Text Changes to the Document

-----

Old text: (Section 7.2.1)

-----

- o The initial cwnd before DATA transmission or after a sufficiently long idle period MUST be set to min(4\*MTU, max (2\*MTU, 4380 bytes)).

-----

New text: (Section 7.2.1)

-----

- o The initial cwnd before DATA transmission MUST be set to min(4\*MTU, max (2\*MTU, 4380 bytes)).

-----

Old text: (Section 7.2.1)

-----

- o When the endpoint does not transmit data on a given transport address, the cwnd of the transport address should be adjusted to  $\max(\text{cwnd}/2, 4 \cdot \text{MTU})$  per RTO.

-----

New text: (Section 7.2.1)

-----

- o While the endpoint does not transmit data on a given transport address, the cwnd of the transport address SHOULD be adjusted to  $\max(\text{cwnd}/2, 4 \cdot \text{MTU})$  once per RTO. Before the first cwnd adjustment, the ssthresh of the transport address SHOULD be set to the cwnd.

This text is in final form, and is not further updated in this document.

### 3.27.3. Solution Description

A rule about cwnd adjustment after a sufficiently long idle period is removed.

The text is updated to describe the ssthresh handling. When the idle period is detected, the cwnd value is stored to the ssthresh value.

## 3.28. Window Updates After Receiver Window Opens Up

### 3.28.1. Description of the Problem

The sending of SACK chunks for window updates is only indirectly referenced in [RFC4960], Section 6.2, where it is stated that an SCTP receiver must not generate more than one SACK for every incoming packet, other than to update the offered window.

However, the sending of window updates when the receiver window opens up is necessary to avoid performance problems.

### 3.28.2. Text Changes to the Document

-----  
Old text: (Section 6.2)  
-----

An SCTP receiver MUST NOT generate more than one SACK for every incoming packet, other than to update the offered window as the receiving application consumes new data.

-----  
New text: (Section 6.2)  
-----

An SCTP receiver MUST NOT generate more than one SACK for every incoming packet, other than to update the offered window as the receiving application consumes new data. When the window opens up, an SCTP receiver SHOULD send additional SACK chunks to update the window even if no new data is received. The receiver MUST avoid sending a large number of window updates, in particular large bursts of them. One way to achieve this is to send a window update only if the window can be increased by at least a quarter of the receive buffer size of the association.

This text is in final form, and is not further updated in this document.

### 3.28.3. Solution Description

The new text makes clear that additional SACK chunks for window updates should be sent as long as excessive bursts are avoided.

### 3.29. Path of DATA and Reply Chunks

#### 3.29.1. Description of the Problem

Section 6.4 of [RFC4960] describes the transmission policy for multi-homed SCTP endpoints. However, there are the following issues with it:

- o It states that a SACK should be sent to the source address of an incoming DATA. However, it is known that other SACK policies (e.g. sending SACKs always to the primary path) may be more beneficial in some situations.
- o Initially it states that an endpoint should always transmit DATA chunks to the primary path. Then it states that the rule for transmittal of reply chunks should also be followed if the endpoint is bundling DATA chunks together with the reply chunk which contradicts with the first statement to always transmit DATA

chunks to the primary path. Some implementations were having problems with it and sent DATA chunks bundled with reply chunks to a different destination address than the primary path that caused many gaps.

### 3.29.2. Text Changes to the Document

-----

Old text: (Section 6.4)

-----

An endpoint SHOULD transmit reply chunks (e.g., SACK, HEARTBEAT ACK, etc.) to the same destination transport address from which it received the DATA or control chunk to which it is replying. This rule should also be followed if the endpoint is bundling DATA chunks together with the reply chunk.

However, when acknowledging multiple DATA chunks received in packets from different source addresses in a single SACK, the SACK chunk may be transmitted to one of the destination transport addresses from which the DATA or control chunks being acknowledged were received.

-----

New text: (Section 6.4)

-----

An endpoint SHOULD transmit reply chunks (e.g., INIT ACK, COOKIE ACK, HEARTBEAT ACK, etc.) in response to control chunks to the same destination transport address from which it received the control chunk to which it is replying.

The selection of the destination transport address for packets containing SACK chunks is implementation dependent. However, an endpoint SHOULD NOT vary the destination transport address of a SACK when it receives DATA chunks coming from the same source address.

When acknowledging multiple DATA chunks received in packets from different source addresses in a single SACK, the SACK chunk MAY be transmitted to one of the destination transport addresses from which the DATA or control chunks being acknowledged were received.

This text is in final form, and is not further updated in this document.

### 3.29.3. Solution Description

The SACK transmission policy is left implementation dependent but it is specified to not vary the destination address of a packet containing a SACK chunk unless there are reasons for it as it may negatively impact RTT measurement.

A confusing statement that prescribes to follow the rule for transmittal of reply chunks when the endpoint is bundling DATA chunks together with the reply chunk is removed.

## 3.30. Outstanding Data, Flightsize and Data In Flight Key Terms

### 3.30.1. Description of the Problem

[RFC4960] uses outstanding data, flightsize and data in flight key terms in formulas and statements but their definitions are not provided in Section 1.3. Furthermore, outstanding data does not include DATA chunks which are classified as lost but which have not been retransmitted yet and there is a paragraph in Section 6.1 of [RFC4960] where this statement is broken.

### 3.30.2. Text Changes to the Document

-----

Old text: (Section 1.3)

-----

- o Congestion window (cwnd): An SCTP variable that limits the data, in number of bytes, a sender can send to a particular destination transport address before receiving an acknowledgement.

...

- o Outstanding TSN (at an SCTP endpoint): A TSN (and the associated DATA chunk) that has been sent by the endpoint but for which it has not yet received an acknowledgement.

-----

New text: (Section 1.3)

-----

- o Outstanding TSN (at an SCTP endpoint): A TSN (and the associated DATA chunk) that has been sent by the endpoint but for which it has not yet received an acknowledgement.
- o Outstanding data (or Data outstanding or Data in flight): The total amount of the DATA chunks associated with outstanding TSNs. A retransmitted DATA chunk is counted once in outstanding data. A DATA chunk which is classified as lost but which has not been retransmitted yet is not in outstanding data.
- o Flightsize: The amount of bytes of outstanding data to a particular destination transport address at any given time.
- o Congestion window (cwnd): An SCTP variable that limits outstanding data, in number of bytes, a sender can send to a particular destination transport address before receiving an acknowledgement.

This text is in final form, and is not further updated in this document.



-----

Old text: (Section 6.1)

-----

- C) When the time comes for the sender to transmit, before sending new DATA chunks, the sender MUST first transmit any outstanding DATA chunks that are marked for retransmission (limited by the current cwnd).

-----

New text: (Section 6.1)

-----

- C) When the time comes for the sender to transmit, before sending new DATA chunks, the sender MUST first transmit any DATA chunks that are marked for retransmission (limited by the current cwnd).

This text is in final form, and is not further updated in this document.

### 3.30.3. Solution Description

Now Section 1.3, Key Terms, includes explanations of outstanding data, data in flight and flightsize key terms. Section 6.1 is corrected to properly use the outstanding data term.

### 3.31. CWND Degradation due to Max.Burst

#### 3.31.1. Description of the Problem

Some implementations were experiencing a degradation of cwnd because of the Max.Burst limit. This was due to misinterpretation of the suggestion in [RFC4960], Section 6.1, on how to use the Max.Burst parameter when calculating the number of packets to transmit.

#### 3.31.2. Text Changes to the Document

-----

Old text: (Section 6.1)

-----

- D) When the time comes for the sender to transmit new DATA chunks, the protocol parameter Max.Burst SHOULD be used to limit the number of packets sent. The limit MAY be applied by adjusting cwnd as follows:

```
if((flightsize + Max.Burst*MTU) < cwnd) cwnd = flightsize +
Max.Burst*MTU
```

Or it MAY be applied by strictly limiting the number of packets emitted by the output routine.

-----

New text: (Section 6.1)

-----

- D) When the time comes for the sender to transmit new DATA chunks, the protocol parameter Max.Burst SHOULD be used to limit the number of packets sent. The limit MAY be applied by adjusting cwnd temporarily as follows:

```
if ((flightsize + Max.Burst*MTU) < cwnd)
    cwnd = flightsize + Max.Burst*MTU
```

Or it MAY be applied by strictly limiting the number of packets emitted by the output routine. When calculating the number of packets to transmit and particularly using the formula above, cwnd SHOULD NOT be changed permanently.

This text is in final form, and is not further updated in this document.

### 3.31.3. Solution Description

The new text clarifies that cwnd should not be changed when applying the Max.Burst limit. This mitigates packet bursts related to the reception of SACK chunks, but not bursts related to an application sending a burst of user messages.

### 3.32. Reduction of RTO.Initial

### 3.32.1. Description of the Problem

[RFC4960] uses 3 seconds as the default value for RTO.Initial in accordance with Section 4.3.2.1 of [RFC1122]. [RFC6298] updates [RFC1122] and lowers the initial value of the retransmission timer from 3 seconds to 1 second.

### 3.32.2. Text Changes to the Document

-----  
Old text: (Section 15)  
-----

The following protocol parameters are RECOMMENDED:

RTO.Initial - 3 seconds  
RTO.Min - 1 second  
RTO.Max - 60 seconds  
Max.Burst - 4  
RTO.Alpha - 1/8  
RTO.Beta - 1/4  
Valid.Cookie.Life - 60 seconds  
Association.Max.Retrans - 10 attempts  
Path.Max.Retrans - 5 attempts (per destination address)  
Max.Init.Retransmits - 8 attempts  
HB.interval - 30 seconds  
HB.Max.Burst - 1

-----  
New text: (Section 15)  
-----

The following protocol parameters are RECOMMENDED:

RTO.Initial - 1 second  
RTO.Min - 1 second  
RTO.Max - 60 seconds  
Max.Burst - 4  
RTO.Alpha - 1/8  
RTO.Beta - 1/4  
Valid.Cookie.Life - 60 seconds  
Association.Max.Retrans - 10 attempts  
Path.Max.Retrans - 5 attempts (per destination address)  
Max.Init.Retransmits - 8 attempts  
HB.interval - 30 seconds  
HB.Max.Burst - 1  
SACK.Delay - 200 milliseconds

This text has been modified by multiple errata. It includes modifications from Section 3.24. It is in final form, and is not further updated in this document.

### 3.32.3. Solution Description

The value `RTO.Initial` has been lowered to 1 second to be in tune with [RFC6298].

## 3.33. Ordering of Bundled SACK and ERROR Chunks

### 3.33.1. Description of the Problem

When an SCTP endpoint receives a DATA chunk with an invalid stream identifier it shall acknowledge it by sending a SACK chunk and indicate that the stream identifier was invalid by sending an ERROR chunk. These two chunks may be bundled. However, [RFC4960] requires in case of bundling that the ERROR chunk follows the SACK chunk. This restriction of the ordering is not necessary and might only limit interoperability.

### 3.33.2. Text Changes to the Document

-----

Old text: (Section 6.5)

-----

Every DATA chunk MUST carry a valid stream identifier. If an endpoint receives a DATA chunk with an invalid stream identifier, it shall acknowledge the reception of the DATA chunk following the normal procedure, immediately send an ERROR chunk with cause set to "Invalid Stream Identifier" (see Section 3.3.10), and discard the DATA chunk. The endpoint may bundle the ERROR chunk in the same packet as the SACK as long as the ERROR follows the SACK.

-----

New text: (Section 6.5)

-----

Every DATA chunk MUST carry a valid stream identifier. If an endpoint receives a DATA chunk with an invalid stream identifier, it SHOULD acknowledge the reception of the DATA chunk following the normal procedure, immediately send an ERROR chunk with cause set to "Invalid Stream Identifier" (see Section 3.3.10), and discard the DATA chunk. The endpoint MAY bundle the ERROR chunk and the SACK Chunk in the same packet.

This text is in final form, and is not further updated in this document.

### 3.33.3. Solution Description

The unnecessary restriction regarding the ordering of the SACK and ERROR chunk has been removed.

## 3.34. Undefined Parameter Returned by RECEIVE Primitive

### 3.34.1. Description of the Problem

[RFC4960] provides a description of an abstract API. In the definition of the RECEIVE primitive an optional parameter with name "delivery number" is mentioned. However, no definition of this parameter is given in [RFC4960] and the parameter is unnecessary.

### 3.34.2. Text Changes to the Document

-----

Old text: (Section 10.1 G))

-----

#### G) Receive

Format: RECEIVE(association id, buffer address, buffer size  
[,stream id])

-> byte count [,transport address] [,stream id] [,stream sequence  
number] [,partial flag] [,delivery number] [,payload protocol-id]

-----

New text: (Section 10.1 G))

-----

#### G) Receive

Format: RECEIVE(association id, buffer address, buffer size  
[,stream id])

-> byte count [,transport address] [,stream id] [,stream sequence  
number] [,partial flag] [,payload protocol-id]

This text is in final form, and is not further updated in this document.

### 3.34.3. Solution Description

The undefined parameter has been removed.

### 3.35. DSCP Changes

#### 3.35.1. Description of the Problem

The upper layer can change the Differentiated Services Code Point (DSCP) used for packets being sent. A change of the DSCP can result in packets hitting different queues on the path and, therefore, the congestion control should be initialized when the DSCP is changed by the upper layer. This is not described in [RFC4960].

#### 3.35.2. Text Changes to the Document

-----

New text: (Section 7.2.5)

-----

#### 7.2.5. Change of Differentiated Services Code Points

SCTP implementations MAY allow an application to configure the Differentiated Services Code Point (DSCP) used for sending packets. If a DSCP change might result in outgoing packets being queued in different queues, the congestion control parameters for all affected destination addresses MUST be reset to their initial values.

This text is in final form, and is not further updated in this document.

-----  
Old text: (Section 10.1 M)  
-----

Mandatory attributes:

- o association id - local handle to the SCTP association.
- o protocol parameter list - the specific names and values of the protocol parameters (e.g., Association.Max.Retrans; see Section 15) that the SCTP user wishes to customize.

-----  
New text: (Section 10.1 M)  
-----

Mandatory attributes:

- o association id - local handle to the SCTP association.
- o protocol parameter list - the specific names and values of the protocol parameters (e.g., Association.Max.Retrans; see Section 15, or other parameters like the DSCP) that the SCTP user wishes to customize.

This text is in final form, and is not further updated in this document.

### 3.35.3. Solution Description

Text describing the required action on DSCP changes has been added.

### 3.36. Inconsistent Handling of ICMPv4 and ICMPv6 Messages

#### 3.36.1. Description of the Problem

Appendix C of [RFC4960] describes the handling of ICMPv4 and ICMPv6 messages. The handling of ICMP messages indicating that the port number is unreachable described in the enumeration is not consistent with the description given in [RFC4960] after the enumeration. Furthermore, the text explicitly describes the handling of ICMPv6 packets indicating reachability problems, but does not do the same for the corresponding ICMPv4 packets.

## 3.36.2. Text Changes to the Document

-----  
Old text: (Appendix C)  
-----

ICMP3) An implementation MAY ignore any ICMPv4 messages where the code does not indicate "Protocol Unreachable" or "Fragmentation Needed".

-----  
New text: (Appendix C)  
-----

ICMP3) An implementation SHOULD ignore any ICMP messages where the code indicates "Port Unreachable".

This text is in final form, and is not further updated in this document.

-----  
Old text: (Appendix C)  
-----

ICMP9) If the ICMPv6 code is "Destination Unreachable", the implementation MAY mark the destination into the unreachable state or alternatively increment the path error counter.

-----  
New text: (Appendix C)  
-----

ICMP9) If the ICMP type is "Destination Unreachable", the implementation MAY mark the destination into the unreachable state or alternatively increment the path error counter.

This text has been modified by multiple errata. It is further updated in Section 3.37.

## 3.36.3. Solution Description

The text has been changed to describe the intended handling of ICMP messages indicating that the port number is unreachable by replacing the third rule. Furthermore, remove the limitation to ICMPv6 in the ninth rule.



### 3.37. Handling of Soft Errors

#### 3.37.1. Description of the Problem

[RFC1122] defines the handling of soft errors and hard errors for TCP. Appendix C of [RFC4960] only deals with hard errors.

#### 3.37.2. Text Changes to the Document

-----  
Old text: (Appendix C)  
-----

ICMP9) If the ICMPv6 code is "Destination Unreachable", the implementation MAY mark the destination into the unreachable state or alternatively increment the path error counter.

-----  
New text: (Appendix C)  
-----

ICMP9) If the ICMP type is "Destination Unreachable", the implementation MAY mark the destination into the unreachable state or alternatively increment the path error counter. SCTP MAY provide information to the upper layer indicating the reception of ICMP messages when reporting a network status change.

This text has been modified by multiple errata. It includes modifications from Section 3.36. It is in final form, and is not further updated in this document.

#### 3.37.3. Solution Description

Text has been added allowing SCTP to notify the application in case of soft errors.

### 3.38. Honoring CWND

#### 3.38.1. Description of the Problem

When using the slow start algorithm, SCTP increases the congestion window only when it is being fully utilized. Since SCTP uses DATA chunks and does not use the congestion window to fragment user messages, this requires that some overbooking of the congestion window is allowed.

## 3.38.2. Text Changes to the Document

-----  
Old text: (Section 6.1)  
-----

- B) At any given time, the sender MUST NOT transmit new data to a given transport address if it has cwnd or more bytes of data outstanding to that transport address.

-----  
New text: (Section 6.1)  
-----

- B) At any given time, the sender MUST NOT transmit new data to a given transport address if it has cwnd + (PMTU - 1) or more bytes of data outstanding to that transport address. If data is available the sender SHOULD exceed cwnd by up to (PMTU-1) bytes on a new data transmission if the flightsize does not currently reach cwnd. The breach of cwnd MUST constitute one packet only.

This text is in final form, and is not further updated in this document.

-----  
Old text: (Section 7.2.1)  
-----

- o Whenever cwnd is greater than zero, the endpoint is allowed to have cwnd bytes of data outstanding on that transport address.

-----  
New text: (Section 7.2.1)  
-----

- o Whenever cwnd is greater than zero, the endpoint is allowed to have cwnd bytes of data outstanding on that transport address. A limited overbooking as described in B) of Section 6.1 SHOULD be supported.

This text is in final form, and is not further updated in this document.

## 3.38.3. Solution Description

Text was added to clarify how the cwnd limit should be handled.

### 3.39. Zero Window Probing

#### 3.39.1. Description of the Problem

The text describing zero window probing was not clearly handling the case where the window was not zero, but too small for the next DATA chunk to be transmitted. Even in this case, zero window probing has to be performed to avoid deadlocks.

#### 3.39.2. Text Changes to the Document

-----

Old text: (Section 6.1)

-----

- A) At any given time, the data sender MUST NOT transmit new data to any destination transport address if its peer's `rwnd` indicates that the peer has no buffer space (i.e., `rwnd` is 0; see Section 6.2.1). However, regardless of the value of `rwnd` (including if it is 0), the data sender can always have one DATA chunk in flight to the receiver if allowed by `cwnd` (see rule B, below). This rule allows the sender to probe for a change in `rwnd` that the sender missed due to the SACK's having been lost in transit from the data receiver to the data sender.

When the receiver's advertised window is zero, this probe is called a zero window probe. Note that a zero window probe SHOULD only be sent when all outstanding DATA chunks have been cumulatively acknowledged and no DATA chunks are in flight. Zero window probing MUST be supported.

-----

New text: (Section 6.1)

-----

- A) At any given time, the data sender MUST NOT transmit new data to any destination transport address if its peer's `rwnd` indicates that the peer has no buffer space (i.e., `rwnd` is smaller than the size of the next DATA chunk; see Section 6.2.1). However, regardless of the value of `rwnd` (including if it is 0), the data sender can always have one DATA chunk in flight to the receiver if allowed by `cwnd` (see rule B, below). This rule allows the sender to probe for a change in `rwnd` that the sender missed due to the SACK's having been lost in transit from the data receiver to the data sender.

When the receiver has no buffer space, this probe is called a zero window probe. Note that a zero window probe SHOULD only be sent when all outstanding DATA chunks have been cumulatively acknowledged and no DATA chunks are in flight. Zero window probing MUST be supported.

This text is in final form, and is not further updated in this document.

### 3.39.3. Solution Description

The terminology is used in a cleaner way.

### 3.40. Updating References Regarding ECN

#### 3.40.1. Description of the Problem

[RFC4960] refers for ECN only to [RFC3168], which will be updated by [RFC8311]. This needs to be reflected when referring to ECN.

#### 3.40.2. Text Changes to the Document

-----  
Old text: (Appendix A)  
-----

ECN [RFC3168] describes a proposed extension to IP that details a method to become aware of congestion outside of datagram loss.

-----  
New text: (Appendix A)  
-----

ECN as specified in [RFC3168] updated by [RFC8311] describes an extension to IP that details a method to become aware of congestion outside of datagram loss.

This text is in final form, and is not further updated in this document.

-----  
Old text: (Appendix A)  
-----

In general, [RFC3168] should be followed with the following exceptions.

-----  
New text: (Appendix A)  
-----

In general, [RFC3168] updated by [RFC8311] SHOULD be followed with the following exceptions.

This text is in final form, and is not further updated in this document.

-----  
Old text: (Appendix A)  
-----

[RFC3168] details negotiation of ECN during the SYN and SYN-ACK stages of a TCP connection.

-----  
New text: (Appendix A)  
-----

[RFC3168] updated by [RFC8311] details negotiation of ECN during the SYN and SYN-ACK stages of a TCP connection.

This text is in final form, and is not further updated in this document.

-----  
Old text: (Appendix A)  
-----

[RFC3168] details a specific bit for a receiver to send back in its TCP acknowledgements to notify the sender of the Congestion Experienced (CE) bit having arrived from the network.

-----  
New text: (Appendix A)  
-----

[RFC3168] updated by [RFC8311] details a specific bit for a receiver to send back in its TCP acknowledgements to notify the sender of the Congestion Experienced (CE) bit having arrived from the network.

This text is in final form, and is not further updated in this document.

-----  
Old text: (Appendix A)  
-----

[RFC3168] details a specific bit for a sender to send in the header of its next outbound TCP segment to indicate to its peer that it has reduced its congestion window.

-----  
New text: (Appendix A)  
-----

[RFC3168] updated by [RFC8311] details a specific bit for a sender to send in the header of its next outbound TCP segment to indicate to its peer that it has reduced its congestion window.

This text is in final form, and is not further updated in this document.

### 3.40.3. Solution Description

References to [RFC8311] have been added. While there, some wordsmithing has been performed.

### 3.41. Host Name Address Parameter Deprecated

#### 3.41.1. Description of the Problem

[RFC4960] defines three types of address parameters to be used with INIT and INIT ACK chunks:

1. IPv4 Address parameters.
2. IPv6 Address parameters.
3. Host Name Address parameters.

The first two are supported by the SCTP kernel implementations of FreeBSD, Linux and Solaris, but the third one is not. In addition, the first two were successfully tested in all nine interoperability tests for SCTP, but the third one has never been successfully tested. Therefore, the Host Name Address parameter should be deprecated.

#### 3.41.2. Text Changes to the Document

-----  
Old text: (Section 3.3.2)  
-----

Note 3: An INIT chunk MUST NOT contain more than one Host Name Address parameter. Moreover, the sender of the INIT MUST NOT combine any other address types with the Host Name Address in the INIT. The receiver of INIT MUST ignore any other address types if the Host Name Address parameter is present in the received INIT chunk.

-----  
New text: (Section 3.3.2)  
-----

Note 3: An INIT chunk MUST NOT contain the Host Name Address parameter. The receiver of an INIT chunk containing a Host Name Address parameter MUST send an ABORT and MAY include an Error Cause indicating an Unresolvable Address.

This text is in final form, and is not further updated in this document.

-----  
Old text: (Section 3.3.2.1)  
-----

The sender of INIT uses this parameter to pass its Host Name (in place of its IP addresses) to its peer. The peer is responsible for resolving the name. Using this parameter might make it more likely for the association to work across a NAT box.

-----  
New text: (Section 3.3.2.1)  
-----

The sender of an INIT chunk MUST NOT include this parameter. The usage of the Host Name Address parameter is deprecated.

This text is in final form, and is not further updated in this document.



-----

Old text: (Section 3.3.2.1)

-----

Address Type: 16 bits (unsigned integer)

This is filled with the type value of the corresponding address TLV (e.g., IPv4 = 5, IPv6 = 6, Host name = 11).

-----

New text: (Section 3.3.2.1)

-----

Address Type: 16 bits (unsigned integer)

This is filled with the type value of the corresponding address TLV (e.g., IPv4 = 5, IPv6 = 6). The value indicating the Host Name Address parameter (Host name = 11) MUST NOT be used.

This text is in final form, and is not further updated in this document.

-----

Old text: (Section 3.3.3)

-----

Note 3: The INIT ACK chunks MUST NOT contain more than one Host Name Address parameter. Moreover, the sender of the INIT ACK MUST NOT combine any other address types with the Host Name Address in the INIT ACK. The receiver of the INIT ACK MUST ignore any other address types if the Host Name Address parameter is present.

-----

New text: (Section 3.3.3)

-----

Note 3: An INIT ACK chunk MUST NOT contain the Host Name Address parameter. The receiver of INIT ACK chunks containing a Host Name Address parameter MUST send an ABORT and MAY include an Error Cause indicating an Unresolvable Address.

This text is in final form, and is not further updated in this document.

-----

Old text: (Section 5.1.2)

-----

B) If there is a Host Name parameter present in the received INIT or

INIT ACK chunk, the endpoint shall resolve that host name to a list of IP address(es) and derive the transport address(es) of this peer by combining the resolved IP address(es) with the SCTP source port.

The endpoint MUST ignore any other IP Address parameters if they are also present in the received INIT or INIT ACK chunk.

The time at which the receiver of an INIT resolves the host name has potential security implications to SCTP. If the receiver of an INIT resolves the host name upon the reception of the chunk, and the mechanism the receiver uses to resolve the host name involves potential long delay (e.g., DNS query), the receiver may open itself up to resource attacks for the period of time while it is waiting for the name resolution results before it can build the State Cookie and release local resources.

Therefore, in cases where the name translation involves potential long delay, the receiver of the INIT MUST postpone the name resolution till the reception of the COOKIE ECHO chunk from the peer. In such a case, the receiver of the INIT SHOULD build the State Cookie using the received Host Name (instead of destination transport addresses) and send the INIT ACK to the source IP address from which the INIT was received.

The receiver of an INIT ACK shall always immediately attempt to resolve the name upon the reception of the chunk.

The receiver of the INIT or INIT ACK MUST NOT send user data (piggy-backed or stand-alone) to its peer until the host name is successfully resolved.

If the name resolution is not successful, the endpoint MUST immediately send an ABORT with "Unresolvable Address" error cause to its peer. The ABORT shall be sent to the source IP address from which the last peer packet was received.

-----  
New text: (Section 5.1.2)  
-----

- B) If there is a Host Name parameter present in the received INIT or INIT ACK chunk, the endpoint MUST immediately send an ABORT and MAY include an Error Cause indicating an Unresolvable Address to its peer. The ABORT SHALL be sent to the source IP address from which the last peer packet was received.

This text is in final form, and is not further updated in this document.

-----  
Old text: (Section 11.2.4.1)  
-----

The use of the host name feature in the INIT chunk could be used to flood a target DNS server. A large backlog of DNS queries, resolving the host name received in the INIT chunk to IP addresses, could be accomplished by sending INITs to multiple hosts in a given domain. In addition, an attacker could use the host name feature in an indirect attack on a third party by sending large numbers of INITs to random hosts containing the host name of the target. In addition to the strain on DNS resources, this could also result in large numbers of INIT ACKs being sent to the target. One method to protect against this type of attack is to verify that the IP addresses received from DNS include the source IP address of the original INIT. If the list of IP addresses received from DNS does not include the source IP address of the INIT, the endpoint MAY silently discard the INIT. This last option will not protect against the attack against the DNS.

-----  
New text: (Section 11.2.4.1)  
-----

The support of the Host Name Address parameter has been removed from the protocol. Endpoints receiving INIT or INIT ACK chunks containing the Host Name Address parameter MUST send an ABORT chunk in response and MAY include an Error Cause indicating an Unresolvable Address.

This text is in final form, and is not further updated in this document.

### 3.41.3. Solution Description

The usage of the Host Name Address parameter has been deprecated.

### 3.42. Conflicting Text Regarding the Supported Address Types Parameter

#### 3.42.1. Description of the Problem

When receiving an SCTP packet containing an INIT chunk sent from an address for which the corresponding address type is not listed in the Supported Address Types, there is conflicting text in Section 5.1.2 of [RFC4960]. It is stated that the association MUST be aborted and also that the association SHOULD be established and there SHOULD NOT be any error indication.

### 3.42.2. Text Changes to the Document

-----  
Old text: (Section 5.1.2)  
-----

The sender of INIT may include a 'Supported Address Types' parameter in the INIT to indicate what types of address are acceptable. When this parameter is present, the receiver of INIT (initiate) MUST either use one of the address types indicated in the Supported Address Types parameter when responding to the INIT, or abort the association with an "Unresolvable Address" error cause if it is unwilling or incapable of using any of the address types indicated by its peer.

-----  
New text: (Section 5.1.2)  
-----

The sender of INIT chunks MAY include a 'Supported Address Types' parameter in the INIT to indicate what types of addresses are acceptable.

This text is in final form, and is not further updated in this document.

### 3.42.3. Solution Description

The conflicting text has been removed.

## 3.43. Integration of RFC 6096

### 3.43.1. Description of the Problem

[RFC6096] updates [RFC4960] by adding a Chunk Flags Registry. This should be integrated into the base specification.

### 3.43.2. Text Changes to the Document

-----  
Old text: (Section 14.1)  
-----

#### 14.1. IETF-Defined Chunk Extension

The assignment of new chunk parameter type codes is done through an IETF Consensus action, as defined in [RFC2434]. Documentation of the chunk parameter MUST contain the following information:

- a) A long and short name for the new chunk type.
- b) A detailed description of the structure of the chunk, which MUST conform to the basic structure defined in Section 3.2.
- c) A detailed definition and description of the intended use of each field within the chunk, including the chunk flags if any.
- d) A detailed procedural description of the use of the new chunk type within the operation of the protocol.

The last chunk type (255) is reserved for future extension if necessary.

-----  
New text: (Section 14.1)  
-----

#### 14.1. IETF-Defined Chunk Extension

The assignment of new chunk type codes is done through an IETF Review action, as defined in [RFC8126]. Documentation of a new chunk MUST contain the following information:

- a) A long and short name for the new chunk type;
- b) A detailed description of the structure of the chunk, which MUST conform to the basic structure defined in Section 3.2 of [RFC4960];
- c) A detailed definition and description of the intended use of each field within the chunk, including the chunk flags if any. Defined chunk flags will be used as initial entries in the chunk flags table for the new chunk type;
- d) A detailed procedural description of the use of the new chunk type within the operation of the protocol.

The last chunk type (255) is reserved for future extension if necessary.

For each new chunk type, IANA creates a registration table for the chunk flags of that type. The procedure for registering particular chunk flags is described in the following Section 14.2.

This text has been modified by multiple errata. It includes modifications from Section 3.3. It is in final form, and is not further updated in this document.

-----  
New text: (Section 14.2)  
-----

#### 14.2. New IETF Chunk Flags Registration

The assignment of new chunk flags is done through an RFC required action, as defined in [RFC8126]. Documentation of the chunk flags MUST contain the following information:

- a) A name for the new chunk flag;
- b) A detailed procedural description of the use of the new chunk flag within the operation of the protocol. It MUST be considered that implementations not supporting the flag will send '0' on transmit and just ignore it on receipt.

IANA selects a chunk flags value. This MUST be one of 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, or 0x80, which MUST be unique within the chunk flag values for the specific chunk type.

This text is in final form, and is not further updated in this document.

Please note that Sections 14.2, 14.3, 14.4, and 14.5 need to be renumbered.

#### 3.43.3. Solution Description

[RFC6096] was integrated and the reference updated to [RFC8126].

#### 3.44. Integration of RFC 6335

##### 3.44.1. Description of the Problem

[RFC6335] updates [RFC4960] by updating Procedures for the Port Numbers Registry. This should be integrated into the base specification. While there, update the reference to the RFC giving guidelines for writing IANA sections to [RFC8126].

##### 3.44.2. Text Changes to the Document

-----  
Old text: (Section 14.5)  
-----

SCTP services may use contact port numbers to provide service to unknown callers, as in TCP and UDP. IANA is therefore requested to

open the existing Port Numbers registry for SCTP using the following rules, which we intend to mesh well with existing Port Numbers registration procedures. An IESG-appointed Expert Reviewer supports IANA in evaluating SCTP port allocation requests, according to the procedure defined in [RFC2434].

Port numbers are divided into three ranges. The Well Known Ports are those from 0 through 1023, the Registered Ports are those from 1024 through 49151, and the Dynamic and/or Private Ports are those from 49152 through 65535. Well Known and Registered Ports are intended for use by server applications that desire a default contact point on a system. On most systems, Well Known Ports can only be used by system (or root) processes or by programs executed by privileged users, while Registered Ports can be used by ordinary user processes or programs executed by ordinary users. Dynamic and/or Private Ports are intended for temporary use, including client-side ports, out-of-band negotiated ports, and application testing prior to registration of a dedicated port; they MUST NOT be registered.

The Port Numbers registry should accept registrations for SCTP ports in the Well Known Ports and Registered Ports ranges. Well Known and Registered Ports SHOULD NOT be used without registration. Although in some cases -- such as porting an application from TCP to SCTP -- it may seem natural to use an SCTP port before registration completes, we emphasize that IANA will not guarantee registration of particular Well Known and Registered Ports. Registrations should be requested as early as possible.

Each port registration SHALL include the following information:

- o A short port name, consisting entirely of letters (A-Z and a-z), digits (0-9), and punctuation characters from "-\_+./\*" (not including the quotes).
- o The port number that is requested for registration.
- o A short English phrase describing the port's purpose.
- o Name and contact information for the person or entity performing the registration, and possibly a reference to a document defining the port's use. Registrations coming from IETF working groups need only name the working group, but indicating a contact person is recommended.

Registrants are encouraged to follow these guidelines when submitting a registration.

- o A port name SHOULD NOT be registered for more than one SCTP port





```

|  Type = 0      | Reserved|U|B|E|      Length      |
+-----+-----+-----+-----+-----+-----+
|                                     TSN                                     |
+-----+-----+-----+-----+-----+-----+
|  Stream Identifier S      | Stream Sequence Number n      |
+-----+-----+-----+-----+-----+-----+
|                                     Payload Protocol Identifier                                     |
+-----+-----+-----+-----+-----+-----+
\                                                                                   \
/                                     User Data (seq n of Stream S)                  /
\                                                                                   \
+-----+-----+-----+-----+-----+-----+

```

Reserved: 5 bits

Should be set to all '0's and ignored by the receiver.

-----  
New text: (Section 3.3.1)  
-----

```

      0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+
|  Type = 0      | Res  |I|U|B|E|      Length      |
+-----+-----+-----+-----+-----+-----+
|                                     TSN                                     |
+-----+-----+-----+-----+-----+-----+
|  Stream Identifier S      | Stream Sequence Number n      |
+-----+-----+-----+-----+-----+-----+
|                                     Payload Protocol Identifier                                     |
+-----+-----+-----+-----+-----+-----+
\                                                                                   \
/                                     User Data (seq n of Stream S)                  /
\                                                                                   \
+-----+-----+-----+-----+-----+-----+

```

Res: 4 bits

SHOULD be set to all '0's and ignored by the receiver.

I bit: 1 bit

The (I)mmediate Bit MAY be set by the sender, whenever the sender of a DATA chunk can benefit from the corresponding SACK chunk being sent back without delay. See [RFC7053] for a discussion about

This text is in final form, and is not further updated in this document.

-----  
New text: (Append to Section 6.1)  
-----

Whenever the sender of a DATA chunk can benefit from the corresponding SACK chunk being sent back without delay, the sender MAY set the I bit in the DATA chunk header. Please note that why the sender has set the I bit is irrelevant to the receiver.

Reasons for setting the I bit include, but are not limited to (see Section 4 of [RFC7053] for the benefits):

- o The application requests to set the I bit of the last DATA chunk of a user message when providing the user message to the SCTP implementation (see Section 7).
- o The sender is in the SHUTDOWN-PENDING state.
- o The sending of a DATA chunk fills the congestion or receiver window.

This text is in final form, and is not further updated in this document.

-----  
Old text: (Section 6.2)  
-----

Note: The SHUTDOWN chunk does not contain Gap Ack Block fields. Therefore, the endpoint should use a SACK instead of the SHUTDOWN chunk to acknowledge DATA chunks received out of order.

-----  
New text: (Section 6.2)  
-----

Note: The SHUTDOWN chunk does not contain Gap Ack Block fields. Therefore, the endpoint SHOULD use a SACK instead of the SHUTDOWN chunk to acknowledge DATA chunks received out of order.

Upon receipt of an SCTP packet containing a DATA chunk with the I bit set, the receiver SHOULD NOT delay the sending of the corresponding SACK chunk, i.e., the receiver SHOULD immediately respond with the corresponding SACK chunk.

Please note that this change is only about adding a paragraph.

This text is in final form, and is not further updated in this document.

-----

Old text: (Section 10.1 E))

-----

E) Send

Format: SEND(association id, buffer address, byte count [,context]  
[,stream id] [,life time] [,destination transport address]  
[,unordered flag] [,no-bundle flag] [,payload protocol-id] )  
-> result

-----

New text: (Section 10.1 E))

-----

E) Send

Format: SEND(association id, buffer address, byte count [,context]  
[,stream id] [,life time] [,destination transport address]  
[,unordered flag] [,no-bundle flag] [,payload protocol-id]  
[,sack immediately] )  
-> result

This text is in final form, and is not further updated in this document.

-----

New text: (Append optional parameter in Subsection E of Section 10.1)

-----

- o sack immediately - set the I bit on the last DATA chunk used for sending buffer.

This text is in final form, and is not further updated in this document.

### 3.45.3. Solution Description

[RFC7053] was integrated.

## 3.46. CRC32c Code Improvements

## 3.46.1. Description of the Problem

The code given for the CRC32c computations uses types like long which may have different length on different operating systems or processors. Therefore, the code is changed to use specific types like uint32\_t.

While there, fix also some syntax errors and a comment.

## 3.46.2. Text Changes to the Document

```

-----
Old text: (Appendix C)
-----
/*****/
/* Note Definition for Ross Williams table generator would */
/* be: TB_WIDTH=4, TB_POLLY=0x1EDC6F41, TB_REVER=TRUE */
/* For Mr. Williams direct calculation code use the settings */
/* cm_width=32, cm_poly=0x1EDC6F41, cm_init=0xFFFFFFFF, */
/* cm_refin=TRUE, cm_refot=TRUE, cm_xorort=0x00000000 */
/*****/

/* Example of the crc table file */
#ifndef __crc32cr_table_h__
#define __crc32cr_table_h__

#define CRC32C_POLY 0x1EDC6F41
#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])

unsigned long crc_c[256] =
{
0x00000000L, 0xF26B8303L, 0xE13B70F7L, 0x1350F3F4L,
0xC79A971FL, 0x35F1141CL, 0x26A1E7E8L, 0xD4CA64EBL,
0x8AD958CFL, 0x78B2DBCCL, 0x6BE22838L, 0x9989AB3BL,
0x4D43CFD0L, 0xBF284CD3L, 0xAC78BF27L, 0x5E133C24L,
0x105EC76FL, 0xE235446CL, 0xF165B798L, 0x030E349BL,
0xD7C45070L, 0x25AFD373L, 0x36FF2087L, 0xC494A384L,
0x9A879FA0L, 0x68EC1CA3L, 0x7BBCEF57L, 0x89D76C54L,
0x5D1D08BFL, 0xAF768BBCL, 0xBC267848L, 0x4E4DFB4BL,
0x20BD8EDEL, 0xD2D60DDDL, 0xC186FE29L, 0x33ED7D2AL,
0xE72719C1L, 0x154C9AC2L, 0x061C6936L, 0xF477EA35L,
0xAA64D611L, 0x580F5512L, 0x4B5FA6E6L, 0xB93425E5L,
0x6DFE410EL, 0x9F95C20DL, 0x8CC531F9L, 0x7EAE2FAL,
0x30E349B1L, 0xC288CAB2L, 0xD1D83946L, 0x23B3BA45L,

```

0xF779DEAEL, 0x05125DADL, 0x1642AE59L, 0xE4292D5AL,  
0xBA3A117EL, 0x4851927DL, 0x5B016189L, 0xA96AE28AL,  
0x7DA08661L, 0x8FCB0562L, 0x9C9BF696L, 0x6EF07595L,  
0x417B1DBCL, 0xB3109EBFL, 0xA0406D4BL, 0x522BEE48L,  
0x86E18AA3L, 0x748A09A0L, 0x67DAFA54L, 0x95B17957L,  
0xCBA24573L, 0x39C9C670L, 0x2A993584L, 0xD8F2B687L,  
0x0C38D26CL, 0xFE53516FL, 0xED03A29BL, 0x1F682198L,  
0x5125DAD3L, 0xA34E59D0L, 0xB01EAA24L, 0x42752927L,  
0x96BF4DCCL, 0x64D4CECFL, 0x77843D3BL, 0x85EFBE38L,  
0xDBFC821CL, 0x2997011FL, 0x3AC7F2EBL, 0xC8AC71E8L,  
0x1C661503L, 0xEE0D9600L, 0xFD5D65F4L, 0x0F36E6F7L,  
0x61C69362L, 0x93AD1061L, 0x80FDE395L, 0x72966096L,  
0xA65C047DL, 0x5437877EL, 0x4767748AL, 0xB50CF789L,  
0xEB1FCBADL, 0x197448AEL, 0x0A24BB5AL, 0xF84F3859L,  
0x2C855CB2L, 0xDEEEDFB1L, 0xCDBE2C45L, 0x3FD5AF46L,  
0x7198540DL, 0x83F3D70EL, 0x90A324FAL, 0x62C8A7F9L,  
0xB602C312L, 0x44694011L, 0x5739B3E5L, 0xA55230E6L,  
0xFB410CC2L, 0x092A8FC1L, 0x1A7A7C35L, 0xE811FF36L,  
0x3CDB9BDDL, 0xCEB018DEL, 0xDDE0EB2AL, 0x2F8B6829L,  
0x82F63B78L, 0x709DB87BL, 0x63CD4B8FL, 0x91A6C88CL,  
0x456CAC67L, 0xB7072F64L, 0xA457DC90L, 0x563C5F93L,  
0x082F63B7L, 0xFA44E0B4L, 0xE9141340L, 0x1B7F9043L,  
0xCFB5F4A8L, 0x3DDE77ABL, 0x2E8E845FL, 0xDCE5075CL,  
0x92A8FC17L, 0x60C37F14L, 0x73938CE0L, 0x81F80FE3L,  
0x55326B08L, 0xA759E80BL, 0xB4091BFFL, 0x466298FCL,  
0x1871A4D8L, 0xEA1A27DBL, 0xF94AD42FL, 0x0B21572CL,  
0xDFEB33C7L, 0x2D80B0C4L, 0x3ED04330L, 0xCCBBC033L,  
0xA24BB5A6L, 0x502036A5L, 0x4370C551L, 0xB11B4652L,  
0x65D122B9L, 0x97BAA1BAL, 0x84EA524EL, 0x7681D14DL,  
0x2892ED69L, 0xDAF96E6AL, 0xC9A99D9EL, 0x3BC21E9DL,  
0xEF087A76L, 0x1D63F975L, 0x0E330A81L, 0xFC588982L,  
0xB21572C9L, 0x407EF1CAL, 0x532E023EL, 0xA145813DL,  
0x758FE5D6L, 0x87E466D5L, 0x94B49521L, 0x66DF1622L,  
0x38CC2A06L, 0xCAA7A905L, 0xD9F75AF1L, 0x2B9CD9F2L,  
0xFF56BD19L, 0x0D3D3E1AL, 0x1E6DCDEEL, 0xEC064EEDL,  
0xC38D26C4L, 0x31E6A5C7L, 0x22B65633L, 0xD0DDD530L,  
0x0417B1DBL, 0xF67C32D8L, 0xE52CC12CL, 0x1747422FL,  
0x49547E0BL, 0xBB3FFD08L, 0xA86F0EFCL, 0x5A048DFFL,  
0x8ECEE914L, 0x7CA56A17L, 0x6FF599E3L, 0x9D9E1AE0L,  
0xD3D3E1ABL, 0x21B862A8L, 0x32E8915CL, 0xC083125FL,  
0x144976B4L, 0xE622F5B7L, 0xF5720643L, 0x07198540L,  
0x590AB964L, 0xAB613A67L, 0xB831C993L, 0x4A5A4A90L,  
0x9E902E7BL, 0x6CFBAD78L, 0x7FAB5E8CL, 0x8DC0DD8FL,  
0xE330A81AL, 0x115B2B19L, 0x020BD8EDL, 0xF0605BEEL,  
0x24AA3F05L, 0xD6C1BC06L, 0xC5914FF2L, 0x37FACCF1L,  
0x69E9F0D5L, 0x9B8273D6L, 0x88D28022L, 0x7AB90321L,  
0xAE7367CAL, 0x5C18E4C9L, 0x4F48173DL, 0xBD23943EL,  
0xF36E6F75L, 0x0105EC76L, 0x12551F82L, 0xE03E9C81L,

```

0x34F4F86AL, 0xC69F7B69L, 0xD5CF889DL, 0x27A40B9EL,
0x79B737BAL, 0x8BDCB4B9L, 0x988C474DL, 0x6AE7C44EL,
0xBE2DA0A5L, 0x4C4623A6L, 0x5F16D052L, 0xAD7D5351L,
};

#endif

```

```

-----
New text: (Appendix B)
-----

```

```

<CODE BEGINS>
/*****
/* Note Definition for Ross Williams table generator would */
/* be: TB_WIDTH=4, TB_POLLY=0x1EDC6F41, TB_REVER=TRUE */
/* For Mr. Williams direct calculation code use the settings */
/* cm_width=32, cm_poly=0x1EDC6F41, cm_init=0xFFFFFFFF, */
/* cm_refin=TRUE, cm_refot=TRUE, cm_xorort=0x00000000 */
*****/

/* Example of the crc table file */
#ifndef __crc32cr_h__
#define __crc32cr_h__

#define CRC32C_POLY 0x1EDC6F41UL
#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])

uint32_t crc_c[256] =
{
0x00000000UL, 0xF26B8303UL, 0xE13B70F7UL, 0x1350F3F4UL,
0xC79A971FUL, 0x35F1141CUL, 0x26A1E7E8UL, 0xD4CA64EBUL,
0x8AD958CFUL, 0x78B2DBCCUL, 0x6BE22838UL, 0x9989AB3BUL,
0x4D43CFD0UL, 0xBF284CD3UL, 0xAC78BF27UL, 0x5E133C24UL,
0x105EC76FUL, 0xE235446CUL, 0xF165B798UL, 0x030E349BUL,
0xD7C45070UL, 0x25AFD373UL, 0x36FF2087UL, 0xC494A384UL,
0x9A879FA0UL, 0x68EC1CA3UL, 0x7BBCEF57UL, 0x89D76C54UL,
0x5D1D08BFUL, 0xAF768BBCUL, 0xBC267848UL, 0x4E4DFB4BUL,
0x20BD8EDEUL, 0xD2D60DDDUL, 0xC186FE29UL, 0x33ED7D2AUL,
0xE72719C1UL, 0x154C9AC2UL, 0x061C6936UL, 0xF477EA35UL,
0xAA64D611UL, 0x580F5512UL, 0x4B5FA6E6UL, 0xB93425E5UL,
0x6DFE410EUL, 0x9F95C20DUL, 0x8CC531F9UL, 0x7EAE82FAUL,
0x30E349B1UL, 0xC288CAB2UL, 0xD1D83946UL, 0x23B3BA45UL,
0xF779DEAEUL, 0x05125DADUL, 0x1642AE59UL, 0xE4292D5AUL,
0xBA3A117EUL, 0x4851927DUL, 0x5B016189UL, 0xA96AE28AUL,
0x7DA08661UL, 0x8FCB0562UL, 0x9C9BF696UL, 0x6EF07595UL,
0x417B1DBCUL, 0xB3109EBFUL, 0xA0406D4BUL, 0x522BEE48UL,
0x86E18AA3UL, 0x748A09A0UL, 0x67DAFA54UL, 0x95B17957UL,
0xCBA24573UL, 0x39C9C670UL, 0x2A993584UL, 0xD8F2B687UL,

```

```
0x0C38D26CUL, 0xFE53516FUL, 0xED03A29BUL, 0x1F682198UL,  
0x5125DAD3UL, 0xA34E59D0UL, 0xB01EAA24UL, 0x42752927UL,  
0x96BF4DCCUL, 0x64D4CECFUL, 0x77843D3BUL, 0x85EFBE38UL,  
0xDBFC821CUL, 0x2997011FUL, 0x3AC7F2EBUL, 0xC8AC71E8UL,  
0x1C661503UL, 0xEE0D9600UL, 0xFD5D65F4UL, 0x0F36E6F7UL,  
0x61C69362UL, 0x93AD1061UL, 0x80FDE395UL, 0x72966096UL,  
0xA65C047DUL, 0x5437877EUL, 0x4767748AUL, 0xB50CF789UL,  
0xEB1FCBADUL, 0x197448AEUL, 0x0A24BB5AUL, 0xF84F3859UL,  
0x2C855CB2UL, 0xDEEEDFB1UL, 0xCDBE2C45UL, 0x3FD5AF46UL,  
0x7198540DUL, 0x83F3D70EUL, 0x90A324FAUL, 0x62C8A7F9UL,  
0xB602C312UL, 0x44694011UL, 0x5739B3E5UL, 0xA55230E6UL,  
0xFB410CC2UL, 0x092A8FC1UL, 0x1A7A7C35UL, 0xE811FF36UL,  
0x3CDB9BDDUL, 0xCEB018DEUL, 0xDDE0EB2AUL, 0x2F8B6829UL,  
0x82F63B78UL, 0x709DB87BUL, 0x63CD4B8FUL, 0x91A6C88CUL,  
0x456CAC67UL, 0xB7072F64UL, 0xA457DC90UL, 0x563C5F93UL,  
0x082F63B7UL, 0xFA44E0B4UL, 0xE9141340UL, 0x1B7F9043UL,  
0xCFB5F4A8UL, 0x3DDE77ABUL, 0x2E8E845FUL, 0xDCE5075CUL,  
0x92A8FC17UL, 0x60C37F14UL, 0x73938CE0UL, 0x81F80FE3UL,  
0x55326B08UL, 0xA759E80BUL, 0xB4091BFFUL, 0x466298FCUL,  
0x1871A4D8UL, 0xEA1A27DBUL, 0xF94AD42FUL, 0x0B21572CUL,  
0xDFEB33C7UL, 0x2D80B0C4UL, 0x3ED04330UL, 0xCCBBC033UL,  
0xA24BB5A6UL, 0x502036A5UL, 0x4370C551UL, 0xB11B4652UL,  
0x65D122B9UL, 0x97BAA1BAUL, 0x84EA524EUL, 0x7681D14DUL,  
0x2892ED69UL, 0xDAF96E6AUL, 0xC9A99D9EUL, 0x3BC21E9DUL,  
0xEF087A76UL, 0x1D63F975UL, 0x0E330A81UL, 0xFC588982UL,  
0xB21572C9UL, 0x407EF1CAUL, 0x532E023EUL, 0xA145813DUL,  
0x758FE5D6UL, 0x87E466D5UL, 0x94B49521UL, 0x66DF1622UL,  
0x38CC2A06UL, 0xCA7A905UL, 0xD9F75AF1UL, 0x2B9CD9F2UL,  
0xFF56BD19UL, 0x0D3D3E1AUL, 0x1E6DCDEEUL, 0xEC064EEDUL,  
0xC38D26C4UL, 0x31E6A5C7UL, 0x22B65633UL, 0xD0DDD530UL,  
0x0417B1DBUL, 0xF67C32D8UL, 0xE52CC12CUL, 0x1747422FUL,  
0x49547E0BUL, 0xBB3FFD08UL, 0xA86F0EFCUL, 0x5A048DFFUL,  
0x8ECEE914UL, 0x7CA56A17UL, 0x6FF599E3UL, 0x9D9E1AE0UL,  
0xD3D3E1ABUL, 0x21B862A8UL, 0x32E8915CUL, 0xC083125FUL,  
0x144976B4UL, 0xE622F5B7UL, 0xF5720643UL, 0x07198540UL,  
0x590AB964UL, 0xAB613A67UL, 0xB831C993UL, 0x4A5A4A90UL,  
0x9E902E7BUL, 0x6CFBAD78UL, 0x7FAB5E8CUL, 0x8DC0DD8FUL,  
0xE330A81AUL, 0x115B2B19UL, 0x020BD8EDUL, 0xF0605BEEUL,  
0x24AA3F05UL, 0xD6C1BC06UL, 0xC5914FF2UL, 0x37FACCF1UL,  
0x69E9F0D5UL, 0x9B8273D6UL, 0x88D28022UL, 0x7AB90321UL,  
0xAE7367CAUL, 0x5C18E4C9UL, 0x4F48173DUL, 0xBD23943EUL,  
0xF36E6F75UL, 0x0105EC76UL, 0x12551F82UL, 0xE03E9C81UL,  
0x34F4F86AUL, 0xC69F7B69UL, 0xD5CF889DUL, 0x27A40B9EUL,  
0x79B737BAUL, 0x8BDCB4B9UL, 0x988C474DUL, 0x6AE7C44EUL,  
0xBE2DA0A5UL, 0x4C4623A6UL, 0x5F16D052UL, 0xAD7D5351UL,  
};
```

```
#endif
```

This text has been modified by multiple errata. It includes modifications from Section 3.10. It is in final form, and is not further updated in this document.

-----

Old text: (Appendix C)

-----

```
/* Example of table build routine */
```

```
#include <stdio.h>
#include <stdlib.h>
```

```
#define OUTPUT_FILE    "crc32cr.h"
#define CRC32C_POLY    0x1EDC6F41L
FILE *tf;
```

```
unsigned long
reflect_32 (unsigned long b)
{
    int i;
    unsigned long rw = 0L;

    for (i = 0; i < 32; i++){
        if (b & 1)
            rw |= 1 << (31 - i);
        b >>= 1;
    }
    return (rw);
}
```

```
unsigned long
build_crc_table (int index)
{
    int i;
    unsigned long rb;

    rb = reflect_32 (index);

    for (i = 0; i < 8; i++){
        if (rb & 0x80000000L)
            rb = (rb << 1) ^ CRC32C_POLY;
        else
            rb <<= 1;
    }
    return (reflect_32 (rb));
}
```



```

main ()
{
    int i;

    printf ("\nGenerating CRC-32c table file <%s>\n",
        OUTPUT_FILE);
    if ((tf = fopen (OUTPUT_FILE, "w")) == NULL){
        printf ("Unable to open %s\n", OUTPUT_FILE);
        exit (1);
    }
    fprintf (tf, "#ifndef __crc32cr_table_h__\n");
    fprintf (tf, "#define __crc32cr_table_h__\n\n");
    fprintf (tf, "#define CRC32C_POLY 0x%08lX\n",
        CRC32C_POLY);
    fprintf (tf,
        "#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])\n");
    fprintf (tf, "\nunsigned long  crc_c[256] =\n{\n");
    for (i = 0; i < 256; i++){
        fprintf (tf, "0x%08lXL, ", build_crc_table (i));
        if ((i & 3) == 3)
            fprintf (tf, "\n");
    }
    fprintf (tf, "};\n\n#endif\n");

    if (fclose (tf) != 0)
        printf ("Unable to close <%s>." OUTPUT_FILE);
    else
        printf ("\nThe CRC-32c table has been written to <%s>.\n",
            OUTPUT_FILE);
}

```

```

-----
New text: (Appendix B)
-----

```

```

/* Example of table build routine */

#include <stdio.h>
#include <stdlib.h>

#define OUTPUT_FILE    "crc32cr.h"
#define CRC32C_POLY    0x1EDC6F41UL

static FILE *tf;

static uint32_t
reflect_32(uint32_t b)
{

```

```

    int i;
    uint32_t rw = 0UL;

    for (i = 0; i < 32; i++) {
        if (b & 1)
            rw |= 1 << (31 - i);
        b >>= 1;
    }
    return (rw);
}

static uint32_t
build_crc_table(int index)
{
    int i;
    uint32_t rb;

    rb = reflect_32(index);

    for (i = 0; i < 8; i++) {
        if (rb & 0x80000000UL)
            rb = (rb << 1) ^ (uint32_t)CRC32C_POLY;
        else
            rb <<= 1;
    }
    return (reflect_32(rb));
}

int
main (void)
{
    int i;

    printf("\nGenerating CRC-32c table file <%=s>\n",
        OUTPUT_FILE);
    if ((tf = fopen(OUTPUT_FILE, "w")) == NULL) {
        printf ("Unable to open %s\n", OUTPUT_FILE);
        exit (1);
    }
    fprintf(tf, "#ifndef __crc32cr_h__\n");
    fprintf(tf, "#define __crc32cr_h__\n\n");
    fprintf(tf, "#define CRC32C_POLY 0x%08XUL\n",
        (uint32_t)CRC32C_POLY);
    fprintf(tf,
        "#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])\n");
    fprintf(tf, "\nuint32_t crc_c[256] =\n{\n");
    for (i = 0; i < 256; i++) {
        fprintf(tf, "0x%08XUL,", build_crc_table (i));

```

```
        if ((i & 3) == 3)
            fprintf(tf, "\n");
        else
            fprintf(tf, " ");
    }
    fprintf(tf, "};\n\n#endif\n");

    if (fclose (tf) != 0)
        printf("Unable to close <%s>.", OUTPUT_FILE);
    else
        printf("\nThe CRC-32c table has been written to <%s>.\n",
            OUTPUT_FILE);
}
```

This text has been modified by multiple errata. It includes modifications from Section 3.10. It is in final form, and is not further updated in this document.

-----  
Old text: (Appendix C)  
-----

```
/* Example of crc insertion */

#include "crc32cr.h"

unsigned long
generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    unsigned long crc32 = ~0L;
    unsigned long result;
    unsigned char byte0,byte1,byte2,byte3;

    for (i = 0; i < length; i++){
        CRC32C(crc32, buffer[i]);
    }

    result = ~crc32;

    /* result now holds the negated polynomial remainder;
     * since the table and algorithm is "reflected" [williams95].
     * That is, result has the same value as if we mapped the message
     * to a polynomial, computed the host-bit-order polynomial
     * remainder, performed final negation, then did an end-for-end
     * bit-reversal.
     */
}
```

```
* Note that a 32-bit bit-reversal is identical to four inplace
* 8-bit reversals followed by an end-for-end byteswap.
* In other words, the bytes of each bit are in the right order,
* but the bytes have been byteswapped. So we now do an explicit
* byteswap. On a little-endian machine, this byteswap and
* the final ntohl cancel out and could be elided.
*/

byte0 = result & 0xff;
byte1 = (result>>8) & 0xff;
byte2 = (result>>16) & 0xff;
byte3 = (result>>24) & 0xff;
crc32 = ((byte0 << 24) |
         (byte1 << 16) |
         (byte2 << 8)  |
         byte3);
return ( crc32 );
}

int
insert_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    unsigned long crc32;
    message = (SCTP_message *) buffer;
    message->common_header.checksum = 0L;
    crc32 = generate_crc32c(buffer,length);
    /* and insert it into the message */
    message->common_header.checksum = htonl(crc32);
    return 1;
}

int
validate_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    unsigned int i;
    unsigned long original_crc32;
    unsigned long crc32 = ~0L;

    /* save and zero checksum */
    message = (SCTP_message *) buffer;
    original_crc32 = ntohl(message->common_header.checksum);
    message->common_header.checksum = 0L;
    crc32 = generate_crc32c(buffer,length);
    return ((original_crc32 == crc32)? 1 : -1);
}
```

```

-----
New text: (Appendix B)
-----

/* Example of crc insertion */

#include "crc32cr.h"

uint32_t
generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    uint32_t crc32 = 0xffffffffUL;
    uint32_t result;
    uint8_t byte0, byte1, byte2, byte3;

    for (i = 0; i < length; i++) {
        CRC32C(crc32, buffer[i]);
    }

    result = ~crc32;

    /* result now holds the negated polynomial remainder;
     * since the table and algorithm is "reflected" [williams95].
     * That is, result has the same value as if we mapped the message
     * to a polynomial, computed the host-bit-order polynomial
     * remainder, performed final negation, then did an end-for-end
     * bit-reversal.
     * Note that a 32-bit bit-reversal is identical to four inplace
     * 8-bit reversals followed by an end-for-end byteswap.
     * In other words, the bits of each byte are in the right order,
     * but the bytes have been byteswapped. So we now do an explicit
     * byteswap. On a little-endian machine, this byteswap and
     * the final ntohl cancel out and could be elided.
     */

    byte0 = result & 0xff;
    byte1 = (result>>8) & 0xff;
    byte2 = (result>>16) & 0xff;
    byte3 = (result>>24) & 0xff;
    crc32 = ((byte0 << 24) |
              (byte1 << 16) |
              (byte2 << 8) |
              byte3);
    return (crc32);
}

int

```

```
insert_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    uint32_t crc32;
    message = (SCTP_message *) buffer;
    message->common_header.checksum = 0UL;
    crc32 = generate_crc32c(buffer, length);
    /* and insert it into the message */
    message->common_header.checksum = htonl(crc32);
    return 1;
}

int
validate_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    unsigned int i;
    uint32_t original_crc32;
    uint32_t crc32;

    /* save and zero checksum */
    message = (SCTP_message *)buffer;
    original_crc32 = ntohl(message->common_header.checksum);
    message->common_header.checksum = 0L;
    crc32 = generate_crc32c(buffer, length);
    return ((original_crc32 == crc32)? 1 : -1);
}
<CODE ENDS>
```

This text has been modified by multiple errata. It includes modifications from Section 3.5 and Section 3.10. It is in final form, and is not further updated in this document.

#### 3.46.3. Solution Description

The code was changed to use platform independent types.

#### 3.47. Clarification of Gap Ack Blocks in SACK Chunks

##### 3.47.1. Description of the Problem

The Gap Ack Blocks in the SACK chunk are intended to be isolated. However, this is not mentioned with normative text.

This issue was reported as part of an Errata for [RFC4960] with Errata ID 5202.

## 3.47.2. Text Changes to the Document

-----

Old text: (Section 3.3.4)

-----

The SACK also contains zero or more Gap Ack Blocks. Each Gap Ack Block acknowledges a subsequence of TSNs received following a break in the sequence of received TSNs. By definition, all TSNs acknowledged by Gap Ack Blocks are greater than the value of the Cumulative TSN Ack.

-----

New text: (Section 3.3.4)

-----

The SACK also contains zero or more Gap Ack Blocks. Each Gap Ack Block acknowledges a subsequence of TSNs received following a break in the sequence of received TSNs. The Gap Ack Blocks SHOULD be isolated. This means that the TSN just before each Gap Ack Block and the TSN just after each Gap Ack Block has not been received. By definition, all TSNs acknowledged by Gap Ack Blocks are greater than the value of the Cumulative TSN Ack.

This text is in final form, and is not further updated in this document.

-----  
Old text: (Section 3.3.4)  
-----

Gap Ack Blocks:

These fields contain the Gap Ack Blocks. They are repeated for each Gap Ack Block up to the number of Gap Ack Blocks defined in the Number of Gap Ack Blocks field. All DATA chunks with TSNs greater than or equal to (Cumulative TSN Ack + Gap Ack Block Start) and less than or equal to (Cumulative TSN Ack + Gap Ack Block End) of each Gap Ack Block are assumed to have been received correctly.

-----  
New text: (Section 3.3.4)  
-----

Gap Ack Blocks:

These fields contain the Gap Ack Blocks. They are repeated for each Gap Ack Block up to the number of Gap Ack Blocks defined in the Number of Gap Ack Blocks field. All DATA chunks with TSNs greater than or equal to (Cumulative TSN Ack + Gap Ack Block Start) and less than or equal to (Cumulative TSN Ack + Gap Ack Block End) of each Gap Ack Block are assumed to have been received correctly. Gap Ack Blocks SHOULD be isolated. That means that the DATA chunks with TSN equal to (Cumulative TSN Ack + Gap Ack Block Start - 1) and (Cumulative TSN Ack + Gap Ack Block End + 1) have not been received.

This text is in final form, and is not further updated in this document.

### 3.47.3. Solution Description

Normative text describing the intended usage of Gap Ack Blocks has been added.

### 3.48. Handling of SSN Wrap Arounds

#### 3.48.1. Description of the Problem

The Stream Sequence Number (SSN) is used for preserving the ordering of user messages within each SCTP stream. The SSN is limited to 16 bits. Therefore, multiple wrap arounds of the SSN might happen within the current send window. To allow the receiver to deliver



ordered user messages in the correct sequence, the sender should limit the number of user messages per stream.

#### 3.48.2. Text Changes to the Document

-----

Old text: (Section 6.1)

-----

Note: The data sender SHOULD NOT use a TSN that is more than  $2^{31} - 1$  above the beginning TSN of the current send window.

-----

New text: (Section 6.1)

-----

Note: The data sender SHOULD NOT use a TSN that is more than  $2^{31} - 1$  above the beginning TSN of the current send window.

Note: For each stream, the data sender SHOULD NOT have more than  $2^{16} - 1$  ordered user messages in the current send window.

This text is in final form, and is not further updated in this document.

#### 3.48.3. Solution Description

The data sender is required to limit the number of ordered user messages within the current send window.

#### 3.49. Update RFC 2119 Boilerplate

##### 3.49.1. Description of the Problem

The text to be used to refer to the [RFC2119] terms has been updated by [RFC8174].

##### 3.49.2. Text Changes to the Document

-----  
Old text: (Section 2)  
-----

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

-----  
New text: (Section 2)  
-----

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This text is in final form, and is not further updated in this document.

#### 3.49.3. Solution Description

The text has been updated to the one specified in [RFC8174].

#### 3.50. Missed Text Removal

##### 3.50.1. Description of the Problem

When integrating the changes to Section 7.2.4 of [RFC2960] as described in Section 2.8.2 of [RFC4460] some text was not removed and is therefore still in [RFC4960].

##### 3.50.2. Text Changes to the Document

-----  
Old text: (Section 7.2.4)  
-----

A straightforward implementation of the above keeps a counter for each TSN hole reported by a SACK. The counter increments for each consecutive SACK reporting the TSN hole. After reaching 3 and starting the Fast-Retransmit procedure, the counter resets to 0. Because cwnd in SCTP indirectly bounds the number of outstanding TSN's, the effect of TCP Fast Recovery is achieved automatically with no adjustment to the congestion control window size.

-----  
New text: (Section 7.2.4)  
-----

This text is in final form, and is not further updated in this document.

### 3.50.3. Solution Description

The text has finally been removed.

## 4. IANA Considerations

Section 3.44 of this document updates the port number registry for SCTP to be consistent with [RFC6335]. IANA is requested to review Section 3.44.

IANA is only requested to check if it is OK to make the proposed text change in an upcoming standards track document that updates [RFC4960]. IANA is not asked to perform any other action and this document does not request IANA to make a change to any registry.

## 5. Security Considerations

This document does not add any security considerations to those given in [RFC4960].

## 6. Acknowledgments

The authors wish to thank Pontus Andersson, Eric W. Biederman, Cedric Bonnet, Spencer Dawkins, Gorrry Fairhurst, Benjamin Kaduk, Mirja Kuehlewind, Peter Lei, Gyula Marosi, Lionel Morand, Jeff Morriss, Karen E. E. Nielsen, Tom Petch, Kacheong Poon, Julien Pourtet, Irene Ruengeler, Michael Welzl, and Qiaobing Xie for their invaluable comments.

## 7. References

### 7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.

### 7.2. Informative References

- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.
- [RFC1858] Ziemba, G., Reed, D., and P. Traina, "Security Considerations for IP Fragment Filtering", RFC 1858, DOI 10.17487/RFC1858, October 1995, <<https://www.rfc-editor.org/info/rfc1858>>.
- [RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, DOI 10.17487/RFC2960, October 2000, <<https://www.rfc-editor.org/info/rfc2960>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC4460] Stewart, R., Arias-Rodriguez, I., Poon, K., Caro, A., and M. Tuexen, "Stream Control Transmission Protocol (SCTP) Specification Errata and Issues", RFC 4460, DOI 10.17487/RFC4460, April 2006, <<https://www.rfc-editor.org/info/rfc4460>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/info/rfc5681>>.

- [RFC6096] Tuexen, M. and R. Stewart, "Stream Control Transmission Protocol (SCTP) Chunk Flags Registration", RFC 6096, DOI 10.17487/RFC6096, January 2011, <<https://www.rfc-editor.org/info/rfc6096>>.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <<https://www.rfc-editor.org/info/rfc6298>>.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <<https://www.rfc-editor.org/info/rfc6335>>.
- [RFC7053] Tuexen, M., Ruengeler, I., and R. Stewart, "SACK-IMMEDIATELY Extension for the Stream Control Transmission Protocol", RFC 7053, DOI 10.17487/RFC7053, November 2013, <<https://www.rfc-editor.org/info/rfc7053>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <<https://www.rfc-editor.org/info/rfc8311>>.

#### Authors' Addresses

Randall R. Stewart  
Netflix, Inc.  
Chapin, SC 29036  
United States

Email: [randall@lakerest.net](mailto:randall@lakerest.net)

Michael Tuexen  
Muenster University of Applied Sciences  
Stegerwaldstrasse 39  
48565 Steinfurt  
Germany

Email: [tuexen@fh-muenster.de](mailto:tuexen@fh-muenster.de)

Maksim Proshin  
Ericsson  
Kistavaegen 25  
Stockholm 164 80  
Sweden

Email: [mproshin@tieto.mera.ru](mailto:mproshin@tieto.mera.ru)

Transport Area Working Group  
Internet-Draft  
Updates: 6040, 2661, 2784, 3931, 4380,  
7450 (if approved)  
Intended status: Standards Track  
Expires: November 25, 2021

B. Briscoe  
Independent  
May 24, 2021

Propagating Explicit Congestion Notification Across IP Tunnel Headers  
Separated by a Shim  
draft-ietf-tsvwg-rfc6040update-shim-14

Abstract

RFC 6040 on "Tunnelling of Explicit Congestion Notification" made the rules for propagation of ECN consistent for all forms of IP in IP tunnel. This specification updates RFC 6040 to clarify that its scope includes tunnels where two IP headers are separated by at least one shim header that is not sufficient on its own for wide area packet forwarding. It surveys widely deployed IP tunnelling protocols that use such shim header(s) and updates the specifications of those that do not mention ECN propagation (L2TPv2, L2TPv3, GRE, Teredo and AMT). This specification also updates RFC 6040 with configuration requirements needed to make any legacy tunnel ingress safe.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 25, 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Terminology . . . . .	3
3. Scope of RFC 6040 . . . . .	3
3.1. Feasibility of ECN Propagation between Tunnel Headers . .	4
3.2. Desirability of ECN Propagation between Tunnel Headers .	5
4. Making a non-ECN Tunnel Ingress Safe by Configuration . . . .	5
5. ECN Propagation and Fragmentation/Reassembly . . . . .	7
6. IP-in-IP Tunnels with Tightly Coupled Shim Headers . . . . .	7
6.1. Specific Updates to Protocols under IETF Change Control .	10
6.1.1. L2TP (v2 and v3) ECN Extension . . . . .	10
6.1.2. GRE . . . . .	13
6.1.3. Teredo . . . . .	14
6.1.4. AMT . . . . .	15
7. IANA Considerations . . . . .	17
8. Security Considerations . . . . .	17
9. Comments Solicited . . . . .	17
10. Acknowledgements . . . . .	17
11. References . . . . .	18
11.1. Normative References . . . . .	18
11.2. Informative References . . . . .	19
Author's Address . . . . .	22

## 1. Introduction

RFC 6040 on "Tunnelling of Explicit Congestion Notification" [RFC6040] made the rules for propagation of Explicit Congestion Notification (ECN [RFC3168]) consistent for all forms of IP in IP tunnel.

A common pattern for many tunnelling protocols is to encapsulate an inner IP header (v4 or v6) with shim header(s) then an outer IP header (v4 or v6). Some of these shim headers are designed as generic encapsulations, so they do not necessarily directly encapsulate an inner IP header. Instead they can encapsulate headers such as link-layer (L2) protocols that in turn often encapsulate IP.



To clear up confusion, this specification clarifies that the scope of RFC 6040 includes any IP-in-IP tunnel, including those with shim header(s) and other encapsulations between the IP headers. Where necessary, it updates the specifications of the relevant encapsulation protocols with the specific text necessary to comply with RFC 6040.

This specification also updates RFC 6040 to state how operators ought to configure a legacy tunnel ingress to avoid unsafe system configurations.

## 2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119] when, and only when, they appear in all capitals, as shown here.

This specification uses the terminology defined in RFC 6040 [RFC6040].

## 3. Scope of RFC 6040

In section 1.1 of RFC 6040, its scope is defined as:

"...ECN field processing at encapsulation and decapsulation for any IP-in-IP tunnelling, whether IPsec or non-IPsec tunnels. It applies irrespective of whether IPv4 or IPv6 is used for either the inner or outer headers. ..."

This was intended to include cases where shim header(s) sit between the IP headers. Many tunnelling implementers have interpreted the scope of RFC 6040 as it was intended, but it is ambiguous. Therefore, this specification updates RFC 6040 by adding the following scoping text after the sentences quoted above:

It applies in cases where an outer IP header encapsulates an inner IP header either directly or indirectly by encapsulating other headers that in turn encapsulate (or might encapsulate) an inner IP header.

There is another problem with the scope of RFC 6040. Like many IETF specifications, RFC 6040 is written as a specification that implementations can choose to claim compliance with. This means it does not cover two important cases:

1. those cases where it is infeasible for an implementation to access an inner IP header when adding or removing an outer IP header;
2. those implementations that choose not to propagate ECN between IP headers.

However, the ECN field is a non-optional part of the IP header (v4 and v6). So any implementation that creates an outer IP header has to give the ECN field some value. There is only one safe value a tunnel ingress can use if it does not know whether the egress supports propagation of the ECN field; it has to clear the ECN field in any outer IP header to 0b00.

However, an RFC has no jurisdiction over implementations that choose not to comply with it or cannot comply with it, including all those implementations that pre-dated the RFC. Therefore it would have been unreasonable to add such a requirement to RFC 6040. Nonetheless, to ensure safe propagation of the ECN field over tunnels, it is reasonable to add requirements on operators, to ensure they configure their tunnels safely (where possible). Before stating these configuration requirements in Section 4, the factors that determine whether propagating ECN is feasible or desirable will be briefly introduced.

### 3.1. Feasibility of ECN Propagation between Tunnel Headers

In many cases shim header(s) and an outer IP header are always added to (or removed from) an inner IP packet as part of the same procedure. We call this a tightly coupled shim header. Processing the shim and outer together is often necessary because the shim(s) are not sufficient for packet forwarding in their own right; not unless complemented by an outer header. In these cases it will often be feasible for an implementation to propagate the ECN field between the IP headers.

In some cases a tunnel adds an outer IP header and a tightly coupled shim header to an inner header that is not an IP header, but that in turn encapsulates an IP header (or might encapsulate an IP header). For instance an inner Ethernet (or other link layer) header might encapsulate an inner IP header as its payload. We call this a tightly coupled shim over an encapsulating header.

Digging to arbitrary depths to find an inner IP header within an encapsulation is strictly a layering violation so it cannot be a required behaviour. Nonetheless, some tunnel endpoints already look within a L2 header for an IP header, for instance to map the Diffserv codepoint between an encapsulated IP header and an outer IP header

[RFC2983]. In such cases at least, it should be feasible to also (independently) propagate the ECN field between the same IP headers. Thus, access to the ECN field within an encapsulating header can be a useful and benign optimization. The guidelines in section 5 of [I-D.ietf-tsvwg-ecn-encap-guidelines] give the conditions for this layering violation to be benign.

### 3.2. Desirability of ECN Propagation between Tunnel Headers

Developers and network operators are encouraged to implement and deploy tunnel endpoints compliant with RFC 6040 (as updated by the present specification) in order to provide the benefits of wider ECN deployment [RFC8087]. Nonetheless, propagation of ECN between IP headers, whether separated by shim headers or not, has to be optional to implement and to use, because:

- o Legacy implementations of tunnels without any ECN support already exist
- o A network might be designed so that there is usually no bottleneck within the tunnel
- o If the tunnel endpoints would have to search within an L2 header to find an encapsulated IP header, it might not be worth the potential performance hit

### 4. Making a non-ECN Tunnel Ingress Safe by Configuration

Even when no specific attempt has been made to implement propagation of the ECN field at a tunnel ingress, it ought to be possible for the operator to render a tunnel ingress safe by configuration. The main safety concern is to disable (clear to zero) the ECN capability in the outer IP header at the ingress if the egress of the tunnel does not implement ECN logic to propagate any ECN markings into the packet forwarded beyond the tunnel. Otherwise the non-ECN egress could discard any ECN marking introduced within the tunnel, which would break all the ECN-based control loops that regulate the traffic load over the tunnel.

Therefore this specification updates RFC 6040 by inserting the following text at the end of section 4.3:

"

Whether or not an ingress implementation claims compliance with RFC 6040, RFC 4301 or RFC3168, when the outer tunnel header is IP (v4 or v6), if possible, the operator MUST configure the ingress to zero the outer ECN field in any of the following cases:

- \* if it is known that the tunnel egress does not support any of the RFCs that define propagation of the ECN field (RFC 6040, RFC 4301 or the full functionality mode of RFC 3168)
- \* or if the behaviour of the egress is not known or an egress with unknown behaviour might be dynamically paired with the ingress.
- \* or if an IP header might be encapsulated within a non-IP header that the tunnel ingress is encapsulating, but the ingress does not inspect within the encapsulation.

For the avoidance of doubt, the above only concerns the outer IP header. The ingress MUST NOT alter the ECN field of the arriving IP header that will become the inner IP header.

In order that the network operator can comply with the above safety rules, even if an implementation of a tunnel ingress does not claim to support RFC 6040, RFC 4301 or the full functionality mode of RFC 3168:

- \* it MUST NOT treat the former ToS octet (IPv4) or the former Traffic Class octet (IPv6) as a single 8-bit field, as the resulting linkage of ECN and Diffserv field propagation between inner and outer is not consistent with the definition of the 6-bit Diffserv field in [RFC2474] and [RFC3260];
- \* it SHOULD be able to be configured to zero the ECN field of the outer header.

"

For instance, if a tunnel ingress with no ECN-specific logic had a configuration capability to refer to the last 2 bits of the old ToS Byte of the outer (e.g. with a 0x3 mask) and set them to zero, while also being able to allow the DSCP to be re-mapped independently, that would be sufficient to satisfy both the above implementation requirements.

There might be concern that the above "MUST NOT" makes compliant implementations non-compliant at a stroke. However, by definition it solely applies to equipment that provides Diffserv configuration. Any such Diffserv equipment that is configuring treatment of the former ToS octet (IPv4) or the former Traffic Class octet (IPv6) as a single 8-bit field must have always been non-compliant with the definition of the 6-bit Diffserv field in [RFC2474] and [RFC3260]. If a tunnel ingress does not have any ECN logic, copying the ECN field as a side-effect of copying the DSCP is a seriously unsafe bug

that risks breaking the feedback loops that regulate load on a tunnel.

Zeroing the outer ECN field of all packets in all circumstances would be safe, but it would not be sufficient to claim compliance with RFC 6040 because it would not meet the aim of introducing ECN support to tunnels (see Section 4.3 of [RFC6040]).

## 5. ECN Propagation and Fragmentation/Reassembly

The following requirements update RFC6040, which omitted handling of the ECN field during fragmentation or reassembly. These changes might alter how many ECN-marked packets are propagated by a tunnel that fragments packets, but this would not raise any backward compatibility issues:

If a tunnel ingress fragments a packet, it MUST set the outer ECN field of all the fragments to the same value as it would have set if it had not fragmented the packet.

Section 5.3 of [RFC3168] specifies ECN requirements for reassembly of sets of outer fragments [I-D.ietf-intarea-tunnels] into packets. The following two additional requirements apply at a tunnel egress:

- o During reassembly of outer fragments [I-D.ietf-intarea-tunnels], if the ECN fields of the outer headers being reassembled into a single packet consist of a mixture of Not-ECT and other ECN codepoints, the packet MUST be discarded.
- o If there is mix of ECT(0) and ECT(1) fragments, then the reassembled packet MUST be set to either ECT(0) or ECT(1). In this case, reassembly SHOULD take into account that the RFC series has so far ensured that ECT(0) and ECT(1) can either be considered equivalent, or they can provide 2 levels of congestion severity, where the ranking of severity from highest to lowest is CE, ECT(1), ECT(0) [RFC6040].

## 6. IP-in-IP Tunnels with Tightly Coupled Shim Headers

There follows a list of specifications of encapsulations with tightly coupled shim header(s), in rough chronological order. The list is confined to standards track or widely deployed protocols. The list is not necessarily exhaustive so, for the avoidance of doubt, the scope of RFC 6040 is defined in Section 3 and is not limited to this list.

- o PPTP (Point-to-Point Tunneling Protocol) [RFC2637];

- o L2TP (Layer 2 Tunneling Protocol), specifically L2TPv2 [RFC2661] and L2TPv3 [RFC3931], which not only includes all the L2-specific specializations of L2TP, but also derivatives such as the Keyed IPv6 Tunnel [RFC8159];
- o GRE (Generic Routing Encapsulation) [RFC2784] and NVGRE (Network Virtualization using GRE) [RFC7637];
- o GTP (GPRS Tunneling Protocol), specifically GTPv1 [GTPv1], GTP v1 User Plane [GTPv1-U], GTP v2 Control Plane [GTPv2-C];
- o Teredo [RFC4380];
- o CAPWAP (Control And Provisioning of Wireless Access Points) [RFC5415];
- o LISP (Locator/Identifier Separation Protocol) [RFC6830];
- o AMT (Automatic Multicast Tunneling) [RFC7450];
- o VXLAN (Virtual eXtensible Local Area Network) [RFC7348] and VXLAN-GPE [I-D.ietf-nvo3-vxlan-gpe];
- o The Network Service Header (NSH [RFC8300]) for Service Function Chaining (SFC);
- o Geneve [RFC8926];
- o GUE (Generic UDP Encapsulation) [I-D.ietf-intarea-gue];
- o Direct tunnelling of an IP packet within a UDP/IP datagram (see Section 3.1.11 of [RFC8085]);
- o TCP Encapsulation of IKE and IPsec Packets (see Section 12.5 of [RFC8229]).

Some of the listed protocols enable encapsulation of a variety of network layer protocols as inner and/or outer. This specification applies in the cases where there is an inner and outer IP header as described in Section 3. Otherwise [I-D.ietf-tsvwg-ecn-encap-guidelines] gives guidance on how to design propagation of ECN into other protocols that might encapsulate IP.

Where protocols in the above list need to be updated to specify ECN propagation and they are under IETF change control, update text is given in the following subsections. For those not under IETF control, it is RECOMMENDED that implementations of encapsulation and decapsulation comply with RFC 6040. It is also RECOMMENDED that

their specifications are updated to add a requirement to comply with RFC 6040 (as updated by the present document).

PPTP is not under the change control of the IETF, but it has been documented in an informational RFC [RFC2637]. However, there is no need for the present specification to update PPTP because L2TP has been developed as a standardized replacement.

NVGRE is not under the change control of the IETF, but it has been documented in an informational RFC [RFC7637]. NVGRE is a specific use-case of GRE (it re-purposes the key field from the initial specification of GRE [RFC1701] as a Virtual Subnet ID). Therefore the text that updates GRE in Section 6.1.2 below is also intended to update NVGRE.

Although the definition of the various GTP shim headers is under the control of the 3GPP, it is hard to determine whether the 3GPP or the IETF controls standardization of the \_process\_ of adding both a GTP and an IP header to an inner IP header. Nonetheless, the present specification is provided so that the 3GPP can refer to it from any of its own specifications of GTP and IP header processing.

The specification of CAPWAP already specifies RFC 3168 ECN propagation and ECN capability negotiation. Without modification the CAPWAP specification already interworks with the backward compatible updates to RFC 3168 in RFC 6040.

LISP made the ECN propagation procedures in RFC 3168 mandatory from the start. RFC 3168 has since been updated by RFC 6040, but the changes are backwards compatible so there is still no need for LISP tunnel endpoints to negotiate their ECN capabilities.

VXLAN is not under the change control of the IETF but it has been documented in an informational RFC. In contrast, VXLAN-GPE (Generic Protocol Extension) is being documented under IETF change control. It is RECOMMENDED that VXLAN and VXLAN-GPE implementations comply with RFC 6040 when the VXLAN header is inserted between (or removed from between) IP headers. The authors of any future update to these specifications are encouraged to add a requirement to comply with RFC 6040 as updated by the present specification.

The Network Service Header (NSH [RFC8300]) has been defined as a shim-based encapsulation to identify the Service Function Path (SFP) in the Service Function Chaining (SFC) architecture [RFC7665]. A proposal has been made for the processing of ECN when handling transport encapsulation [I-D.ietf-sfc-nsh-ecn-support].

The specifications of Geneve and GUE already refer to RFC 6040 for ECN encapsulation.

Section 3.1.11 of RFC 8085 already explains that a tunnel that encapsulates an IP header within a UDP/IP datagram needs to follow RFC 6040 when propagating the ECN field between inner and outer IP headers. The requirements in Section 4 update RFC 6040, and hence implicitly update the UDP usage guidelines in RFC 8085 to add the important but previously unstated requirement that, if the UDP tunnel egress does not, or might not, support ECN propagation, a UDP tunnel ingress has to clear the outer IP ECN field to 0b00, e.g. by configuration.

Section 12.5 of TCP Encapsulation of IKE and IPsec Packets [RFC8229] already recommends the compatibility mode of RFC 6040 in this case, because there is not a one-to-one mapping between inner and outer packets.

## 6.1. Specific Updates to Protocols under IETF Change Control

### 6.1.1. L2TP (v2 and v3) ECN Extension

The L2TP terminology used here is defined in [RFC2661] and [RFC3931].

L2TPv3 [RFC3931] is used as a shim header between any packet-switched network (PSN) header (e.g. IPv4, IPv6, MPLS) and many types of layer 2 (L2) header. The L2TPv3 shim header encapsulates an L2-specific sub-layer then an L2 header that is likely to contain an inner IP header (v4 or v6). Then this whole stack of headers can be encapsulated optionally within an outer UDP header then an outer PSN header that is typically IP (v4 or v6).

L2TPv2 is used as a shim header between any PSN header and a PPP header, which is in turn likely to encapsulate an IP header.

Even though these shims are rather fat (particularly in the case of L2TPv3), they still fit the definition of a tightly coupled shim header over an encapsulating header (Section 3.1), because all the headers encapsulating the L2 header are added (or removed) together. L2TPv2 and L2TPv3 are therefore within the scope of RFC 6040, as updated by Section 3 above.

L2TP maintainers are RECOMMENDED to implement the ECN extension to L2TPv2 and L2TPv3 defined in Section 6.1.1.2 below, in order to provide the benefits of ECN [RFC8087], whenever a node within an L2TP tunnel becomes the bottleneck for an end-to-end traffic flow.



#### 6.1.1.1. Safe Configuration of a 'Non-ECN' Ingress LCCE

The following text is appended to both Section 5.3 of [RFC2661] and Section 4.5 of [RFC3931] as an update to the base L2TPv2 and L2TPv3 specifications:

The operator of an LCCE that does not support the ECN Extension in Section 6.1.1.2 of RFCXXXX MUST follow the configuration requirements in Section 4 of RFCXXXX to ensure it clears the outer IP ECN field to 0b00 when the outer PSN header is IP (v4 or v6). {RFCXXXX refers to the present document so it will need to be inserted by the RFC Editor}

In particular, for an LCCE implementation that does not support the ECN Extension, this means that configuration of how it propagates the ECN field between inner and outer IP headers MUST be independent of any configuration of the Diffserv extension of L2TP [RFC3308].

#### 6.1.1.2. ECN Extension for L2TP (v2 or v3)

When the outer PSN header and the payload inside the L2 header are both IP (v4 or v6), to comply with RFC 6040, an LCCE will follow the rules for propagation of the ECN field at ingress and egress in Section 4 of RFC 6040 [RFC6040].

Before encapsulating any data packets, RFC 6040 requires an ingress LCCE to check that the egress LCCE supports ECN propagation as defined in RFC 6040 or one of its compatible predecessors ([RFC4301] or the full functionality mode of [RFC3168]). If the egress supports ECN propagation, the ingress LCCE can use the normal mode of encapsulation (copying the ECN field from inner to outer). Otherwise, the ingress LCCE has to use compatibility mode [RFC6040] (clearing the outer IP ECN field to 0b00).

An LCCE can determine the remote LCCE's support for ECN either statically (by configuration) or by dynamic discovery during setup of each control connection between the LCCEs, using the Capability AVP defined in Section 6.1.1.2.1 below.

Where the outer PSN header is some protocol other than IP that supports ECN, the appropriate ECN propagation specification will need to be followed, e.g. "Explicit Congestion Marking in MPLS" [RFC5129]. Where no specification exists for ECN propagation by a particular PSN, [I-D.ietf-tsvwg-ecn-encap-guidelines] gives general guidance on how to design ECN propagation into a protocol that encapsulates IP.

## 6.1.1.2.1. LCCE Capability AVP for ECN Capability Negotiation

The LCCE Capability Attribute-Value Pair (AVP) defined here has Attribute Type ZZ. The Attribute Value field for this AVP is a bit-mask with the following 16-bit format:

```

      0                               1
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|X X X X X X X X X X X X X X X E|
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

```

Figure 1: Value Field for the LCCE Capability Attribute

This AVP MAY be present in the following message types: SCCRQ and SCCRP (Start-Control-Connection-Request and Start-Control-Connection-Reply). This AVP MAY be hidden (the H-bit set to 0 or 1) and is optional (M-bit not set). The length (before hiding) of this AVP MUST be 8 octets. The Vendor ID is the IETF Vendor ID of 0.

Bit 15 of the Value field of the LCCE Capability AVP is defined as the ECN Capability flag (E). When the ECN Capability flag is set to 1, it indicates that the sender supports ECN propagation. When the ECN Capability flag is cleared to zero, or when no LCCE Capability AVP is present, it indicates that the sender does not support ECN propagation. All the other bits are reserved. They MUST be cleared to zero when sent and ignored when received or forwarded.

An LCCE initiating a control connection will send a Start-Control-Connection-Request (SCCRQ) containing an LCCE Capability AVP with the ECN Capability flag set to 1. If the tunnel terminator supports ECN, it will return a Start-Control-Connection-Reply (SCCRP) that also includes an LCCE Capability AVP with the ECN Capability flag set to 1. Then, for any sessions created by that control connection, both ends of the tunnel can use the normal mode of RFC 6040, i.e. it can copy the IP ECN field from inner to outer when encapsulating data packets.

If, on the other hand, the tunnel terminator does not support ECN it will ignore the ECN flag in the LCCE Capability AVP and send an SCCRP to the tunnel initiator without a Capability AVP (or with a Capability AVP but with the ECN Capability flag cleared to zero). The tunnel initiator interprets the absence of the ECN Capability flag in the SCCRP as an indication that the tunnel terminator is incapable of supporting ECN. When encapsulating data packets for any sessions created by that control connection, the tunnel initiator will then use the compatibility mode of RFC 6040 to clear the ECN field of the outer IP header to 0b00.

If the tunnel terminator does not support this ECN extension, the network operator is still expected to configure it to comply with the safety provisions set out in Section 6.1.1.1 above, when it acts as an ingress LCCE.

#### 6.1.2. GRE

The GRE terminology used here is defined in [RFC2784]. GRE is often used as a tightly coupled shim header between IP headers. Sometimes the GRE shim header encapsulates an L2 header, which might in turn encapsulate an IP header. Therefore GRE is within the scope of RFC 6040 as updated by Section 3 above.

GRE tunnel endpoint maintainers are RECOMMENDED to support [RFC6040] as updated by the present specification, in order to provide the benefits of ECN [RFC8087] whenever a node within a GRE tunnel becomes the bottleneck for an end-to-end IP traffic flow tunnelled over GRE using IP as the delivery protocol (outer header).

GRE itself does not support dynamic set-up and configuration of tunnels. However, control plane protocols such as Mobile IPv4 (MIP4) [RFC5944], Mobile IPv6 (MIP6) [RFC6275], Proxy Mobile IP (PMIP) [RFC5845] and IKEv2 [RFC7296] are sometimes used to set up GRE tunnels dynamically.

When these control protocols set up IP-in-IP or IPSec tunnels, it is likely that they propagate the ECN field as defined in RFC 6040 or one of its compatible predecessors (RFC 4301 or the full functionality mode of RFC 3168). However, if they use a GRE encapsulation, this presumption is less sound.

Therefore, If the outer delivery protocol is IP (v4 or v6) the operator is obliged to follow the safe configuration requirements in Section 4 above. Section 6.1.2.1 below updates the base GRE specification with this requirement, to emphasize its importance.

Where the delivery protocol is some protocol other than IP that supports ECN, the appropriate ECN propagation specification will need to be followed, e.g Explicit Congestion Marking in MPLS [RFC5129]. Where no specification exists for ECN propagation by a particular PSN, [I-D.ietf-tsvwg-ecn-encap-guidelines] gives more general guidance on how to propagate ECN to and from protocols that encapsulate IP.

#### 6.1.2.1. Safe Configuration of a 'Non-ECN' GRE Ingress

The following text is appended to Section 3 of [RFC2784] as an update to the base GRE specification:

The operator of a GRE tunnel ingress MUST follow the configuration requirements in Section 4 of RFCXXXX when the outer delivery protocol is IP (v4 or v6). {RFCXXXX refers to the present document so it will need to be inserted by the RFC Editor}

#### 6.1.3. Teredo

Teredo [RFC4380] provides a way to tunnel IPv6 over an IPv4 network, with a UDP-based shim header between the two.

For Teredo tunnel endpoints to provide the benefits of ECN, the Teredo specification would have to be updated to include negotiation of the ECN capability between Teredo tunnel endpoints. Otherwise it would be unsafe for a Teredo tunnel ingress to copy the ECN field to the IPv6 outer.

It is believed that current implementations do not support propagation of ECN, but that they do safely zero the ECN field in the outer IPv6 header. However the specification does not mention anything about this.

To make existing Teredo deployments safe, it would be possible to add ECN capability negotiation to those that are subject to remote OS update. However, for those implementations not subject to remote OS update, it will not be feasible to require them to be configured correctly, because Teredo tunnel endpoints are generally deployed on hosts.

Therefore, until ECN support is added to the specification of Teredo, the only feasible further safety precaution available here is to update the specification of Teredo implementations with the following text, as a new section 5.1.3:

##### "5.1.3 Safe 'Non-ECN' Teredo Encapsulation

A Teredo tunnel ingress implementation that does not support ECN propagation as defined in RFC 6040 or one of its compatible predecessors (RFC 4301 or the full functionality mode of RFC 3168) MUST zero the ECN field in the outer IPv6 header."

#### 6.1.4. AMT

Automatic Multicast Tunneling (AMT [RFC7450]) is a tightly coupled shim header that encapsulates an IP packet and is itself encapsulated within a UDP/IP datagram. Therefore AMT is within the scope of RFC 6040 as updated by Section 3 above.

AMT tunnel endpoint maintainers are RECOMMENDED to support [RFC6040] as updated by the present specification, in order to provide the benefits of ECN [RFC8087] whenever a node within an AMT tunnel becomes the bottleneck for an IP traffic flow tunnelled over AMT.

To comply with RFC 6040, an AMT relay and gateway will follow the rules for propagation of the ECN field at ingress and egress respectively, as described in Section 4 of RFC 6040 [RFC6040].

Before encapsulating any data packets, RFC 6040 requires an ingress AMT relay to check that the egress AMT gateway supports ECN propagation as defined in RFC 6040 or one of its compatible predecessors (RFC 4301 or the full functionality mode of RFC 3168). If the egress gateway supports ECN, the ingress relay can use the normal mode of encapsulation (copying the IP ECN field from inner to outer). Otherwise, the ingress relay has to use compatibility mode, which means it has to clear the outer ECN field to zero [RFC6040].

An AMT tunnel is created dynamically (not manually), so the relay will need to determine the remote gateway's support for ECN using the ECN capability declaration defined in Section 6.1.4.2 below.

##### 6.1.4.1. Safe Configuration of a 'Non-ECN' Ingress AMT Relay

The following text is appended to Section 4.2.2 of [RFC7450] as an update to the AMT specification:

The operator of an AMT relay that does not support RFC 6040 or one of its compatible predecessors (RFC 4301 or the full functionality mode of RFC 3168) MUST follow the configuration requirements in Section 4 of RFCXXXX to ensure it clears the outer IP ECN field to zero. {RFCXXXX refers to the present document so it will need to be inserted by the RFC Editor}

##### 6.1.4.2. ECN Capability Declaration of an AMT Gateway

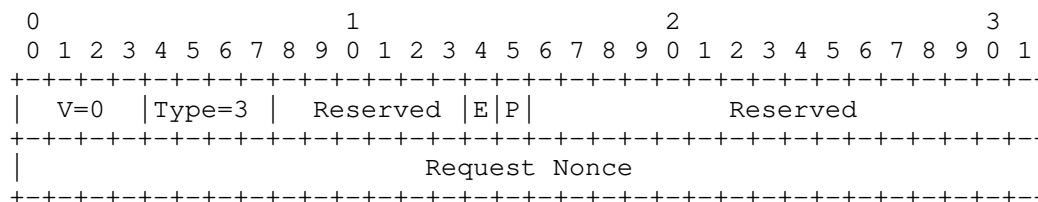


Figure 2: Updated AMT Request Message Format

Bit 14 of the AMT Request Message counting from 0 (or bit 7 of the Reserved field counting from 1) is defined here as the AMT Gateway ECN Capability flag (E), as shown in Figure 2. The definitions of all other fields in the AMT Request Message are unchanged from RFC 7450.

When the E flag is set to 1, it indicates that the sender of the message supports RFC 6040 ECN propagation. When it is cleared to zero, it indicates the sender of the message does not support RFC 6040 ECN propagation. An AMT gateway "that supports RFC 6040 ECN propagation" means one that propagates the ECN field to the forwarded data packet based on the combination of arriving inner and outer ECN fields, as defined in Section 4 of RFC 6040.

The other bits of the Reserved field remain reserved. They will continue to be cleared to zero when sent and ignored when either received or forwarded, as specified in Section 5.1.3.3. of RFC 7450.

An AMT gateway that does not support RFC 6040 MUST NOT set the E flag of its Request Message to 1.

An AMT gateway that supports RFC 6040 ECN propagation MUST set the E flag of its Relay Discovery Message to 1.

The action of the corresponding AMT relay that receives a Request message with the E flag set to 1 depends on whether the relay itself supports RFC 6040 ECN propagation:

- o If the relay supports RFC 6040 ECN propagation, it will store the ECN capability of the gateway along with its address. Then whenever it tunnels datagrams towards this gateway, it MUST use the normal mode of RFC 6040 to propagate the ECN field when encapsulating datagrams (i.e. it copies the IP ECN field from inner to outer).

- o If the discovered AMT relay does not support RFC 6040 ECN propagation, it will ignore the E flag in the Reserved field, as per section 5.1.3.3. of RFC 7450.

If the AMT relay does not support RFC 6040 ECN propagation, the network operator is still expected to configure it to comply with the safety provisions set out in Section 6.1.4.1 above.

## 7. IANA Considerations

IANA is requested to assign the following L2TP Control Message Attribute Value Pair:

Attribute Type	Description	Reference
ZZ	ECN Capability	RFCXXXX

[TO BE REMOVED: This registration should take place at the following location: <https://www.iana.org/assignments/l2tp-parameters/l2tp-parameters.xhtml> ]

## 8. Security Considerations

The Security Considerations in [RFC6040] and [I-D.ietf-tsvwg-ecn-encap-guidelines] apply equally to the scope defined for the present specification.

## 9. Comments Solicited

Comments and questions are encouraged and very welcome. They can be addressed to the IETF Transport Area working group mailing list <tsvwg@ietf.org>, and/or to the authors.

## 10. Acknowledgements

Thanks to Ing-jyh (Inton) Tsang for initial discussions on the need for ECN propagation in L2TP and its applicability. Thanks also to Carlos Pignataro, Tom Herbert, Ignacio Goyret, Alia Atlas, Praveen Balasubramanian, Joe Touch, Mohamed Boucadair, David Black, Jake Holland and Sri Gundavelli for helpful advice and comments. "A Comparison of IPv6-over-IPv4 Tunnel Mechanisms" [RFC7059] helped to identify a number of tunnelling protocols to include within the scope of this document.

Bob Briscoe was part-funded by the Research Council of Norway through the TimeIn project. The views expressed here are solely those of the authors.

## 11. References

### 11.1. Normative References

- [I-D.ietf-tsvwg-ecn-encap-guidelines]  
Briscoe, B. and J. Kaippallimalil, "Guidelines for Adding Congestion Notification to Protocols that Encapsulate IP", draft-ietf-tsvwg-ecn-encap-guidelines-15 (work in progress), March 2021.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <<https://www.rfc-editor.org/info/rfc2474>>.
- [RFC2661] Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G., and B. Palter, "Layer Two Tunneling Protocol "L2TP"", RFC 2661, DOI 10.17487/RFC2661, August 1999, <<https://www.rfc-editor.org/info/rfc2661>>.
- [RFC2784] Farinacci, D., Li, T., Hanks, S., Meyer, D., and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 2784, DOI 10.17487/RFC2784, March 2000, <<https://www.rfc-editor.org/info/rfc2784>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC3931] Lau, J., Ed., Townsley, M., Ed., and I. Goyret, Ed., "Layer Two Tunneling Protocol - Version 3 (L2TPv3)", RFC 3931, DOI 10.17487/RFC3931, March 2005, <<https://www.rfc-editor.org/info/rfc3931>>.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<https://www.rfc-editor.org/info/rfc4301>>.



- [RFC4380] Huitema, C., "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)", RFC 4380, DOI 10.17487/RFC4380, February 2006, <<https://www.rfc-editor.org/info/rfc4380>>.
- [RFC5129] Davie, B., Briscoe, B., and J. Tay, "Explicit Congestion Marking in MPLS", RFC 5129, DOI 10.17487/RFC5129, January 2008, <<https://www.rfc-editor.org/info/rfc5129>>.
- [RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <<https://www.rfc-editor.org/info/rfc6040>>.

## 11.2. Informative References

- [GTPv1] 3GPP, "GPRS Tunnelling Protocol (GTP) across the Gn and Gp interface", Technical Specification TS 29.060.
- [GTPv1-U] 3GPP, "General Packet Radio System (GPRS) Tunnelling Protocol User Plane (GTPv1-U)", Technical Specification TS 29.281.
- [GTPv2-C] 3GPP, "Evolved General Packet Radio Service (GPRS) Tunnelling Protocol for Control plane (GTPv2-C)", Technical Specification TS 29.274.
- [I-D.ietf-intarea-gue]  
Herbert, T., Yong, L., and O. Zia, "Generic UDP Encapsulation", draft-ietf-intarea-gue-09 (work in progress), October 2019.
- [I-D.ietf-intarea-tunnels]  
Touch, J. and M. Townsley, "IP Tunnels in the Internet Architecture", draft-ietf-intarea-tunnels-10 (work in progress), September 2019.
- [I-D.ietf-nvo3-vxlan-gpe]  
(Editor), F. M., (editor), L. K., and U. E. (editor), "Generic Protocol Extension for VXLAN (VXLAN-GPE)", draft-ietf-nvo3-vxlan-gpe-11 (work in progress), March 2021.
- [I-D.ietf-sfc-nsh-ecn-support]  
Eastlake, D. E., Briscoe, B., Li, Y., Malis, A. G., and X. Wei, "Explicit Congestion Notification (ECN) and Congestion Feedback Using the Network Service Header (NSH) and IPFIX", draft-ietf-sfc-nsh-ecn-support-05 (work in progress), April 2021.

- [RFC1701] Hanks, S., Li, T., Farinacci, D., and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 1701, DOI 10.17487/RFC1701, October 1994, <<https://www.rfc-editor.org/info/rfc1701>>.
- [RFC2637] Hamzeh, K., Pall, G., Verthein, W., Taarud, J., Little, W., and G. Zorn, "Point-to-Point Tunneling Protocol (PPTP)", RFC 2637, DOI 10.17487/RFC2637, July 1999, <<https://www.rfc-editor.org/info/rfc2637>>.
- [RFC2983] Black, D., "Differentiated Services and Tunnels", RFC 2983, DOI 10.17487/RFC2983, October 2000, <<https://www.rfc-editor.org/info/rfc2983>>.
- [RFC3260] Grossman, D., "New Terminology and Clarifications for Diffserv", RFC 3260, DOI 10.17487/RFC3260, April 2002, <<https://www.rfc-editor.org/info/rfc3260>>.
- [RFC3308] Calhoun, P., Luo, W., McPherson, D., and K. Peirce, "Layer Two Tunneling Protocol (L2TP) Differentiated Services Extension", RFC 3308, DOI 10.17487/RFC3308, November 2002, <<https://www.rfc-editor.org/info/rfc3308>>.
- [RFC5415] Calhoun, P., Ed., Montemurro, M., Ed., and D. Stanley, Ed., "Control And Provisioning of Wireless Access Points (CAPWAP) Protocol Specification", RFC 5415, DOI 10.17487/RFC5415, March 2009, <<https://www.rfc-editor.org/info/rfc5415>>.
- [RFC5845] Muhanna, A., Khalil, M., Gundavelli, S., and K. Leung, "Generic Routing Encapsulation (GRE) Key Option for Proxy Mobile IPv6", RFC 5845, DOI 10.17487/RFC5845, June 2010, <<https://www.rfc-editor.org/info/rfc5845>>.
- [RFC5944] Perkins, C., Ed., "IP Mobility Support for IPv4, Revised", RFC 5944, DOI 10.17487/RFC5944, November 2010, <<https://www.rfc-editor.org/info/rfc5944>>.
- [RFC6275] Perkins, C., Ed., Johnson, D., and J. Arkko, "Mobility Support in IPv6", RFC 6275, DOI 10.17487/RFC6275, July 2011, <<https://www.rfc-editor.org/info/rfc6275>>.
- [RFC6830] Farinacci, D., Fuller, V., Meyer, D., and D. Lewis, "The Locator/ID Separation Protocol (LISP)", RFC 6830, DOI 10.17487/RFC6830, January 2013, <<https://www.rfc-editor.org/info/rfc6830>>.

- [RFC7059] Steffann, S., van Beijnum, I., and R. van Rein, "A Comparison of IPv6-over-IPv4 Tunnel Mechanisms", RFC 7059, DOI 10.17487/RFC7059, November 2013, <<https://www.rfc-editor.org/info/rfc7059>>.
- [RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <<https://www.rfc-editor.org/info/rfc7296>>.
- [RFC7348] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", RFC 7348, DOI 10.17487/RFC7348, August 2014, <<https://www.rfc-editor.org/info/rfc7348>>.
- [RFC7450] Bumgardner, G., "Automatic Multicast Tunneling", RFC 7450, DOI 10.17487/RFC7450, February 2015, <<https://www.rfc-editor.org/info/rfc7450>>.
- [RFC7637] Garg, P., Ed. and Y. Wang, Ed., "NVGRE: Network Virtualization Using Generic Routing Encapsulation", RFC 7637, DOI 10.17487/RFC7637, September 2015, <<https://www.rfc-editor.org/info/rfc7637>>.
- [RFC7665] Halpern, J., Ed. and C. Pignataro, Ed., "Service Function Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <<https://www.rfc-editor.org/info/rfc7665>>.
- [RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <<https://www.rfc-editor.org/info/rfc8085>>.
- [RFC8087] Fairhurst, G. and M. Welzl, "The Benefits of Using Explicit Congestion Notification (ECN)", RFC 8087, DOI 10.17487/RFC8087, March 2017, <<https://www.rfc-editor.org/info/rfc8087>>.
- [RFC8159] Konstantynowicz, M., Ed., Heron, G., Ed., Schatzmayr, R., and W. Henderickx, "Keyed IPv6 Tunnel", RFC 8159, DOI 10.17487/RFC8159, May 2017, <<https://www.rfc-editor.org/info/rfc8159>>.
- [RFC8229] Pauly, T., Touati, S., and R. Mantha, "TCP Encapsulation of IKE and IPsec Packets", RFC 8229, DOI 10.17487/RFC8229, August 2017, <<https://www.rfc-editor.org/info/rfc8229>>.

- [RFC8300] Quinn, P., Ed., Elzur, U., Ed., and C. Pignataro, Ed.,  
"Network Service Header (NSH)", RFC 8300,  
DOI 10.17487/RFC8300, January 2018,  
<<https://www.rfc-editor.org/info/rfc8300>>.
- [RFC8926] Gross, J., Ed., Ganga, I., Ed., and T. Sridhar, Ed.,  
"Geneve: Generic Network Virtualization Encapsulation",  
RFC 8926, DOI 10.17487/RFC8926, November 2020,  
<<https://www.rfc-editor.org/info/rfc8926>>.

## Author's Address

Bob Briscoe  
Independent  
UK

EMail: [ietf@bobbriscoe.net](mailto:ietf@bobbriscoe.net)  
URI: <http://bobbriscoe.net/>

TSVWG  
Internet-Draft  
Intended status: Standards Track  
Expires: December 20, 2019

V. Roca  
B. Teibi  
INRIA  
June 18, 2019

Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC)  
Schemes for FECFRAME  
draft-ietf-tsvwg-rlc-fec-scheme-16

Abstract

This document describes two fully-specified Forward Erasure Correction (FEC) Schemes for Sliding Window Random Linear Codes (RLC), one for RLC over the Galois Field (A.K.A. Finite Field)  $GF(2)$ , a second one for RLC over the Galois Field  $GF(2^{8})$ , each time with the possibility of controlling the code density. They can protect arbitrary media streams along the lines defined by FECFRAME extended to sliding window FEC codes. These sliding window FEC codes rely on an encoding window that slides over the source symbols, generating new repair symbols whenever needed. Compared to block FEC codes, these sliding window FEC codes offer key advantages with real-time flows in terms of reduced FEC-related latency while often providing improved packet erasure recovery capabilities.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 20, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1.	Introduction . . . . .	3
1.1.	Limits of Block Codes with Real-Time Flows . . . . .	4
1.2.	Lower Latency and Better Protection of Real-Time Flows with the Sliding Window RLC Codes . . . . .	4
1.3.	Small Transmission Overheads with the Sliding Window RLC FEC Scheme . . . . .	5
1.4.	Document Organization . . . . .	6
2.	Definitions and Abbreviations . . . . .	6
3.	Common Procedures . . . . .	7
3.1.	Codec Parameters . . . . .	7
3.2.	ADU, ADUI and Source Symbols Mappings . . . . .	9
3.3.	Encoding Window Management . . . . .	10
3.4.	Source Symbol Identification . . . . .	11
3.5.	Pseudo-Random Number Generator (PRNG) . . . . .	11
3.6.	Coding Coefficients Generation Function . . . . .	13
3.7.	Finite Fields Operations . . . . .	15
3.7.1.	Finite Field Definitions . . . . .	15
3.7.2.	Linear Combination of Source Symbols Computation . . . . .	15
4.	Sliding Window RLC FEC Scheme over $GF(2^{^8})$ for Arbitrary Packet Flows . . . . .	16
4.1.	Formats and Codes . . . . .	16
4.1.1.	FEC Framework Configuration Information . . . . .	16
4.1.2.	Explicit Source FEC Payload ID . . . . .	18
4.1.3.	Repair FEC Payload ID . . . . .	18
4.2.	Procedures . . . . .	20
5.	Sliding Window RLC FEC Scheme over $GF(2)$ for Arbitrary Packet Flows . . . . .	20
5.1.	Formats and Codes . . . . .	20
5.1.1.	FEC Framework Configuration Information . . . . .	20
5.1.2.	Explicit Source FEC Payload ID . . . . .	20
5.1.3.	Repair FEC Payload ID . . . . .	20
5.2.	Procedures . . . . .	21
6.	FEC Code Specification . . . . .	21
6.1.	Encoding Side . . . . .	21
6.2.	Decoding Side . . . . .	22
7.	Implementation Status . . . . .	22

8.	Security Considerations . . . . .	23
8.1.	Attacks Against the Data Flow . . . . .	23
8.1.1.	Access to Confidential Content . . . . .	23
8.1.2.	Content Corruption . . . . .	23
8.2.	Attacks Against the FEC Parameters . . . . .	23
8.3.	When Several Source Flows are to be Protected Together .	25
8.4.	Baseline Secure FEC Framework Operation . . . . .	25
8.5.	Additional Security Considerations for Numerical Computations . . . . .	25
9.	Operations and Management Considerations . . . . .	26
9.1.	Operational Recommendations: Finite Field GF(2) Versus GF(2 <sup>8</sup> ) . . . . .	26
9.2.	Operational Recommendations: Coding Coefficients Density Threshold . . . . .	26
10.	IANA Considerations . . . . .	27
11.	Acknowledgments . . . . .	27
12.	References . . . . .	27
12.1.	Normative References . . . . .	27
12.2.	Informative References . . . . .	28
Appendix A.	TinyMT32 Validation Criteria (Normative) . . . . .	30
Appendix B.	Assessing the PRNG Adequacy (Informational) . . . . .	31
Appendix C.	Possible Parameter Derivation (Informational) . . . . .	33
C.1.	Case of a CBR Real-Time Flow . . . . .	34
C.2.	Other Types of Real-Time Flow . . . . .	36
C.3.	Case of a Non Real-Time Flow . . . . .	37
Appendix D.	Decoding Beyond Maximum Latency Optimization (Informational) . . . . .	37
Authors' Addresses	. . . . .	38

## 1. Introduction

Application-Level Forward Erasure Correction (AL-FEC) codes, or simply FEC codes, are a key element of communication systems. They are used to recover from packet losses (or erasures) during content delivery sessions to a potentially large number of receivers (multicast/broadcast transmissions). This is the case with the FLUTE/ALC protocol [RFC6726] when used for reliable file transfers over lossy networks, and the FECFRAME protocol [RFC6363] when used for reliable continuous media transfers over lossy networks.

The present document only focuses on the FECFRAME protocol, used in multicast/broadcast delivery mode, in particular for contents that feature stringent real-time constraints: each source packet has a maximum validity period after which it will not be considered by the destination application.

### 1.1. Limits of Block Codes with Real-Time Flows

With FECFRAME, there is a single FEC encoding point (either an end-host/server (source) or a middlebox) and a single FEC decoding point per receiver (either an end-host (receiver) or middlebox). In this context, currently standardized AL-FEC codes for FECFRAME like Reed-Solomon [RFC6865], LDPC-Staircase [RFC6816], or Raptor/RaptorQ [RFC6681], are all linear block codes: they require the data flow to be segmented into blocks of a predefined maximum size.

To define this block size, it is required to find an appropriate balance between robustness and decoding latency: the larger the block size, the higher the robustness (e.g., in case of long packet erasure bursts), but also the higher the maximum decoding latency (i.e., the maximum time required to recover a lost (erased) packet thanks to FEC protection). Therefore, with a multicast/broadcast session where different receivers experience different packet loss rates, the block size should be chosen by considering the worst communication conditions one wants to support, but without exceeding the desired maximum decoding latency. This choice then impacts the FEC-related latency of all receivers, even those experiencing a good communication quality, since no FEC encoding can happen until all the source data of the block is available at the sender, which directly depends on the block size.

### 1.2. Lower Latency and Better Protection of Real-Time Flows with the Sliding Window RLC Codes

This document introduces two fully-specified FEC Schemes that do not follow the block code approach: the Sliding Window Random Linear Codes (RLC) over either Galois Fields (A.K.A. Finite Fields)  $GF(2)$  (the "binary case") or  $GF(2^{255})$ , each time with the possibility of controlling the code density. These FEC Schemes are used to protect arbitrary media streams along the lines defined by FECFRAME extended to sliding window FEC codes [fecframe-ext]. These FEC Schemes, and more generally Sliding Window FEC codes, are recommended for instance, with media that feature real-time constraints sent within a multicast/broadcast session [Roca17].

The RLC codes belong to the broad class of sliding-window AL-FEC codes (A.K.A. convolutional codes) [RFC8406]. The encoding process is based on an encoding window that slides over the set of source packets (in fact source symbols as we will see in Section 3.2), this window being either of fixed size or variable size (A.K.A. an elastic window). Repair symbols are generated on-the-fly, by computing a random linear combination of the source symbols present in the current encoding window, and passed to the transport layer.



At the receiver, a linear system is managed from the set of received source and repair packets. New variables (representing source symbols) and equations (representing the linear combination carried by each repair symbol received) are added upon receiving new packets. Variables and the equations they are involved in are removed when they are too old with respect to their validity period (real-time constraints). Lost source symbols are then recovered thanks to this linear system whenever its rank permits to solve it (at least partially).

The protection of any multicast/broadcast session needs to be dimensioned by considering the worst communication conditions one wants to support. This is also true with RLC (more generally any sliding window) code. However, the receivers experiencing a good to medium communication quality will observe a reduced FEC-related latency compared to block codes [Roca17] since an isolated lost source packet is quickly recovered with the following repair packet. On the opposite, with a block code, recovering an isolated lost source packet always requires waiting for the first repair packet to arrive after the end of the block. Additionally, under certain situations (e.g., with a limited FEC-related latency budget and with constant bitrate transmissions after FECFRAME encoding), sliding window codes can more efficiently achieve a target transmission quality (e.g., measured by the residual loss after FEC decoding) by sending fewer repair packets (i.e., higher code rate) than block codes.

### 1.3. Small Transmission Overheads with the Sliding Window RLC FEC Scheme

The Sliding Window RLC FEC Scheme is designed to limit the packet header overhead. The main requirement is that each repair packet header must enable a receiver to reconstruct the set of source symbols plus the associated coefficients used during the encoding process. In order to minimize packet overhead, the set of source symbols in the encoding window as well as the set of coefficients over  $GF(2^m)$  (where  $m$  is 1 or 8, depending on the FEC Scheme) used in the linear combination are not individually listed in the repair packet header. Instead, each FEC Repair Packet header contains:

- o the Encoding Symbol Identifier (ESI) of the first source symbol in the encoding window as well as the number of symbols (since this number may vary with a variable size, elastic window). These two pieces of information enable each receiver to reconstruct the set of source symbols considered during encoding, the only constraint being that there cannot be any gap;
- o the seed and density threshold parameters used by a coding coefficients generation function (Section 3.6). These two pieces

of information enable each receiver to generate the same set of coding coefficients over  $GF(2^m)$  as the sender;

Therefore, no matter the number of source symbols present in the encoding window, each FEC Repair Packet features a fixed 64-bit long header, called Repair FEC Payload ID (Figure 8). Similarly, each FEC Source Packet features a fixed 32-bit long trailer, called Explicit Source FEC Payload ID (Figure 6), that contains the ESI of the first source symbol (Section 3.2).

#### 1.4. Document Organization

This fully-specified FEC Scheme follows the structure required by [RFC6363], section 5.6. "FEC Scheme Requirements", namely:

3. Procedures: This section describes procedures specific to this FEC Scheme, namely: RLC parameters derivation, ADUI and source symbols mapping, pseudo-random number generator, and coding coefficients generation function;
4. Formats and Codes: This section defines the Source FEC Payload ID and Repair FEC Payload ID formats, carrying the signaling information associated to each source or repair symbol. It also defines the FEC Framework Configuration Information (FFCI) carrying signaling information for the session;
5. FEC Code Specification: Finally this section provides the code specification.

#### 2. Definitions and Abbreviations

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses the following definitions and abbreviations:

$a^b$   $a$  to the power of  $b$

$GF(q)$  denotes a finite field (also known as the Galois Field) with  $q$  elements. We assume that  $q = 2^m$  in this document

$m$  defines the length of the elements in the finite field, in bits.

In this document,  $m$  is equal to 1 or 8

ADU: Application Data Unit

ADUI: Application Data Unit Information (includes the F, L and padding fields in addition to the ADU)

E: size of an encoding symbol (i.e., source or repair symbol), assumed fixed (in bytes)

br\_in: transmission bitrate at the input of the FECFRAME sender, assumed fixed (in bits/s)  
br\_out: transmission bitrate at the output of the FECFRAME sender, assumed fixed (in bits/s)  
max\_lat: maximum FEC-related latency within FECFRAME (a decimal number expressed in seconds)  
cr: RLC coding rate, ratio between the total number of source symbols and the total number of source plus repair symbols  
ew\_size: encoding window current size at a sender (in symbols)  
ew\_max\_size: encoding window maximum size at a sender (in symbols)  
dw\_max\_size: decoding window maximum size at a receiver (in symbols)  
ls\_max\_size: linear system maximum size (or width) at a receiver (in symbols)  
WSR: window size ratio parameter used to derive ew\_max\_size (encoder) and ls\_max\_size (decoder).  
PRNG: pseudo-random number generator  
TinyMT32: PRNG used in this specification.  
DT: coding coefficients density threshold, an integer between 0 and 15 (inclusive) the controls the fraction of coefficients that are non zero

### 3. Common Procedures

This section introduces the procedures that are used by these FEC Schemes.

#### 3.1. Codec Parameters

A codec implementing the Sliding Window RLC FEC Scheme relies on several parameters:

Maximum FEC-related latency budget, max\_lat (a decimal number expressed in seconds) with real-time flows:  
a source ADU flow can have real-time constraints, and therefore any FECFRAME related operation should take place within the validity period of each ADU (Appendix D describes an exception to this rule). When there are multiple flows with different real-time constraints, we consider the most stringent constraints (see [RFC6363], Section 10.2, item 6, for recommendations when several flows are globally protected). The maximum FEC-related latency budget, max\_lat, accounts for all sources of latency added by FEC encoding (at a sender) and FEC decoding (at a receiver). Other sources of latency (e.g., added by network communications) are out of scope and must be considered separately (said differently, they have already been deducted from max\_lat). max\_lat can be regarded as the latency budget permitted for all FEC-related operations. This is an input parameter that enables a FECFRAME sender to derive other internal parameters (see Appendix C);

Encoding window current (resp. maximum) size, `ew_size` (resp. `ew_max_size`) (in symbols):

at a FECFRAME sender, during FEC encoding, a repair symbol is computed as a linear combination of the `ew_size` source symbols present in the encoding window. The `ew_max_size` is the maximum size of this window, while `ew_size` is the current size. For example, in the common case at session start, upon receiving new source ADUs, the `ew_size` progressively increases until it reaches its maximum value, `ew_max_size`. We have:

$$0 < \text{ew\_size} \leq \text{ew\_max\_size}$$

Decoding window maximum size, `dw_max_size` (in symbols): at a FECFRAME receiver, `dw_max_size` is the maximum number of received or lost source symbols that are still within their latency budget;

Linear system maximum size, `ls_max_size` (in symbols): at a FECFRAME receiver, the linear system maximum size, `ls_max_size`, is the maximum number of received or lost source symbols in the linear system (i.e., the variables). It SHOULD NOT be smaller than `dw_max_size` since it would mean that, even after receiving a sufficient number of FEC Repair Packets, a lost ADU may not be recovered just because the associated source symbols have been prematurely removed from the linear system, which is usually counter-productive. On the opposite, the linear system MAY grow beyond the `dw_max_size` (Appendix D);

Symbol size, `E` (in bytes): the `E` parameter determines the source and repair symbol sizes (necessarily equal). This is an input parameter that enables a FECFRAME sender to derive other internal parameters, as explained below. An implementation at a sender MUST fix the `E` parameter and MUST communicate it as part of the FEC Scheme-Specific Information (Section 4.1.1.2).

Code rate, `cr`: The code rate parameter determines the amount of redundancy added to the flow. More precisely the `cr` is the ratio between the total number of source symbols and the total number of source plus repair symbols and by definition:  $0 < \text{cr} \leq 1$ . This is an input parameter that enables a FECFRAME sender to derive other internal parameters, as explained below. However, there is no need to communicate the `cr` parameter per se (it's not required to process a repair symbol at a receiver). This code rate parameter can be static. However, in specific use-cases (e.g., with unicast transmissions in presence of a feedback mechanism that estimates the communication quality, out of scope of FECFRAME), the code rate may be adjusted dynamically.

Appendix C proposes non normative techniques to derive those parameters, depending on the use-case specificities.

### 3.2. ADU, ADUI and Source Symbols Mappings

At a sender, an ADU coming from the application is not directly mapped to source symbols. When multiple source flows (e.g., media streams) are mapped onto the same FECFRAME instance, each flow is assigned its own Flow ID value (see below). This Flow ID is then prepended to each ADU before FEC encoding. This way, FEC decoding at a receiver also recovers this Flow ID and the recovered ADU can be assigned to the right source flow (note that the 5-tuple used to identify the right source flow of a received ADU is absent with a recovered ADU since it is not FEC protected).

Additionally, since ADUs are of variable size, padding is needed so that each ADU (with its flow identifier) contribute to an integral number of source symbols. This requires adding the original ADU length to each ADU before doing FEC encoding. Because of these requirements, an intermediate format, the ADUI, or ADU Information, is considered [RFC6363].

For each incoming ADU, an ADUI MUST be created as follows. First of all, 3 bytes are prepended (Figure 1):

Flow ID (F) (8-bit field): this unsigned byte contains the integer identifier associated to the source ADU flow to which this ADU belongs. It is assumed that a single byte is sufficient, which implies that no more than 256 flows will be protected by a single FECFRAME session instance.

Length (L) (16-bit field): this unsigned integer contains the length of this ADU, in network byte order (i.e., big endian). This length is for the ADU itself and does not include the F, L, or Pad fields.

Then, zero padding is added to the ADU if needed:

Padding (Pad) (variable size field): this field contains zero padding to align the F, L, ADU and padding up to a size that is multiple of E bytes (i.e., the source and repair symbol length).

The data unit resulting from the ADU and the F, L, and Pad fields is called ADUI. Since ADUs can have different sizes, this is also the case for ADUIs. However, an ADUI always contributes to an integral number of source symbols.

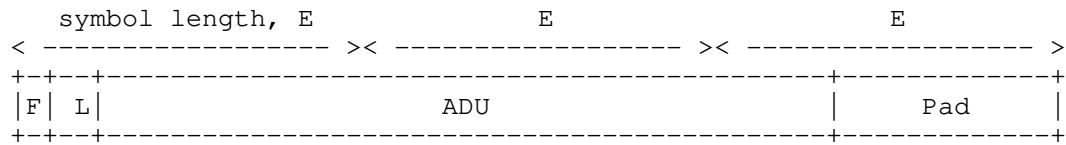


Figure 1: ADUI Creation example (here 3 source symbols are created for this ADUI).

Note that neither the initial 3 bytes nor the optional padding are sent over the network. However, they are considered during FEC encoding, and a receiver who lost a certain FEC Source Packet (e.g., the UDP datagram containing this FEC Source Packet when UDP is used as the transport protocol) will be able to recover the ADUI if FEC decoding succeeds. Thanks to the initial 3 bytes, this receiver will get rid of the padding (if any) and identify the corresponding ADU flow.

### 3.3. Encoding Window Management

Source symbols and the corresponding ADUs are removed from the encoding window:

- o when the sliding encoding window has reached its maximum size, `ew_max_size`. In that case the oldest symbol MUST be removed before adding a new symbol, so that the current encoding window size always remains inferior or equal to the maximum size: `ew_size <= ew_max_size`;
- o when an ADU has reached its maximum validity duration in case of a real-time flow. When this happens, all source symbols corresponding to the ADUI that expired SHOULD be removed from the encoding window;

Source symbols are added to the sliding encoding window each time a new ADU arrives, once the ADU-to-source symbols mapping has been performed (Section 3.2). The current size of the encoding window, `ew_size`, is updated after adding new source symbols. This process may require to remove old source symbols so that: `ew_size <= ew_max_size`.

Note that a FEC codec may feature practical limits in the number of source symbols in the encoding window (e.g., for computational complexity reasons). This factor may further limit the `ew_max_size` value, in addition to the maximum FEC-related latency budget (Section 3.1).

### 3.4. Source Symbol Identification

Each source symbol is identified by an Encoding Symbol ID (ESI), an unsigned integer. The ESI of source symbols MUST start with value 0 for the first source symbol and MUST be managed sequentially. Wrapping to zero happens after reaching the maximum value made possible by the ESI field size (this maximum value is FEC Scheme dependant, for instance,  $2^{32}-1$  with FEC Schemes XXX and YYY).

No such consideration applies to repair symbols.

### 3.5. Pseudo-Random Number Generator (PRNG)

In order to compute coding coefficients (see Section 3.6), the RLC FEC Schemes rely on the TinyMT32 PRNG defined in [tinymt32] with two additional functions defined in this section.

This PRNG MUST first be initialized with a 32-bit unsigned integer, used as a seed, with:

```
void tinymt32_init (tinymt32_t * s, uint32_t seed);
```

With the FEC Schemes defined in this document, the seed is in practice restricted to a value between 0 and 0xFFFF inclusive (note that this PRNG accepts a seed value equal to 0), since this is the Repair\_Key 16-bit field value of the Repair FEC Payload ID (Section 4.1.3). In practice, how to manage the seed and Repair\_Key values (both are equal) is left to the implementer, using a monotonically increasing counter being one possibility (Section 6.1). In addition to the seed, this function takes as parameter a pointer to an instance of a tinymt32\_t structure that is used to keep the internal state of the PRNG.

Then, each time a new pseudo-random integer between 0 and 15 inclusive (4-bit pseudo-random integer) is needed, the following function is used:

```
uint32_t tinymt32_rand16 (tinymt32_t * s);
```

This function takes as parameter a pointer to the same tinymt32\_t structure (that is left unchanged between successive calls to the function).

Similarly, each time a new pseudo-random integer between 0 and 255 inclusive (8-bit pseudo-random integer) is needed, the following function is used:

```
uint32_t tinymt32_rand256 (tinymt32_t * s);
```

These two functions keep respectively the 4 or 8 less significant bits of the 32-bit pseudo-random number generated by the `tinymt32_generate_uint32()` function of [tinymt32]. This is done by computing the result of a binary AND between the `tinymt32_generate_uint32()` output and respectively the 0xF or 0xFF constants, using 32-bit unsigned integer operations. Figure 2 shows a possible implementation. This is a C language implementation, written for C99 [C99]. Test results discussed in Appendix B show that this simple technique, applied to this PRNG, is in line with the RLC FEC Schemes needs.

```
<CODE BEGINS>
/**
 * This function outputs a pseudo-random integer in [0 .. 15] range.
 *
 * @param s      pointer to tinymt internal state.
 * @return       unsigned integer between 0 and 15 inclusive.
 */
uint32_t tinymt32_rand16(tinymt32_t *s)
{
    return (tinymt32_generate_uint32(s) & 0xF);
}

/**
 * This function outputs a pseudo-random integer in [0 .. 255] range.
 *
 * @param s      pointer to tinymt internal state.
 * @return       unsigned integer between 0 and 255 inclusive.
 */
uint32_t tinymt32_rand256(tinymt32_t *s)
{
    return (tinymt32_generate_uint32(s) & 0xFF);
}
<CODE ENDS>
```

Figure 2: 4-bit and 8-bit mapping functions for TinyMT32

Any implementation of this PRNG MUST have the same output as that provided by the reference implementation of [tinymt32]. In order to increase the compliancy confidence, three criteria are proposed: the one described in [tinymt32] (for the TinyMT32 32-bit unsigned integer generator), and the two others detailed in Appendix A (for the mapping to 4-bit and 8-bit intervals). Because of the way the mapping functions work, it is unlikely that an implementation that fulfills the first criterion fails to fulfill the two others.



### 3.6. Coding Coefficients Generation Function

The coding coefficients, used during the encoding process, are generated at the RLC encoder by the `generate_coding_coefficients()` function each time a new repair symbol needs to be produced. The fraction of coefficients that are non zero (i.e., the density) is controlled by the DT (Density Threshold) parameter. DT has values between 0 (the minimum value) and 15 (the maximum value), and the average probability of having a non zero coefficient equals  $(DT + 1) / 16$ . In particular, when DT equals 15 the function guaranties that all coefficients are non zero (i.e., maximum density).

These considerations apply to both the RLC over GF(2) and RLC over GF( $2^{^8}$ ), the only difference being the value of the m parameter. With the RLC over GF(2) FEC Scheme (Section 5), m is equal to 1. With RLC over GF( $2^{^8}$ ) FEC Scheme (Section 4), m is equal to 8.

Figure 3 shows the reference `generate_coding_coefficients()` implementation. This is a C language implementation, written for C99 [C99].

```
<CODE BEGINS>
#include <string.h>

/*
 * Fills in the table of coding coefficients (of the right size)
 * provided with the appropriate number of coding coefficients to
 * use for the repair symbol key provided.
 *
 * (in) repair_key    key associated to this repair symbol. This
 *                    parameter is ignored (useless) if m=1 and dt=15
 * (in/out) cc_tab    pointer to a table of the right size to store
 *                    coding coefficients. All coefficients are
 *                    stored as bytes, regardless of the m parameter,
 *                    upon return of this function.
 * (in) cc_nb         number of entries in the cc_tab table. This
 *                    value is equal to the current encoding window
 *                    size.
 * (in) dt            integer between 0 and 15 (inclusive) that
 *                    controls the density. With value 15, all
 *                    coefficients are guaranteed to be non zero
 *                    (i.e. equal to 1 with GF(2) and equal to a
 *                    value in {1,... 255} with GF( $2^{^8}$ )), otherwise
 *                    a fraction of them will be 0.
 * (in) m             Finite Field GF( $2^{^m}$ ) parameter. In this
 *                    document only values 1 and 8 are considered.
 * (out)              returns 0 in case of success, an error code
 *                    different than 0 otherwise.
```

```
*/
int generate_coding_coefficients (uint16_t  repair_key,
                                uint8_t*   cc_tab,
                                uint16_t   cc_nb,
                                uint8_t    dt,
                                uint8_t    m)
{
    uint32_t    i;
    tinymt32_t  s;    /* PRNG internal state */

    if (dt > 15) {
        return -1; /* error, bad dt parameter */
    }
    switch (m) {
    case 1:
        if (dt == 15) {
            /* all coefficients are 1 */
            memset(cc_tab, 1, cc_nb);
        } else {
            /* here coefficients are either 0 or 1 */
            tinymt32_init(&s, repair_key);
            for (i = 0 ; i < cc_nb ; i++) {
                cc_tab[i] = (tinymt32_rand16(&s) <= dt) ? 1 : 0;
            }
        }
        break;

    case 8:
        tinymt32_init(&s, repair_key);
        if (dt == 15) {
            /* coefficient 0 is avoided here in order to include
             * all the source symbols */
            for (i = 0 ; i < cc_nb ; i++) {
                do {
                    cc_tab[i] = (uint8_t) tinymt32_rand256(&s);
                } while (cc_tab[i] == 0);
            }
        } else {
            /* here a certain number of coefficients should be 0 */
            for (i = 0 ; i < cc_nb ; i++) {
                if (tinymt32_rand16(&s) <= dt) {
                    do {
                        cc_tab[i] = (uint8_t) tinymt32_rand256(&s);
                    } while (cc_tab[i] == 0);
                } else {
                    cc_tab[i] = 0;
                }
            }
        }
    }
}
```

```

        }
        break;

    default:
        return -2; /* error, bad parameter m */
    }
    return 0; /* success */
}
<CODE ENDS>

```

Figure 3: Coding Coefficients Generation Function Reference Implementation

### 3.7. Finite Fields Operations

#### 3.7.1. Finite Field Definitions

The two RLC FEC Schemes specified in this document reuse the Finite Fields defined in [RFC5510], section 8.1. More specifically, the elements of the field  $GF(2^m)$  are represented by polynomials with binary coefficients (i.e., over  $GF(2)$ ) and degree lower or equal to  $m-1$ . The addition between two elements is defined as the addition of binary polynomials in  $GF(2)$ , which is equivalent to a bitwise XOR operation on the binary representation of these elements.

With  $GF(2^8)$ , multiplication between two elements is the multiplication modulo a given irreducible polynomial of degree 8. The following irreducible polynomial is used for  $GF(2^8)$ :

$$x^8 + x^4 + x^3 + x^2 + 1$$

With  $GF(2)$ , multiplication corresponds to a logical AND operation.

#### 3.7.2. Linear Combination of Source Symbols Computation

The two RLC FEC Schemes require the computation of a linear combination of source symbols, using the coding coefficients produced by the `generate_coding_coefficients()` function and stored in the `cc_tab[]` array.

With the RLC over  $GF(2^8)$  FEC Scheme, a linear combination of the `ew_size` source symbol present in the encoding window, say `src_0` to `src_ew_size_1`, in order to generate a repair symbol, is computed as follows. For each byte of position `i` in each source and the repair symbol, where `i` belongs to `[0; E-1]`, compute:

$$\text{repair}[i] = \text{cc\_tab}[0] * \text{src\_0}[i] \text{ XOR } \text{cc\_tab}[1] * \text{src\_1}[i] \text{ XOR } \dots \text{ XOR } \text{cc\_tab}[\text{ew\_size} - 1] * \text{src\_ew\_size\_1}[i]$$

where  $*$  is the multiplication over  $GF(2^{^8})$ . In practice various optimizations need to be used in order to make this computation efficient (see in particular [PGM13]).

With the RLC over  $GF(2)$  FEC Scheme (binary case), a linear combination is computed as follows. The repair symbol is the XOR sum of all the source symbols corresponding to a coding coefficient  $cc\_tab[j]$  equal to 1 (i.e., the source symbols corresponding to zero coding coefficients are ignored). The XOR sum of the byte of position  $i$  in each source is computed and stored in the corresponding byte of the repair symbol, where  $i$  belongs to  $[0; E-1]$ . In practice, the XOR sums will be computed several bytes at a time (e.g., on 64 bit words, or on arrays of 16 or more bytes when using SIMD CPU extensions).

With both FEC Schemes, the details of how to optimize the computation of these linear combinations are of high practical importance but out of scope of this document.

#### 4. Sliding Window RLC FEC Scheme over $GF(2^{^8})$ for Arbitrary Packet Flows

This fully-specified FEC Scheme defines the Sliding Window Random Linear Codes (RLC) over  $GF(2^{^8})$ .

##### 4.1. Formats and Codes

###### 4.1.1. FEC Framework Configuration Information

Following the guidelines of [RFC6363], section 5.6, this section provides the FEC Framework Configuration Information (or FFCI). This FFCI needs to be shared (e.g., using SDP) between the FECFRAME sender and receiver instances in order to synchronize them. It includes a FEC Encoding ID, mandatory for any FEC Scheme specification, plus scheme-specific elements.

###### 4.1.1.1. FEC Encoding ID

- o FEC Encoding ID: the value assigned to this fully specified FEC Scheme MUST be XXXX, as assigned by IANA (Section 10).

When SDP is used to communicate the FFCI, this FEC Encoding ID is carried in the 'encoding-id' parameter.

#### 4.1.1.2. FEC Scheme-Specific Information

The FEC Scheme-Specific Information (FSSI) includes elements that are specific to the present FEC Scheme. More precisely:

Encoding symbol size (E): a non-negative integer that indicates the size of each encoding symbol in bytes;

Window Size Ratio (WSR) parameter: a non-negative integer between 0 and 255 (both inclusive) used to initialize window sizes. A value of 0 indicates this parameter is not considered (e.g., a fixed encoding window size may be chosen). A value between 1 and 255 inclusive is required by certain of the parameter derivation techniques described in Appendix C;

This element is required both by the sender (RLC encoder) and the receiver(s) (RLC decoder).

When SDP is used to communicate the FFCI, this FEC Scheme-specific information is carried in the 'fssi' parameter in textual representation as specified in [RFC6364]. For instance:

```
fssi=E:1400,WSR:191
```

In that case the name values "E" and "WSR" are used to convey the E and WSR parameters respectively.

If another mechanism requires the FSSI to be carried as an opaque octet string, the encoding format consists of the following three octets, where the E field is carried in "big-endian" or "network order" format, that is, most significant byte first:

Encoding symbol length (E): 16-bit field;  
Window Size Ratio Parameter (WSR): 8-bit field.

These three octets can be communicated as such, or for instance, be subject to an additional Base64 encoding.

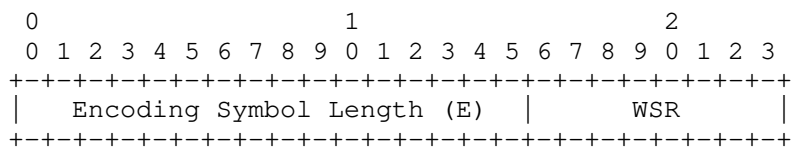


Figure 4: FSSI Encoding Format

#### 4.1.2. Explicit Source FEC Payload ID

A FEC Source Packet MUST contain an Explicit Source FEC Payload ID that is appended to the end of the packet as illustrated in Figure 5.

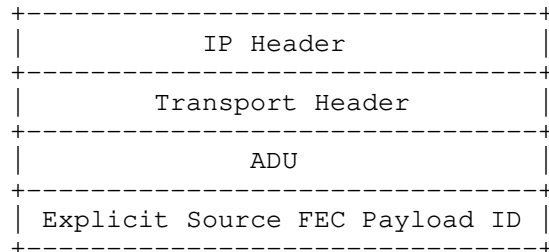


Figure 5: Structure of an FEC Source Packet with the Explicit Source FEC Payload ID

More precisely, the Explicit Source FEC Payload ID is composed of the following field, carried in "big-endian" or "network order" format, that is, most significant byte first (Figure 6):

Encoding Symbol ID (ESI) (32-bit field): this unsigned integer identifies the first source symbol of the ADUI corresponding to this FEC Source Packet. The ESI is incremented for each new source symbol, and after reaching the maximum value ( $2^{32}-1$ ), wrapping to zero occurs.

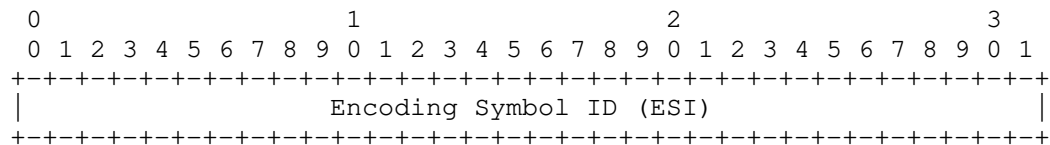


Figure 6: Source FEC Payload ID Encoding Format

#### 4.1.3. Repair FEC Payload ID

A FEC Repair Packet MAY contain one or more repair symbols. When there are several repair symbols, all of them MUST have been generated from the same encoding window, using Repair\_Key values that are managed as explained below. A receiver can easily deduce the number of repair symbols within a FEC Repair Packet by comparing the received FEC Repair Packet size (equal to the UDP payload size when UDP is the underlying transport protocol) and the symbol size, E, communicated in the FFCI.

A FEC Repair Packet MUST contain a Repair FEC Payload ID that is prepended to the repair symbol as illustrated in Figure 7.

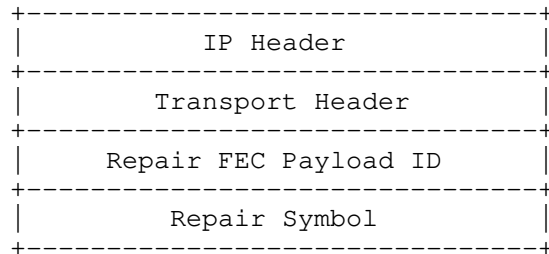


Figure 7: Structure of an FEC Repair Packet with the Repair FEC Payload ID

More precisely, the Repair FEC Payload ID is composed of the following fields where all integer fields are carried in "big-endian" or "network order" format, that is, most significant byte first (Figure 8):

**Repair\_Key (16-bit field):** this unsigned integer is used as a seed by the coefficient generation function (Section 3.6) in order to generate the desired number of coding coefficients. This repair key may be a monotonically increasing integer value that loops back to 0 after reaching 65535 (see Section 6.1). When a FEC Repair Packet contains several repair symbols, this repair key value is that of the first repair symbol. The remaining repair keys can be deduced by incrementing by 1 this value, up to a maximum value of 65535 after which it loops back to 0.

**Density Threshold for the coding coefficients, DT (4-bit field):** this unsigned integer carries the Density Threshold (DT) used by the coding coefficient generation function Section 3.6. More precisely, it controls the probability of having a non zero coding coefficient, which equals  $(DT+1) / 16$ . When a FEC Repair Packet contains several repair symbols, the DT value applies to all of them;

**Number of Source Symbols in the encoding window, NSS (12-bit field):**

this unsigned integer indicates the number of source symbols in the encoding window when this repair symbol was generated. When a FEC Repair Packet contains several repair symbols, this NSS value applies to all of them;

**ESI of First Source Symbol in the encoding window, FSS\_ESI (32-bit field):**

this unsigned integer indicates the ESI of the first source symbol in the encoding window when this repair symbol was generated.

When a FEC Repair Packet contains several repair symbols, this FSS\_ESI value applies to all of them;

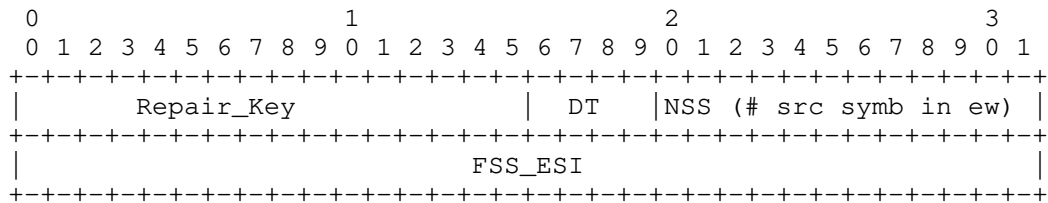


Figure 8: Repair FEC Payload ID Encoding Format

#### 4.2. Procedures

All the procedures of Section 3 apply to this FEC Scheme.

#### 5. Sliding Window RLC FEC Scheme over GF(2) for Arbitrary Packet Flows

This fully-specified FEC Scheme defines the Sliding Window Random Linear Codes (RLC) over GF(2) (binary case).

##### 5.1. Formats and Codes

###### 5.1.1. FEC Framework Configuration Information

###### 5.1.1.1. FEC Encoding ID

- o FEC Encoding ID: the value assigned to this fully specified FEC Scheme MUST be YYYY, as assigned by IANA (Section 10).

When SDP is used to communicate the FFCI, this FEC Encoding ID is carried in the 'encoding-id' parameter.

###### 5.1.1.2. FEC Scheme-Specific Information

All the considerations of Section 4.1.1.2 apply here.

###### 5.1.2. Explicit Source FEC Payload ID

All the considerations of Section 4.1.2 apply here.

###### 5.1.3. Repair FEC Payload ID

All the considerations of Section 4.1.3 apply here, with the only exception that the Repair\_Key field is useless if DT = 15 (indeed, in that case all the coefficients are necessarily equal to 1 and the coefficient generation function does not use any PRNG). When DT = 15



the FECFRAME sender MUST set the Repair\_Key field to zero on transmission and a receiver MUST ignore it on receipt.

## 5.2. Procedures

All the procedures of Section 3 apply to this FEC Scheme.

## 6. FEC Code Specification

### 6.1. Encoding Side

This section provides a high level description of a Sliding Window RLC encoder.

Whenever a new FEC Repair Packet is needed, the RLC encoder instance first gathers the ew\_size source symbols currently in the sliding encoding window. Then it chooses a repair key, which can be a monotonically increasing integer value, incremented for each repair symbol up to a maximum value of 65535 (as it is carried within a 16-bit field) after which it loops back to 0. This repair key is communicated to the coefficient generation function (Section 3.6) in order to generate ew\_size coding coefficients. Finally, the FECFRAME sender computes the repair symbol as a linear combination of the ew\_size source symbols using the ew\_size coding coefficients (Section 3.7). When E is small and when there is an incentive to pack several repair symbols within the same FEC Repair Packet, the appropriate number of repair symbols are computed. In that case the repair key for each of them MUST be incremented by 1, keeping the same ew\_size source symbols, since only the first repair key will be carried in the Repair FEC Payload ID. The FEC Repair Packet can then be passed to the transport layer for transmission. The source versus repair FEC packet transmission order is out of scope of this document and several approaches exist that are implementation-specific.

Other solutions are possible to select a repair key value when a new FEC Repair Packet is needed, for instance, by choosing a random integer between 0 and 65535. However, selecting the same repair key as before (which may happen in case of a random process) is only meaningful if the encoding window has changed, otherwise the same FEC Repair Packet will be generated. In any case, choosing the repair key is entirely at the discretion of the sender, since it is communicated to the receiver(s) in each Repair FEC Payload ID. A receiver should not make any assumption on the way the repair key is managed.

## 6.2. Decoding Side

This section provides a high level description of a Sliding Window RLC decoder.

A FECFRAME receiver needs to maintain a linear system whose variables are the received and lost source symbols. Upon receiving a FEC Repair Packet, a receiver first extracts all the repair symbols it contains (in case several repair symbols are packed together). For each repair symbol, when at least one of the corresponding source symbols it protects has been lost, the receiver adds an equation to the linear system (or no equation if this repair packet does not change the linear system rank). This equation of course re-uses the `ew_size` coding coefficients that are computed by the same coefficient generation function (Section 3.6), using the repair key and encoding window descriptions carried in the Repair FEC Payload ID. Whenever possible (i.e., when a sub-system covering one or more lost source symbols is of full rank), decoding is performed in order to recover lost source symbols. Gaussian elimination is one possible algorithm to solve this linear system. Each time an ADUI can be totally recovered, padding is removed (thanks to the Length field, `L`, of the ADUI) and the ADU is assigned to the corresponding application flow (thanks to the Flow ID field, `F`, of the ADUI). This ADU is finally passed to the corresponding upper application. Received FEC Source Packets, containing an ADU, MAY be passed to the application either immediately or after some time to guaranty an ordered delivery to the application. This document does not mandate any approach as this is an operational and management decision.

With real-time flows, a lost ADU that is decoded after the maximum latency or an ADU received after this delay has no value to the application. This raises the question of deciding whether or not an ADU is late. This decision MAY be taken within the FECFRAME receiver (e.g., using the decoding window, see Section 3.1) or within the application (e.g., using RTP timestamps within the ADU). Deciding which option to follow and whether or not to pass all ADUs, including those assumed late, to the application are operational decisions that depend on the application and are therefore out of scope of this document. Additionally, Appendix D discusses a backward compatible optimization whereby late source symbols MAY still be used within the FECFRAME receiver in order to improve transmission robustness.

## 7. Implementation Status

Editor's notes: RFC Editor, please remove this section motivated by RFC 6982 before publishing the RFC. Thanks.

An implementation of the Sliding Window RLC FEC Scheme for FECFRAME exists:

- o Organisation: Inria
- o Description: This is an implementation of the Sliding Window RLC FEC Scheme limited to  $GF(2^{8})$ . It relies on a modified version of our OpenFEC (<http://openfec.org>) FEC code library. It is integrated in our FECFRAME software (see [fecframe-ext]).
- o Maturity: prototype.
- o Coverage: this software complies with the Sliding Window RLC FEC Scheme.
- o Licensing: proprietary.
- o Contact: [vincent.roca@inria.fr](mailto:vincent.roca@inria.fr)

## 8. Security Considerations

The FEC Framework document [RFC6363] provides a fairly comprehensive analysis of security considerations applicable to FEC Schemes. Therefore, the present section follows the security considerations section of [RFC6363] and only discusses specific topics.

### 8.1. Attacks Against the Data Flow

#### 8.1.1. Access to Confidential Content

The Sliding Window RLC FEC Scheme specified in this document does not change the recommendations of [RFC6363]. To summarize, if confidentiality is a concern, it is RECOMMENDED that one of the solutions mentioned in [RFC6363] is used with special considerations to the way this solution is applied (e.g., is encryption applied before or after FEC protection, within the end-system or in a middlebox), to the operational constraints (e.g., performing FEC decoding in a protected environment may be complicated or even impossible) and to the threat model.

#### 8.1.2. Content Corruption

The Sliding Window RLC FEC Scheme specified in this document does not change the recommendations of [RFC6363]. To summarize, it is RECOMMENDED that one of the solutions mentioned in [RFC6363] is used on both the FEC Source and Repair Packets.

### 8.2. Attacks Against the FEC Parameters

The FEC Scheme specified in this document defines parameters that can be the basis of attacks. More specifically, the following parameters of the FFCI may be modified by an attacker who targets receivers (Section 4.1.1.2):

- o FEC Encoding ID: changing this parameter leads a receiver to consider a different FEC Scheme. The consequences are severe, the format of the Explicit Source FEC Payload ID and Repair FEC Payload ID of received packets will probably differ, leading to various malfunctions. Even if the original and modified FEC Schemes share the same format, FEC decoding will either fail or lead to corrupted decoded symbols. This will happen if an attacker turns value YYYY (i.e., RLC over  $GF(2)$ ) to value XXXX (RLC over  $GF(2^{8})$ ), an additional consequence being a higher processing overhead at the receiver. In any case, the attack results in a form of Denial of Service (DoS) or corrupted content.
- o Encoding symbol length (E): setting this E parameter to a different value will confuse a receiver. If the size of a received FEC Repair Packet is no longer multiple of the modified E value, a receiver quickly detects a problem and SHOULD reject the packet. If the new E value is a sub-multiple of the original E value (e.g., half the original value), then receivers may not detect the problem immediately. For instance, a receiver may think that a received FEC Repair Packet contains more repair symbols (e.g., twice as many if E is reduced by half), leading to malfunctions whose nature depends on implementation details. Here also, the attack always results in a form of DoS or corrupted content.

It is therefore RECOMMENDED that security measures be taken to guarantee the FFCI integrity, as specified in [RFC6363]. How to achieve this depends on the way the FFCI is communicated from the sender to the receiver, which is not specified in this document.

Similarly, attacks are possible against the Explicit Source FEC Payload ID and Repair FEC Payload ID. More specifically, in case of a FEC Source Packet, the following value can be modified by an attacker who targets receivers:

- o Encoding Symbol ID (ESI): changing the ESI leads a receiver to consider a wrong ADU, resulting in severe consequences, including corrupted content passed to the receiving application;

And in case of a FEC Repair Packet:

- o Repair Key: changing this value leads a receiver to generate a wrong coding coefficient sequence, and therefore any source symbol decoded using the repair symbols contained in this packet will be corrupted;
- o DT: changing this value also leads a receiver to generate a wrong coding coefficient sequence, and therefore any source symbol decoded using the repair symbols contained in this packet will be corrupted. In addition, if the DT value is significantly

- increased, it will generate a higher processing overhead at a receiver. In case of very large encoding windows, this may impact the terminal performance;
- o NSS: changing this value leads a receiver to consider a different set of source symbols, and therefore any source symbol decoded using the repair symbols contained in this packet will be corrupted. In addition, if the NSS value is significantly increased, it will generate a higher processing overhead at a receiver, which may impact the terminal performance;
  - o FSS\_ESI: changing this value also leads a receiver to consider a different set of source symbols and therefore any source symbol decoded using the repair symbols contained in this packet will be corrupted.

It is therefore RECOMMENDED that security measures are taken to guarantee the FEC Source and Repair Packets as stated in [RFC6363].

### 8.3. When Several Source Flows are to be Protected Together

The Sliding Window RLC FEC Scheme specified in this document does not change the recommendations of [RFC6363].

### 8.4. Baseline Secure FEC Framework Operation

The Sliding Window RLC FEC Scheme specified in this document does not change the recommendations of [RFC6363] concerning the use of the IPsec/ESP security protocol as a mandatory to implement (but not mandatory to use) security scheme. This is well suited to situations where the only insecure domain is the one over which the FEC Framework operates.

### 8.5. Additional Security Considerations for Numerical Computations

In addition to the above security considerations, inherited from [RFC6363], the present document introduces several formulae, in particular in Appendix C.1. It is RECOMMENDED to check that the computed values stay within reasonable bounds since numerical overflows, caused by an erroneous implementation or an erroneous input value, may lead to hazardous behaviours. However, what "reasonable bounds" means is use-case and implementation dependent and is not detailed in this document.

Appendix C.2 also mentions the possibility of "using the timestamp field of an RTP packet header" when applicable. A malicious attacker may deliberately corrupt this header field in order to trigger hazardous behaviours at a FECFRAME receiver. Protection against this type of content corruption can be addressed with the above recommendations on a baseline secure operation. In addition, it is

also RECOMMENDED to check that the timestamp value be within reasonable bounds.

## 9. Operations and Management Considerations

The FEC Framework document [RFC6363] provides a fairly comprehensive analysis of operations and management considerations applicable to FEC Schemes. Therefore, the present section only discusses specific topics.

### 9.1. Operational Recommendations: Finite Field GF(2) Versus GF(2<sup>8</sup>)

The present document specifies two FEC Schemes that differ on the Finite Field used for the coding coefficients. It is expected that the RLC over GF(2<sup>8</sup>) FEC Scheme will be mostly used since it warrants a higher packet loss protection. In case of small encoding windows, the associated processing overhead is not an issue (e.g., we measured decoding speeds between 745 Mbps and 2.8 Gbps on an ARM Cortex-A15 embedded board in [Roca17] depending on the code rate and the channel conditions, using an encoding window of size 18 or 23 symbols; see the above article for the details). Of course the CPU overhead will increase with the encoding window size, because more operations in the GF(2<sup>8</sup>) finite field will be needed.

The RLC over GF(2) FEC Scheme offers an alternative. In that case operations symbols can be directly XOR-ed together which warrants high bitrate encoding and decoding operations, and can be an advantage with large encoding windows. However, packet loss protection is significantly reduced by using this FEC Scheme.

### 9.2. Operational Recommendations: Coding Coefficients Density Threshold

In addition to the choice of the Finite Field, the two FEC Schemes define a coding coefficient density threshold (DT) parameter. This parameter enables a sender to control the code density, i.e., the proportion of coefficients that are non zero on average. With RLC over GF(2<sup>8</sup>), it is usually appropriate that small encoding windows be associated to a density threshold equal to 15, the maximum value, in order to warrant a high loss protection.

On the opposite, with larger encoding windows, it is usually appropriate that the density threshold be reduced. With large encoding windows, an alternative can be to use RLC over GF(2) and a density threshold equal to 7 (i.e., an average density equal to 1/2) or smaller.

Note that using a density threshold equal to 15 with RLC over GF(2) is equivalent to using an XOR code that computes the XOR sum of all

the source symbols in the encoding window. In that case: (1) only a single repair symbol can be produced for any encoding window, and (2) the `repair_key` parameter becomes useless (the coding coefficients generation function does not rely on the PRNG).

## 10. IANA Considerations

This document registers two values in the "FEC Framework (FECFRAME) FEC Encoding IDs" registry [RFC6363] as follows:

- o YYYY refers to the Sliding Window Random Linear Codes (RLC) over GF(2) FEC Scheme for Arbitrary Packet Flows, as defined in Section 5 of this document.
- o XXXX refers to the Sliding Window Random Linear Codes (RLC) over GF(2<sup>8</sup>) FEC Scheme for Arbitrary Packet Flows, as defined in Section 4 of this document.

## 11. Acknowledgments

The authors would like to thank the three TSVWG chairs, Wesley Eddy, our shepherd, David Black and Gorrry Fairhurst, as well as Spencer Dawkins, our responsible AD, and all those who provided comments, namely (alphabetical order) Alan DeKok, Jonathan Detchart, Russ Housley, Emmanuel Lochin, Marie-Jose Montpetit, and Greg Skinner. Last but not least, the authors are really grateful to the IESG members, in particular Benjamin Kaduk, Mirja Kuhlewind, Eric Rescorla, Adam Roach, and Roman Danyliw for their highly valuable feedbacks that greatly contributed to improve this specification.

## 12. References

### 12.1. Normative References

- [C99] "Programming languages - C: C99, correction 3:2007", International Organization for Standardization, ISO/IEC 9899:1999/Cor 3:2007, November 2007.
- [fecframe-ext]  
Roca, V. and A. Begen, "Forward Error Correction (FEC) Framework Extension to Sliding Window Codes", Transport Area Working Group (TSVWG) draft-ietf-tsvwg-fecframe-ext (Work in Progress), January 2019, <<https://tools.ietf.org/html/draft-ietf-tsvwg-fecframe-ext>>.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC6363] Watson, M., Begen, A., and V. Roca, "Forward Error Correction (FEC) Framework", RFC 6363, DOI 10.17487/RFC6363, October 2011, <<https://www.rfc-editor.org/info/rfc6363>>.
- [RFC6364] Begen, A., "Session Description Protocol Elements for the Forward Error Correction (FEC) Framework", RFC 6364, DOI 10.17487/RFC6364, October 2011, <<https://www.rfc-editor.org/info/rfc6364>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [tinymt32] Saito, M., Matsumoto, M., Roca, V., and E. Baccelli, "TinyMT32 Pseudo Random Number Generator (PRNG)", Transport Area Working Group (TSVWG) draft-roca-tsvwg-tinymt32 (Work in Progress), February 2019, <<https://tools.ietf.org/html/draft-roca-tsvwg-tinymt32>>.

## 12.2. Informative References

- [PGM13] Plank, J., Greenan, K., and E. Miller, "A Complete Treatment of Software Implementations of Finite Field Arithmetic for Erasure Coding Applications", University of Tennessee Technical Report UT-CS-13-717, <http://web.eecs.utk.edu/~plank/plank/papers/UT-CS-13-717.html>, October 2013, <<http://web.eecs.utk.edu/~plank/plank/papers/UT-CS-13-717.html>>.
- [RFC5170] Roca, V., Neumann, C., and D. Furodet, "Low Density Parity Check (LDPC) Staircase and Triangle Forward Error Correction (FEC) Schemes", RFC 5170, DOI 10.17487/RFC5170, June 2008, <<https://www.rfc-editor.org/info/rfc5170>>.
- [RFC5510] Lacan, J., Roca, V., Peltotalo, J., and S. Peltotalo, "Reed-Solomon Forward Error Correction (FEC) Schemes", RFC 5510, DOI 10.17487/RFC5510, April 2009, <<https://www.rfc-editor.org/info/rfc5510>>.



- [RFC6681] Watson, M., Stockhammer, T., and M. Luby, "Raptor Forward Error Correction (FEC) Schemes for FECFRAME", RFC 6681, DOI 10.17487/RFC6681, August 2012, <<https://www.rfc-editor.org/info/rfc6681>>.
- [RFC6726] Paila, T., Walsh, R., Luby, M., Roca, V., and R. Lehtonen, "FLUTE - File Delivery over Unidirectional Transport", RFC 6726, DOI 10.17487/RFC6726, November 2012, <<https://www.rfc-editor.org/info/rfc6726>>.
- [RFC6816] Roca, V., Cunche, M., and J. Lacan, "Simple Low-Density Parity Check (LDPC) Staircase Forward Error Correction (FEC) Scheme for FECFRAME", RFC 6816, DOI 10.17487/RFC6816, December 2012, <<https://www.rfc-editor.org/info/rfc6816>>.
- [RFC6865] Roca, V., Cunche, M., Lacan, J., Bouabdallah, A., and K. Matsuzono, "Simple Reed-Solomon Forward Error Correction (FEC) Scheme for FECFRAME", RFC 6865, DOI 10.17487/RFC6865, February 2013, <<https://www.rfc-editor.org/info/rfc6865>>.
- [RFC8406] Adamson, B., Adjih, C., Bilbao, J., Firoiu, V., Fitzek, F., Ghanem, S., Lochin, E., Masucci, A., Montpetit, M-J., Pedersen, M., Peralta, G., Roca, V., Ed., Saxena, P., and S. Sivakumar, "Taxonomy of Coding Techniques for Efficient Network Communications", RFC 8406, DOI 10.17487/RFC8406, June 2018, <<https://www.rfc-editor.org/info/rfc8406>>.
- [Roca16] Roca, V., Teibi, B., Burdinat, C., Tran, T., and C. Thienot, "Block or Convolutional AL-FEC Codes? A Performance Comparison for Robust Low-Latency Communications", HAL open-archive document, hal-01395937 <https://hal.inria.fr/hal-01395937/en/>, November 2016, <<https://hal.inria.fr/hal-01395937/en/>>.
- [Roca17] Roca, V., Teibi, B., Burdinat, C., Tran, T., and C. Thienot, "Less Latency and Better Protection with AL-FEC Sliding Window Codes: a Robust Multimedia CBR Broadcast Case Study", 13th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob17), October 2017 <https://hal.inria.fr/hal-01571609v1/en/>, October 2017, <<https://hal.inria.fr/hal-01571609v1/en/>>.

## Appendix A. TinyMT32 Validation Criteria (Normative)

PRNG determinism, for a given seed, is a requirement. Consequently, in order to validate an implementation of the TinyMT32 PRNG, the following criteria MUST be met.

The first criterion focusses on the `tinymt32_rand256()`, where the 32-bit integer of the core TinyMT32 PRNG is scaled down to an 8-bit integer. Using a seed value of 1, the first 50 values returned by: `tinymt32_rand256()` as 8-bit unsigned integers MUST be equal to values provided in Figure 9, to be read line by line.

37	225	177	176	21
246	54	139	168	237
211	187	62	190	104
135	210	99	176	11
207	35	40	113	179
214	254	101	212	211
226	41	234	232	203
29	194	211	112	107
217	104	197	135	23
89	210	252	109	166

Figure 9: First 50 decimal values (to be read per line) returned by `tinymt32_rand256()` as 8-bit unsigned integers, with a seed value of 1.

The second criterion focusses on the `tinymt32_rand16()`, where the 32-bit integer of the core TinyMT32 PRNG is scaled down to a 4-bit integer. Using a seed value of 1, the first 50 values returned by: `tinymt32_rand16()` as 4-bit unsigned integers MUST be equal to values provided in Figure 10, to be read line by line.

5	1	1	0	5
6	6	11	8	13
3	11	14	14	8
7	2	3	0	11
15	3	8	1	3
6	14	5	4	3
2	9	10	8	11
13	2	3	0	11
9	8	5	7	7
9	2	12	13	6

Figure 10: First 50 decimal values (to be read per line) returned by `tinymt32_rand16()` as 4-bit unsigned integers, with a seed value of 1.

## Appendix B. Assessing the PRNG Adequacy (Informational)

This annex discusses the adequacy of the TinyMT32 PRNG and the `tinymt32_rand16()` and `tinymt32_rand256()` functions, to the RLC FEC Schemes. The goal is to assess the adequacy of these two functions in producing coding coefficients that are sufficiently different from one another, across various repair symbols with repair key values in sequence (we can expect this approach to be commonly used by implementers, see Section 6.1). This section is purely informational and does not claim to be a solid evaluation.

The two RLC FEC Schemes use the PRNG to produce pseudo-random coding coefficients (Section 3.6), each time a new repair symbol is needed. A different repair key is used for each repair symbol, usually by incrementing the repair key value (Section 6.1). For each repair symbol, a limited number of pseudo-random numbers is needed, depending on the DT and encoding window size (Section 3.6), using either `tinymt32_rand16()` or `tinymt32_rand256()`. Therefore we are more interested in the randomness of small sequences of random numbers mapped to 4-bit or 8-bit integers, than in the randomness of a very large sequence of random numbers which is not representative of the usage of the PRNG.

Evaluation of `tinymt32_rand16()`: We first generate a huge number (1,000,000,000) of small sequences (20 pseudo-random numbers per sequence), increasing the seed value for each sequence, and perform statistics on the number of occurrences of each of the 16 possible values across all sequences. In this first test we consider 32-bit seed values in order to assess the PRNG quality after output truncation to 4 bits.

value	occurrences	percentage (%) (total of 20000000000)
0	1250036799	6.2502
1	1249995831	6.2500
2	1250038674	6.2502
3	1250000881	6.2500
4	1250023929	6.2501
5	1249986320	6.2499
6	1249995587	6.2500
7	1250020363	6.2501
8	1249995276	6.2500
9	1249982856	6.2499
10	1249984111	6.2499
11	1250009551	6.2500
12	1249955768	6.2498
13	1249994654	6.2500
14	1250000569	6.2500
15	1249978831	6.2499

Figure 11: `tinymt32_rand16()`: occurrence statistics across a huge number (1,000,000,000) of small sequences (20 pseudo-random numbers per sequence), with 0 as the first PRNG seed.

The results (Figure 11) show that all possible values are almost equally represented, or said differently, that the `tinymt32_rand16()` output converges to a uniform distribution where each of the 16 possible values would appear exactly  $1 / 16 * 100 = 6.25\%$  of times.

Since the RLC FEC Schemes use of this PRNG will be limited to 16-bit seed values, we carried out the same test for the first  $2^{16}$  seed values only. The distribution (not shown) is of course less uniform, with value occurrences ranging between 6.2121% (i.e., 81,423 occurrences out of a total of  $65536 * 20 = 1,310,720$ ) and 6.2948% (i.e., 82,507 occurrences). However, we do not believe it significantly impacts the RLC FEC Scheme behavior.

Other types of biases may exist that may be visible with smaller tests, for instance to evaluate the convergence speed to a uniform distribution. We therefore perform 200 tests, each of them consisting in producing 200 sequences, keeping only the first value of each sequence. We use non overlapping repair keys for each sequence, starting with value 0 and increasing it after each use.

value	min occurrences	max occurrences	average occurrences
0	4	21	6.3675
1	4	22	6.0200
2	4	20	6.3125
3	5	23	6.1775
4	5	24	6.1000
5	4	21	6.5925
6	5	30	6.3075
7	6	22	6.2225
8	5	26	6.1750
9	3	21	5.9425
10	5	24	6.3175
11	4	22	6.4300
12	5	21	6.1600
13	5	22	6.3100
14	4	26	6.3950
15	4	21	6.1700

Figure 12: `tinymt32_rand16()`: occurrence statistics across 200 tests, each of them consisting in 200 sequences of 1 pseudo-random number each, with non overlapping PRNG seeds in sequence starting from 0.

Figure 12 shows across all 200 tests, for each of the 16 possible pseudo-random number values, the minimum (resp. maximum) number of times it appeared in a test, as well as the average number of occurrences across the 200 tests. Although the distribution is not perfect, there is no major bias. On the opposite, in the same conditions, the Park-Miller linear congruential PRNG of [RFC5170] with a result scaled down to 4-bit values, using seeds in sequence starting from 1, returns systematically 0 as the first value during some time, then after a certain repair key value threshold, it systematically returns 1, etc.

Evaluation of `tinymt32_rand256()`: The same approach is used here. Results (not shown) are similar: occurrences vary between 7,810,3368 (i.e., 0.3905%) and 7,814,7952 (i.e., 0.3907%). Here also we see a convergence to the theoretical uniform distribution where each of the 256 possible values would appear exactly  $1 / 256 * 100 = 0.390625\%$  of times.

#### Appendix C. Possible Parameter Derivation (Informational)

Section 3.1 defines several parameters to control the encoder or decoder. This annex proposes techniques to derive these parameters according to the target use-case. This annex is informational, in the sense that using a different derivation technique will not prevent the encoder and decoder to interoperate: a decoder can still recover an erased source symbol without any error. However, in case

of a real-time flow, an inappropriate parameter derivation may lead to the decoding of erased source packets after their validity period, making them useless to the target application. This annex proposes an approach to reduce this risk, among other things.

The FEC Schemes defined in this document can be used in various manners, depending on the target use-case:

- o the source ADU flow they protect may or may not have real-time constraints;
- o the source ADU flow may be a Constant Bitrate (CBR) or Variable BitRate (VBR) flow;
- o with a VBR source ADU flow, the flow's minimum and maximum bitrates may or may not be known;
- o and the communication path between encoder and decoder may be a CBR communication path (e.g., as with certain LTE-based broadcast channels) or not (general case, e.g., with Internet).

The parameter derivation technique should be suited to the use-case, as described in the following sections.

#### C.1. Case of a CBR Real-Time Flow

In the following, we consider a real-time flow with `max_lat` latency budget. The encoding symbol size, `E`, is constant. The code rate, `cr`, is also constant, its value depending on the expected communication loss model (this choice is out of scope of this document).

In a first configuration, the source ADU flow bitrate at the input of the FECFRAME sender is fixed and equal to `br_in` (in bits/s), and this value is known by the FECFRAME sender. It follows that the transmission bitrate at the output of the FECFRAME sender will be higher, depending on the added repair flow overhead. In order to comply with the maximum FEC-related latency budget, we have:

$$\text{dw\_max\_size} = (\text{max\_lat} * \text{br\_in}) / (8 * E)$$

assuming that the encoding and decoding times are negligible with respect to the target `max_lat`. This is a reasonable assumption in many situations (e.g., see Section 9.1 in case of small window sizes). Otherwise the `max_lat` parameter should be adjusted in order to avoid the problem. In any case, interoperability will never be compromised by choosing a too large value.

In a second configuration, the FECFRAME sender generates a fixed bitrate flow, equal to the CBR communication path bitrate equal to `br_out` (in bits/s), and this value is known by the FECFRAME sender,

as in [Roca17]. The maximum source flow bitrate needs to be such that, with the added repair flow overhead, the total transmission bitrate remains inferior or equal to `br_out`. We have:

$$\text{dw\_max\_size} = (\text{max\_lat} * \text{br\_out} * \text{cr}) / (8 * E)$$

assuming here also that the encoding and decoding times are negligible with respect to the target `max_lat`.

For decoding to be possible within the latency budget, it is required that the encoding window maximum size be smaller than or at most equal to the decoding window maximum size. The `ew_max_size` is the main parameter at a FECFRAME sender, but its exact value has no impact on the the FEC-related latency budget. The `ew_max_size` parameter is computed as follows:

$$\text{ew\_max\_size} = \text{dw\_max\_size} * \text{WSR} / 255$$

In line with [Roca17], `WSR = 191` is considered as a reasonable value (the resulting encoding to decoding window size ratio is then close to 0.75), but other values between 1 and 255 inclusive are possible, depending on the use-case.

The `dw_max_size` is computed by a FECFRAME sender but not explicitly communicated to a FECFRAME receiver. However, a FECFRAME receiver can easily evaluate the `ew_max_size` by observing the maximum Number of Source Symbols (NSS) value contained in the Repair FEC Payload ID of received FEC Repair Packets (Section 4.1.3). A receiver can then easily compute `dw_max_size`:

$$\text{dw\_max\_size} = \text{max\_NSS\_observed} * 255 / \text{WSR}$$

A receiver can then chose an appropriate linear system maximum size:

$$\text{ls\_max\_size} \geq \text{dw\_max\_size}$$

It is good practice to use a larger value for `ls_max_size` as explained in Appendix D, which does not impact maximum latency nor interoperability.

In any case, for a given use-case (i.e., for target encoding and decoding devices and desired protection levels in front of communication impairments) and for the computed `ew_max_size`, `dw_max_size` and `ls_max_size` values, it is RECOMMENDED to check that the maximum encoding time and maximum memory requirements at a FECFRAME sender, and maximum decoding time and maximum memory requirements at a FECFRAME receiver, stay within reasonable bounds. When assuming that the encoding and decoding times are negligible

with respect to the target `max_lat`, this should be verified as well, otherwise the `max_lat` SHOULD be adjusted accordingly.

The particular case of session start needs to be managed appropriately since the `ew_size`, starting at zero, increases each time a new source ADU is received by the FECFRAME sender, until it reaches the `ew_max_size` value. Therefore a FECFRAME receiver SHOULD continuously observe the received FEC Repair Packets, since the NSS value carried in the Repair FEC Payload ID will increase too, and adjust its `ls_max_size` accordingly if need be. With a CBR flow, session start is expected to be the only moment when the encoding window size will increase. Similarly, with a CBR real-time flow, the session end is expected to be the only moment when the encoding window size will progressively decrease. No adjustment of the `ls_max_size` is required at the FECFRAME receiver in that case.

## C.2. Other Types of Real-Time Flow

In the following, we consider a real-time source ADU flow with a `max_lat` latency budget and a variable bitrate (VBR) measured at the entry of the FECFRAME sender. A first approach consists in considering the smallest instantaneous bitrate of the source ADU flow, when this parameter is known, and to reuse the derivation of Appendix C.1. Considering the smallest bitrate means that the encoding and decoding window maximum size estimations are pessimistic: these windows have the smallest size required to enable on-time decoding at a FECFRAME receiver. If the instantaneous bitrate is higher than this smallest bitrate, this approach leads to an encoding window that is unnecessarily small, which reduces robustness in front of long erasure bursts.

Another approach consists in using ADU timing information (e.g., using the timestamp field of an RTP packet header, or registering the time upon receiving a new ADU). From the global FEC-related latency budget, the FECFRAME sender can derive a practical maximum latency budget for encoding operations, `max_lat_for_encoding`. For the FEC Schemes specified in this document, this latency budget SHOULD be computed with:

$$\text{max\_lat\_for\_encoding} = \text{max\_lat} * \text{WSR} / 255$$

It follows that any source symbols associated to an ADU that has timed-out with respect to `max_lat_for_encoding` SHOULD be removed from the encoding window. With this approach there is no pre-determined `ew_size` value: this value fluctuates over the time according to the instantaneous source ADU flow bitrate. For practical reasons, a FECFRAME sender may still require that `ew_size` does not increase beyond a maximum value (Appendix C.3).



With both approaches, and no matter the choice of the FECFRAME sender, a FECFRAME receiver can still easily evaluate the `ew_max_size` by observing the maximum Number of Source Symbols (NSS) value contained in the Repair FEC Payload ID of received FEC Repair Packets. A receiver can then compute `dw_max_size` and derive an appropriate `ls_max_size` as explained in Appendix C.1.

When the observed NSS fluctuates significantly, a FECFRAME receiver may want to adapt its `ls_max_size` accordingly. In particular when the NSS is significantly reduced, a FECFRAME receiver may want to reduce the `ls_max_size` too in order to limit computation complexity. A balance must be found between using an `ls_max_size` "too large" (which increases computation complexity and memory requirements) and the opposite (which reduces recovery performance).

### C.3. Case of a Non Real-Time Flow

Finally there are configurations where a source ADU flow has no real-time constraints. FECFRAME and the FEC Schemes defined in this document can still be used. The choice of appropriate parameter values can be directed by practical considerations. For instance, it can derive from an estimation of the maximum memory amount that could be dedicated to the linear system at a FECFRAME receiver, or the maximum computation complexity at a FECFRAME receiver, both of them depending on the `ls_max_size` parameter. The same considerations also apply to the FECFRAME sender, where the maximum memory amount and computation complexity depend on the `ew_max_size` parameter.

Here also, the NSS value contained in FEC Repair Packets is used by a FECFRAME receiver to determine the current coding window size and `ew_max_size` by observing its maximum value over the time.

## Appendix D. Decoding Beyond Maximum Latency Optimization (Informational)

This annex introduces non normative considerations. It is provided as suggestions, without any impact on interoperability. For more information see [Roca16].

With a real-time source ADU flow, it is possible to improve the decoding performance of sliding window codes without impacting maximum latency, at the cost of extra memory and CPU overhead. The optimization consists, for a FECFRAME receiver, to extend the linear system beyond the decoding window maximum size, by keeping a certain number of old source symbols whereas their associated ADUs timed-out:

$$ls\_max\_size > dw\_max\_size$$

Usually the following choice is a good trade-off between decoding performance and extra CPU overhead:

```
ls_max_size = 2 * dw_max_size
```

When the `dw_max_size` is very small, it may be preferable to keep a minimum `ls_max_size` value (e.g., `LS_MIN_SIZE_DEFAULT = 40` symbols). Going below this threshold will not save a significant amount of memory nor CPU cycles. Therefore:

```
ls_max_size = max(2 * dw_max_size, LS_MIN_SIZE_DEFAULT)
```

Finally, it is worth noting that a receiver that benefits from an FEC protection significantly higher than what is required to recover from packet losses, can choose to reduce the `ls_max_size`. In that case lost ADUs will be recovered without relying on this optimization.

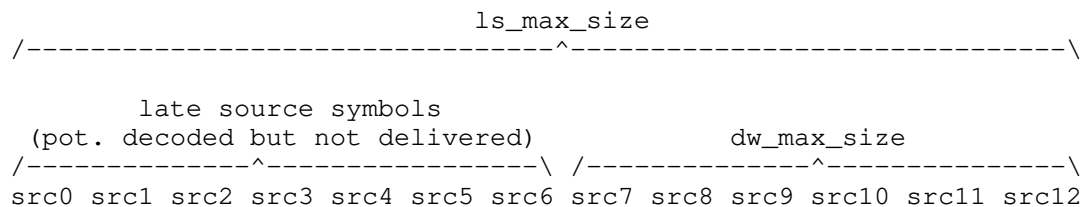


Figure 13: Relationship between parameters to decode beyond maximum latency.

It means that source symbols, and therefore ADUs, may be decoded even if the added latency exceeds the maximum value permitted by the application (the "late source symbols" of Figure 13). It follows that the corresponding ADUs will not be useful to the application. However, decoding these "late symbols" significantly improves the global robustness in bad reception conditions and is therefore recommended for receivers experiencing bad communication conditions [Roca16]. In any case whether or not to use this optimization and what exact value to use for the `ls_max_size` parameter are local decisions made by each receiver independently, without any impact on the other receivers nor on the source.

#### Authors' Addresses

Vincent Roca  
INRIA  
Univ. Grenoble Alpes  
France

EMail: [vincent.roca@inria.fr](mailto:vincent.roca@inria.fr)

Belkacem Teibi  
INRIA  
Univ. Grenoble Alpes  
France

EMail: [belkacem.teibi@gmail.com](mailto:belkacem.teibi@gmail.com)

TSVWG  
Internet-Draft  
Intended status: Informational  
Expires: October 20, 2021

G. Fairhurst  
University of Aberdeen  
C. Perkins  
University of Glasgow  
April 18, 2021

Considerations around Transport Header Confidentiality, Network  
Operations, and the Evolution of Internet Transport Protocols  
draft-ietf-tsvwg-transport-encrypt-21

Abstract

To protect user data and privacy, Internet transport protocols have supported payload encryption and authentication for some time. Such encryption and authentication is now also starting to be applied to the transport protocol headers. This helps avoid transport protocol ossification by middleboxes, mitigate attacks against the transport protocol, and protect metadata about the communication. Current operational practice in some networks inspect transport header information within the network, but this is no longer possible when those transport headers are encrypted.

This document discusses the possible impact when network traffic uses a protocol with an encrypted transport header. It suggests issues to consider when designing new transport protocols or features.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 20, 2021.

## Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Current uses of Transport Headers within the Network . . . .	4
2.1. To Separate Flows in Network Devices . . . . .	5
2.2. To Identify Transport Protocols and Flows . . . . .	5
2.3. To Understand Transport Protocol Performance . . . . .	6
2.4. To Support Network Operations . . . . .	13
2.5. To Mitigate the Effects of Constrained Networks . . . . .	18
2.6. To Verify SLA Compliance . . . . .	19
3. Research, Development and Deployment . . . . .	20
3.1. Independent Measurement . . . . .	20
3.2. Measurable Transport Protocols . . . . .	21
3.3. Other Sources of Information . . . . .	22
4. Encryption and Authentication of Transport Headers . . . . .	23
5. Intentionally Exposing Transport Information to the Network .	28
5.1. Exposing Transport Information in Extension Headers . . .	28
5.2. Common Exposed Transport Information . . . . .	29
5.3. Considerations for Exposing Transport Information . . . .	29
6. Addition of Transport OAM Information to Network-Layer Headers . . . . .	29
6.1. Use of OAM within a Maintenance Domain . . . . .	30
6.2. Use of OAM across Multiple Maintenance Domains . . . . .	30
7. Conclusions . . . . .	31
8. Security Considerations . . . . .	34
9. IANA Considerations . . . . .	36
10. Acknowledgements . . . . .	36
11. Informative References . . . . .	36
Appendix A. Revision information . . . . .	46
Authors' Addresses . . . . .	49

## 1. Introduction

The transport layer supports the end-to-end flow of data across a network path, providing features such as connection establishment, reliability, framing, ordering, congestion control, flow control, etc., as needed to support applications. One of the core functions of an Internet transport is to discover and adapt to the characteristics of the network path that is currently being used.

For some years, it has been common for the transport layer payload to be protected by encryption and authentication, but for the transport layer headers to be sent unprotected. Examples of protocols that behave in this manner include Transport Layer Security (TLS) over TCP [RFC8446], Datagram TLS [RFC6347] [I-D.ietf-tls-dtls13], the Secure Real-time Transport Protocol [RFC3711], and tcpcrypt [RFC8548]. The use of unencrypted transport headers has led some network operators, researchers, and others to develop tools and processes that rely on observations of transport headers both in aggregate and at the flow level to infer details of the network's behaviour and inform operational practice.

Transport protocols are now being developed that encrypt some or all of the transport headers, in addition to the transport payload data. The QUIC transport protocol [I-D.ietf-quic-transport] is an example of such a protocol. Such transport header encryption makes it difficult to observe transport protocol behaviour from the vantage point of the network. This document discusses some implications of transport header encryption for network operators and researchers that have previously observed transport headers, and highlights some issues to consider for transport protocol designers.

As discussed in [RFC7258], the IETF has concluded that Pervasive Monitoring (PM) is a technical attack that needs to be mitigated in the design of IETF protocols. This document supports that conclusion. It also recognises that RFC7258 states "Making networks unmanageable to mitigate PM is not an acceptable outcome, but ignoring PM would go against the consensus documented here. An appropriate balance will emerge over time as real instances of this tension are considered". This document is written to provide input to the discussion around what is an appropriate balance, by highlighting some implications of transport header encryption.

Current uses of transport header information by network devices on the Internet path are explained. These uses can be beneficial or malicious. This is written to provide input to the discussion around what is an appropriate balance, by highlighting some implications of transport header encryption.

## 2. Current uses of Transport Headers within the Network

In response to pervasive monitoring [RFC7624] revelations and the IETF consensus that "Pervasive Monitoring is an Attack" [RFC7258], efforts are underway to increase encryption of Internet traffic. Applying confidentiality to transport header fields can improve privacy, and can help to mitigate certain attacks or manipulation of packets by devices on the network path, but it can also affect network operations and measurement [RFC8404].

When considering what parts of the transport headers should be encrypted to provide confidentiality, and what parts should be visible to network devices (including non-encrypted but authenticated headers), it is necessary to consider both the impact on network operations and management, and the implications for ossification and user privacy [Measurement]. Different parties will view the relative importance of these concerns differently. For some, the benefits of encrypting all the transport headers outweigh the impact of doing so; others might analyse the security, privacy, and ossification impacts and arrive at a different trade-off.

This section reviews examples of the observation of transport layer headers within the network by devices on the network path, or using information exported by an on-path device. Unencrypted transport headers provide information that can support network operations and management, and this section notes some ways in which this has been done. Unencrypted transport header information also contributes metadata that can be exploited for purposes unrelated to network transport measurement, diagnostics or troubleshooting (e.g., to block or to throttle traffic from a specific content provider), and this section also notes some threats relating to unencrypted transport headers.

Exposed transport information also provides a source of information that contributes to linked data sets, which could be exploited to deduce private information, e.g., user patterns, user location, tracking behaviour, etc. This might reveal information the parties did not intend to be revealed. [RFC6973] aims to make designers, implementers, and users of Internet protocols aware of privacy-related design choices in IETF protocols.

This section does not consider intentional modification of transport headers by middleboxes, such as devices performing Network Address Translation (NAT) or Firewalls.

## 2.1. To Separate Flows in Network Devices

Some network layer mechanisms separate network traffic by flow, without resorting to identifying the type of traffic. Hash-based load-sharing sharing across paths (e.g., equal cost multi path, ECMP), sharing across a group of links (e.g., using a link aggregation group, LAG), ensuring equal access to link capacity (e.g., fair queuing, FQ), or distributing traffic to servers (e.g., load balancing). To prevent packet reordering, forwarding engines can consistently forward the same transport flows along the same forwarding path, often achieved by calculating a hash using an n-tuple gleaned from a combination of link header information through to transport header information. This n-tuple can use the MAC address, IP addresses, and can include observable transport header information.

When transport header information cannot be observed, there can be less information to separate flows at equipment along the path. Flow separation might not be possible when, a transport that forms traffic into an encrypted aggregate. For IPv6, the Flow Label [RFC6437] can be used even when all transport information is encrypted, enabling Flow Label-based ECMP [RFC6438] and Load-Sharing [RFC7098].

## 2.2. To Identify Transport Protocols and Flows

Information in exposed transport layer headers can be used by the network to identify transport protocols and flows [RFC8558]. The ability to identify transport protocols, flows, and sessions is a common function performed, for example, by measurement activities, Quality of Service (QoS) classifiers, and firewalls. These functions can be beneficial, and performed with the consent of, and in support of, the end user. Alternatively, the same mechanisms could be used to support practises that might be adversarial to the end user, including blocking, de-prioritising, and monitoring traffic without consent.

Observable transport header information, together with information in the network header, has been used to identify flows and their connection state, together with the set of protocol options being used. Transport protocols, such as TCP [RFC7414] and the Stream Control Transport Protocol (SCTP) [RFC4960], specify a standard base header that includes sequence number information and other data. They also have the possibility to negotiate additional headers at connection setup, identified by an option number in the transport header.

In some uses, an assigned transport port (e.g., 0..49151) can identify the upper-layer protocol or service [RFC7605]. However,



port information alone is not sufficient to guarantee identification. Applications can use arbitrary ports and do not need to use assigned port numbers. The use of an assigned port number is also not limited to the protocol for which the port is intended. Multiple sessions can also be multiplexed on a single port, and ports can be re-used by subsequent sessions.

Some flows can be identified by observing signalling data (e.g., [RFC3261], [RFC8837]) or through the use of magic numbers placed in the first byte(s) of a datagram payload [RFC7983].

When transport header information cannot be observed, this removes information that could have been used to classify flows by passive observers along the path. More ambitious ways could be used to collect, estimate, or infer flow information, including heuristics based on the analysis of traffic patterns, such as classification of flows relying on timing, volumes of information, and correlation between multiple flows. For example, an operator that cannot access the Session Description Protocol (SDP) session descriptions [RFC4566] to classify a flow as audio traffic, might instead use (possibly less-reliable) heuristics to infer that short UDP packets with regular spacing carry audio traffic. Operational practises aimed at inferring transport parameters are out of scope for this document, and are only mentioned here to recognise that encryption does not prevent operators from attempting to apply practises that were used with unencrypted transport headers.

The IAB [RFC8546] have provided a summary of expected implications of increased encryption on network functions that use the observable headers and describe the expected benefits of designs that explicitly declare protocol invariant header information that can be used for this purpose.

### 2.3. To Understand Transport Protocol Performance

This subsection describes use by the network of exposed transport layer headers to understand transport protocol performance and behaviour.

#### 2.3.1. Using Information Derived from Transport Layer Headers

Observable transport headers enable explicit measurement and analysis of protocol performance, and detection of network anomalies at any point along the Internet path. Some operators use passive monitoring to manage their portion of the Internet by characterising the performance of link/network segments. Inferences from transport headers are used to derive performance metrics:

**Traffic Rate and Volume:** Per-application traffic rate and volume measures can be used to characterise the traffic that uses a network segment or the pattern of network usage. Observing the protocol sequence number and packet size offers one way to measure this (e.g., measurements observing counters in periodic reports such as RTCP; or measurements observing protocol sequence numbers in statistical samples of packet flows, or specific control packets, such as those observed at the start and end of a flow).

Measurements can be per endpoint, or for an endpoint aggregate. These could be used to assess usage or for subscriber billing.

Such measurements can be used to trigger traffic shaping, and to associate QoS support within the network and lower layers. This can be done with consent and in support of an end user, to improve quality of service; or could be used by the network to de-prioritise certain flows without user consent.

The traffic rate and volume can be determined providing that the packets belonging to individual flows can be identified, but there might be no additional information about a flow when the transport headers cannot be observed.

**Loss Rate and Loss Pattern:** Flow loss rate can be derived (e.g., from transport sequence numbers or inferred from observing transport protocol interactions) and has been used as a metric for performance assessment and to characterise transport behaviour. Network operators have used the variation in patterns to detect changes in the offered service. Understanding the location and root cause of loss can help an operator determine whether this requires corrective action.

There are various causes of loss, including: corruption of link frames (e.g., due to interference on a radio link), buffering loss (e.g., overflow due to congestion, Active Queue Management, AQM [RFC7567], or inadequate provision following traffic pre-emption), and policing (traffic management [RFC2475]). Understanding flow loss rates requires maintaining per-flow state (flow identification often requires transport layer information) and either observing the increase in sequence numbers in the network or transport headers, or comparing a per-flow packet counter with the number of packets that the flow actually sent. Per-hop loss can also sometimes be monitored at the interface level by devices on the network path, or using in-situ methods operating over a network segment (see Section 3.3).

The pattern of loss can provide insight into the cause of loss. Losses can often occur as bursts, randomly-timed events, etc. It

can also be valuable to understand the conditions under which loss occurs. This usually requires relating loss to the traffic flowing at a network node or segment at the time of loss. Transport header information can help identify cases where loss could have been wrongly identified, or where the transport did not require retransmission of a lost packet.

**Throughput and Goodput:** Throughput is the amount of payload data sent by a flow per time interval. Goodput (the subset of throughput consisting of useful traffic) (see Section 2.5 of [RFC7928] and [RFC5166]) is a measure of useful data exchanged. The throughput of a flow can be determined in the absence of transport header information, providing that the individual flow can be identified, and the overhead known. Goodput requires ability to differentiate loss and retransmission of packets, for example by observing packet sequence numbers in the TCP or RTP headers [RFC3550].

**Latency:** Latency is a key performance metric that impacts application and user-perceived response times. It often indirectly impacts throughput and flow completion time. This determines the reaction time of the transport protocol itself, impacting flow setup, congestion control, loss recovery, and other transport mechanisms. The observed latency can have many components [Latency]. Of these, unnecessary/unwanted queueing in buffers of the network devices on the path has often been observed as a significant factor [bufferbloat]. Once the cause of unwanted latency has been identified, this can often be eliminated.

To measure latency across a part of a path, an observation point [RFC7799] can measure the experienced round trip time (RTT) using packet sequence numbers and acknowledgements, or by observing header timestamp information. Such information allows an observation point on the network path to determine not only the path RTT, but also allows measurement of the upstream and downstream contribution to the RTT. This could be used to locate a source of latency, e.g., by observing cases where the median RTT is much greater than the minimum RTT for a part of a path.

The service offered by network operators can benefit from latency information to understand the impact of configuration changes and to tune deployed services. Latency metrics are key to evaluating and deploying AQM [RFC7567], DiffServ [RFC2474], and Explicit Congestion Notification (ECN) [RFC3168] [RFC8087]. Measurements could identify excessively large buffers, indicating where to deploy or configure AQM. An AQM method is often deployed in combination with other techniques, such as scheduling [RFC7567] [RFC8290] and although parameter-less methods are desired

[RFC7567], current methods often require tuning [RFC8290] [RFC8289] [RFC8033] because they cannot scale across all possible deployment scenarios.

Latency and round-trip time information can potentially expose some information useful for approximate geolocation, as discussed in [PAM-RTT].

**Variation in delay:** Some network applications are sensitive to (small) changes in packet timing (jitter). Short and long-term delay variation can impact on the latency of a flow and hence the perceived quality of applications using a network path. For example, jitter metrics are often cited when characterising paths supporting real-time traffic. The expected performance of such applications, can be inferred from a measure of the variation in delay observed along a portion of the path [RFC3393] [RFC5481]. The requirements resemble those for the measurement of latency.

**Flow Reordering:** Significant packet reordering within a flow can impact time-critical applications and can be interpreted as loss by reliable transports. Many transport protocol techniques are impacted by reordering (e.g., triggering TCP retransmission or re-buffering of real-time applications). Packet reordering can occur for many reasons, from equipment design to misconfiguration of forwarding rules. Flow identification is often required to avoid significant packet mis-ordering (e.g., when using ECMP, or LAG). Network tools can detect and measure unwanted/excessive reordering, and the impact on transport performance.

There have been initiatives in the IETF transport area to reduce the impact of reordering within a transport flow, possibly leading to a reduction in the requirements for preserving ordering. These have potential to simplify network equipment design as well as the potential to improve robustness of the transport service. Measurements of reordering can help understand the present level of reordering, and inform decisions about how to progress new mechanisms.

Techniques for measuring reordering typically observe packet sequence numbers. Metrics have been defined that evaluate whether a network path has maintained packet order on a packet-by-packet basis [RFC4737] [RFC5236]. Some protocols provide in-built monitoring and reporting functions. Transport fields in the RTP header [RFC3550] [RFC4585] can be observed to derive traffic volume measurements and provide information on the progress and quality of a session using RTP. Metadata assists in understanding the context under which the data was collected, including the time, observation point [RFC7799], and way in which metrics were

accumulated. The RTCP protocol directly reports some of this information in a form that can be directly visible by devices on the network path.

In some cases, measurements could involve active injection of test traffic to perform a measurement (see Section 3.4 of [RFC7799]). However, most operators do not have access to user equipment, therefore the point of test is normally different from the transport endpoint. Injection of test traffic can incur an additional cost in running such tests (e.g., the implications of capacity tests in a mobile network segment are obvious). Some active measurements [RFC7799] (e.g., response under load or particular workloads) perturb other traffic, and could require dedicated access to the network segment.

Passive measurements (see Section 3.6 of [RFC7799]) can have advantages in terms of eliminating unproductive test traffic, reducing the influence of test traffic on the overall traffic mix, and the ability to choose the point of observation (see Section 2.4.1). Measurements can rely on observing packet headers, which is not possible if those headers are encrypted, but could utilise information about traffic volumes or patterns of interaction to deduce metrics.

Passive packet sampling techniques are also often used to scale the processing involved in observing packets on high rate links. This exports only the packet header information of (randomly) selected packets. Interpretation of the exported information relies on understanding of the header information. The utility of these measurements depends on the type of network segment/link and number of mechanisms used by the network devices. Simple routers are relatively easy to manage, but a device with more complexity demands understanding of the choice of many system parameters.

#### 2.3.2. Using Information Derived from Network Layer Header Fields

Information from the transport header can be used by a multi-field (MF) classifier as a part of policy framework. Policies are commonly used for management of the QoS or Quality of Experience (QoE) in resource-constrained networks, or by firewalls to implement access rules (see also Section 2.2.2 of [RFC8404]). Policies can support user applications/services or protect against unwanted, or lower priority traffic (Section 2.4.4).

Transport layer information can also be explicitly carried in network-layer header fields that are not encrypted, serving as a replacement/addition to the exposed transport header information [RFC8558]. This information can enable a different forwarding

treatment by the devices forming the network path, even when a transport employs encryption to protect other header information.

On the one hand, the user of a transport that multiplexes multiple sub-flows might want to obscure the presence and characteristics of these sub-flows. On the other hand, an encrypted transport could set the network-layer information to indicate the presence of sub-flows, and to reflect the service requirements of individual sub-flows. There are several ways this could be done:

**IP Address:** Applications normally expose the endpoint addresses used in the forwarding decisions in network devices. Address and other protocol information can be used by a MF-classifier to determine how traffic is treated [RFC2475], and hence affect the quality of experience for a flow. Common issues concerning IP address sharing are described in [RFC6269].

**Using the IPv6 Network-Layer Flow Label:** A number of Standards Track and Best Current Practice RFCs (e.g., [RFC8085], [RFC6437], [RFC6438]) encourage endpoints to set the IPv6 flow label field of the network-layer header. IPv6 "source nodes SHOULD assign each unrelated transport connection and application data stream to a new flow" [RFC6437]. A multiplexing transport could choose to use multiple flow labels to allow the network to independently forward sub-flows. RFC6437 provides further guidance on choosing a flow label value, stating these "should be chosen such that their bits exhibit a high degree of variability", and chosen so that "third parties should be unlikely to be able to guess the next value that a source of flow labels will choose".

Once set, a flow label can provide information that can help inform network-layer queueing and forwarding, including use with IPsec, [RFC6294] and use with Equal Cost Multi-Path routing and Link Aggregation [RFC6438].

The choice of how to assign a flow label needs to avoid introducing linkages between flows that a network device could not otherwise observe. Inappropriate use by the transport can have privacy implications (e.g., assigning the same label to two independent flows that ought not to be classified the same).

**Using the Network-Layer Differentiated Services Code Point:** Applications can expose their delivery expectations to network devices by setting the Differentiated Services Code Point (DSCP) field of IPv4 and IPv6 packets [RFC2474]. For example, WebRTC applications identify different forwarding treatments for individual sub-flows (audio vs. video) based on the value of the DSCP field [I-D.ietf-tsvwg-rtcweb-qos]). This provides explicit

information to inform network-layer queueing and forwarding, rather than an operator inferring traffic requirements from transport and application headers via a multi-field classifier. Inappropriate use by the transport can have privacy implications (e.g., assigning a different DSCP to a subflow could assist in a network device discovering the traffic pattern used by an application). The field is mutable, i.e., some network devices can be expected to change this field. Since the DSCP value can impact the quality of experience for a flow, observations of service performance have to consider this field when a network path supports differentiated service treatment.

**Using Explicit Congestion Marking:** ECN [RFC3168] is a transport mechanism that uses the ECN field in the network-layer header. Use of ECN explicitly informs the network-layer that a transport is ECN-capable, and requests ECN treatment of the flow. An ECN-capable transport can offer benefits when used over a path with equipment that implements an AQM method with CE marking of IP packets [RFC8087], since it can react to congestion without also having to recover from lost packets.

ECN exposes the presence of congestion. The reception of CE-marked packets can be used to estimate the level of incipient congestion on the upstream portion of the path from the point of observation (Section 2.5 of [RFC8087]). Interpreting the marking behaviour (i.e., assessing congestion and diagnosing faults) requires context from the transport layer, such as path RTT.

AQM and ECN offer a range of algorithms and configuration options. Tools therefore have to be available to network operators and researchers to understand the implication of configuration choices and transport behaviour as the use of ECN increases and new methods emerge [RFC7567].

**Network-Layer Options** Network protocols can carry optional headers (see Section 5.1). These can explicitly expose transport header information to on-path devices operating at the network layer (as discussed further in Section 6).

IPv4 [RFC0791] has provision for optional header fields. IP routers can examine these headers and are required to ignore IPv4 options that they do not recognise. Many current paths include network devices that forward packets that carry options on a slower processing path. Some network devices (e.g., firewalls) can be (and are) configured to drop these packets [RFC7126]. BCP 186 [RFC7126] provides Best Current Practice guidance on how operators should treat IPv4 packets that specify options.

IPv6 can encode optional network-layer information in separate headers that may be placed between the IPv6 header and the upper-layer header [RFC8200]. (e.g., the IPv6 Alternate Marking Method [I-D.ietf-6man-ipv6-alt-mark], which can be used to measure packet loss and delay metrics). The Hop-by-Hop options header, when present, immediately follows the IPv6 header. IPv6 permits this header to be examined by any node along the path if explicitly configured [RFC8200].

Careful use of the network layer features (e.g., Extension Headers can Section 5) help provide similar information in the case where the network is unable to inspect transport protocol headers.

#### 2.4. To Support Network Operations

Some network operators make use of on-path observations of transport headers to analyse the service offered to the users of a network segment, and to inform operational practice, and can help detect and locate network problems. [RFC8517] gives an operator's perspective about such use.

When observable transport header information is not available, those seeking an understanding of transport behaviour and dynamics might learn to work without that information. Alternatively, they might use more limited measurements combined with pattern inference and other heuristics to infer network behaviour (see Section 2.1.1 of [RFC8404]). Operational practises aimed at inferring transport parameters are out of scope for this document, and are only mentioned here to recognise that encryption does not necessarily stop operators from attempting to apply practises that have been used with unencrypted transport headers.

This section discusses topics concerning observation of transport flows, with a focus on transport measurement.

##### 2.4.1. Problem Location

Observations of transport header information can be used to locate the source of problems or to assess the performance of a network segment. Often issues can only be understood in the context of the other flows that share a particular path, particular device configuration, interface port, etc. A simple example is monitoring of a network device that uses a scheduler or active queue management technique [RFC7567], where it could be desirable to understand whether the algorithms are correctly controlling latency, or if overload protection is working. This implies knowledge of how traffic is assigned to any sub-queues used for flow scheduling, but can require information about how the traffic dynamics impact active



queue management, starvation prevention mechanisms, and circuit-breakers.

Sometimes correlating observations of headers at multiple points along the path (e.g., at the ingress and egress of a network segment), allows an observer to determine the contribution of a portion of the path to an observed metric. e.g., to locate a source of delay, jitter, loss, reordering, or congestion marking.

#### 2.4.2. Network Planning and Provisioning

Traffic rate and volume measurements are used to help plan deployment of new equipment and configuration in networks. Data is also valuable to equipment vendors who want to understand traffic trends and patterns of usage as inputs to decisions about planning products and provisioning for new deployments.

Trends in aggregate traffic can be observed and can be related to the endpoint addresses being used, but when transport header information is not observable, it might be impossible to correlate patterns in measurements with changes in transport protocols. This increases the dependency on other indirect sources of information to inform planning and provisioning.

#### 2.4.3. Compliance with Congestion Control

The traffic that can be observed by on-path network devices (the "wire image") is a function of transport protocol design/options, network use, applications, and user characteristics. In general, when only a small proportion of the traffic has a specific (different) characteristic, such traffic seldom leads to operational concern, although the ability to measure and monitor it is lower. The desire to understand the traffic and protocol interactions typically grows as the proportion of traffic increases. The challenges increase when multiple instances of an evolving protocol contribute to the traffic that share network capacity.

Operators can manage traffic load (e.g., when the network is severely overloaded) by deploying rate-limiters, traffic shaping, or network transport circuit breakers [RFC8084]. The information provided by observing transport headers is a source of data that can help to inform such mechanisms.

**Congestion Control Compliance of Traffic:** Congestion control is a key transport function [RFC2914]. Many network operators implicitly accept that TCP traffic complies with a behaviour that is acceptable for the shared Internet. TCP algorithms have been continuously improved over decades, and have reached a level of

efficiency and correctness that is difficult to match in custom application-layer mechanisms [RFC8085].

A standards-compliant TCP stack provides congestion control that is judged safe for use across the Internet. Applications developed on top of well-designed transports can be expected to appropriately control their network usage, reacting when the network experiences congestion, by back-off and reduce the load placed on the network. This is the normal expected behaviour for IETF-specified transports (e.g., TCP and SCTP).

Congestion Control Compliance for UDP traffic: UDP provides a minimal message-passing datagram transport that has no inherent congestion control mechanisms. Because congestion control is critical to the stable operation of the Internet, applications and other protocols that choose to use UDP as a transport have to employ mechanisms to prevent collapse, avoid unacceptable contributions to jitter/latency, and to establish an acceptable share of capacity with concurrent traffic [RFC8085].

UDP flows that expose a well-known header can be observed to gain understanding of the dynamics of a flow and its congestion control behaviour. For example, tools exist to monitor various aspects of RTP header information and RTCP reports for real-time flows (see Section 2.3). The Secure RTP and RTCP extensions [RFC3711] were explicitly designed to expose some header information to enable such observation, while protecting the payload data.

A network operator can observe the headers of transport protocols layered above UDP to understand if the datagram flows comply with congestion control expectations. This can help inform a decision on whether it might be appropriate to deploy methods such as rate-limiters to enforce acceptable usage. The available information determines the level of precision with which flows can be classified and the design space for conditioning mechanisms (e.g., rate limiting, circuit breaker techniques [RFC8084], or blocking of uncharacterised traffic) [RFC5218].

When anomalies are detected, tools can interpret the transport header information to help understand the impact of specific transport protocols (or protocol mechanisms) on the other traffic that shares a network. An observer on the network path can gain an understanding of the dynamics of a flow and its congestion control behaviour. Analysing observed flows can help to build confidence that an application flow backs-off its share of the network load under persistent congestion, and hence to understand whether the behaviour is appropriate for sharing limited network capacity. For example, it is common to visualise plots of TCP sequence numbers versus time for

a flow to understand how a flow shares available capacity, deduce its dynamics in response to congestion, etc.

The ability to identify sources and flows that contribute to persistent congestion is important to the safe operation of network infrastructure, and can inform configuration of network devices to complement the endpoint congestion avoidance mechanisms [RFC7567] [RFC8084] to avoid a portion of the network being driven into congestion collapse [RFC2914].

#### 2.4.4. To Characterise "Unknown" Network Traffic

The patterns and types of traffic that share Internet capacity change over time as networked applications, usage patterns and protocols continue to evolve.

Encryption can increase the volume of "unknown" or "uncharacterised" traffic seen by the network. If these traffic patterns form a small part of the traffic aggregate passing through a network device or segment of the network path, the dynamics of the uncharacterised traffic might not have a significant collateral impact on the performance of other traffic that shares this network segment. Once the proportion of this traffic increases, monitoring the traffic can determine if appropriate safety measures have to be put in place.

Tracking the impact of new mechanisms and protocols requires traffic volume to be measured and new transport behaviours to be identified. This is especially true of protocols operating over a UDP substrate. The level and style of encryption needs to be considered in determining how this activity is performed.

Traffic that cannot be classified typically receives a default treatment. Some networks block or rate-limit traffic that cannot be classified.

#### 2.4.5. To Support Network Security Functions

On-path observation of the transport headers of packets can be used for various security functions. For example, Denial of Service (DoS) and Distributed DoS (DDoS) attacks against the infrastructure or against an endpoint can be detected and mitigated by characterising anomalous traffic (see Section 2.4.4) on a shorter timescale. Other uses include support for security audits (e.g., verifying the compliance with cipher suites), client and application fingerprinting for inventory, and to provide alerts for network intrusion detection and other next generation firewall functions.

When using an encrypted transport, endpoints can directly provide information to support these security functions. Another method, if the endpoints do not provide this information, is to use an on-path network device that relies on pattern inferences in the traffic, and heuristics or machine learning instead of processing observed header information. An endpoint could also explicitly cooperate with an on-path device (e.g., a QUIC endpoint could share information about current uses of connection IDs).

#### 2.4.6. Network Diagnostics and Troubleshooting

Operators monitor the health of a network segment to support a variety of operational tasks [RFC8404] including procedures to provide early warning and trigger action: to diagnose network problems, to manage security threats (including DoS), to evaluate equipment or protocol performance, or to respond to user performance questions. Information about transport flows can assist in setting buffer sizes, and help identify whether link/network tuning is effective. Information can also support debugging and diagnosis of the root causes of faults that concern a particular user's traffic and can support post-mortem investigation after an anomaly. Section 3.1.2 and Section 5 of [RFC8404] provide further examples.

Network segments vary in their complexity. The design trade-offs for radio networks are often very different from those of wired networks [RFC8462]. A radio-based network (e.g., cellular mobile, enterprise Wireless LAN (WLAN), satellite access/back-haul, point-to-point radio) adds a subsystem that performs radio resource management, with impact on the available capacity, and potentially loss/reordering of packets. This impact can differ by traffic type, and can be correlated with link propagation and interference. These can impact the cost and performance of a provided service, and is expected to increase in importance as operators bring together heterogeneous types of network equipment and deploy opportunistic methods to access shared radio spectrum.

#### 2.4.7. Tooling and Network Operations

A variety of open source and proprietary tools have been deployed that use the transport header information observable with widely used protocols such as TCP or RTP/UDP/IP. Tools that dissect network traffic flows can alert to potential problems that are hard to derive from volume measurements, link statistics or device measurements alone.

Any introduction of a new transport protocol, protocol feature, or application might require changes to such tools, and so could impact operational practice and policies. Such changes have associated

costs that are incurred by the network operators that need to update their tooling or develop alternative practises that work without access to the changed/removed information.

The use of encryption has the desirable effect of preventing unintended observation of the payload data and these tools seldom seek to observe the payload, or other application details. A flow that hides its transport header information could imply "don't touch" to some operators. This might limit a trouble-shooting response to "can't help, no trouble found".

An alternative that does not require access to observable transport headers is to access endpoint diagnostic tools or to include user involvement in diagnosing and troubleshooting unusual use cases or to troubleshoot non-trivial problems. Another approach is to use traffic pattern analysis. Such tools can provide useful information during network anomalies (e.g., detecting significant reordering, high or intermittent loss), however indirect measurements need to be carefully designed to provide information for diagnostics and troubleshooting.

If new protocols, or protocol extensions, are made to closely resemble or match existing mechanisms, then the changes to tooling and the associated costs can be small. Equally, more extensive changes to the transport tend to require more extensive, and more expensive, changes to tooling and operational practice. Protocol designers can mitigate these costs by explicitly choosing to expose selected information as invariants that are guaranteed not to change for a particular protocol (e.g., the header invariants and the spin-bit in QUIC [I-D.ietf-quic-transport]). Specification of common log formats and development of alternative approaches can also help mitigate the costs of transport changes.

## 2.5. To Mitigate the Effects of Constrained Networks

Some link and network segments are constrained by the capacity they can offer, by the time it takes to access capacity (e.g., due to under-lying radio resource management methods), or by asymmetries in the design (e.g., many link are designed so that the capacity available is different in the forward and return directions; some radio technologies have different access methods in the forward and return directions resulting from differences in the power budget).

The impact of path constraints can be mitigated using a proxy operating at or above the transport layer to use an alternate transport protocol.

In many cases, one or both endpoints are unaware of the characteristics of the constraining link or network segment and mitigations are applied below the transport layer: Packet classification and QoS methods (described in various sections) can be beneficial in differentially prioritising certain traffic when there is a capacity constraint or additional delay in scheduling link transmissions. Another common mitigation is to apply header compression over the specific link or subnetwork (see Section 2.5.1).

#### 2.5.1. To Provide Header Compression

Header compression saves link capacity by compressing network and transport protocol headers on a per-hop basis. This has been widely used with low bandwidth dial-up access links, and still finds application on wireless links that are subject to capacity constraints. These methods are effective for bit-congestive links sending small packets (e.g., reducing the cost for sending control packets or small data packets over radio links).

Examples of header compression include use with TCP/IP and RTP/UDP/IP flows [RFC2507], [RFC6846], [RFC2508], [RFC5795], [RFC8724]. Successful compression depends on observing the transport headers and understanding of the way fields change between packets, and is hence incompatible with header encryption. Devices that compress transport headers are dependent on a stable header format, implying ossification of that format.

Introducing a new transport protocol, or changing the format of the transport header information, will limit the effectiveness of header compression until the network devices are updated. Encrypting the transport protocol headers will tend to cause the header compression to fall back to compressing only the network layer headers, with a significant reduction in efficiency. This can limit connectivity if the resulting flow exceeds the link capacity, or if the packets are dropped because they exceed the link MTU.

The Secure RTP (SRTP) extensions [RFC3711] were explicitly designed to leave the transport protocol headers unencrypted, but authenticated, since support for header compression was considered important.

#### 2.6. To Verify SLA Compliance

Observable transport headers coupled with published transport specifications allow operators and regulators to explore and verify compliance with Service Level Agreements (SLAs). It can also be used to understand whether a service is providing differential treatment to certain flows.

When transport header information cannot be observed, other methods have to be found to confirm that the traffic produced conforms to the expectations of the operator or developer.

Independently verifiable performance metrics can be utilised to demonstrate regulatory compliance in some jurisdictions, and as a basis for informing design decisions. This can bring assurance to those operating networks, often avoiding deployment of complex techniques that routinely monitor and manage Internet traffic flows (e.g., avoiding the capital and operational costs of deploying flow rate-limiting and network circuit-breaker methods [RFC8084]).

### 3. Research, Development and Deployment

Research and development of new protocols and mechanisms need to be informed by measurement data (as described in the previous section). Data can also help promote acceptance of proposed standards specifications by the wider community (e.g., as a method to judge the safety for Internet deployment).

Observed data is important to ensure the health of the research and development communities, and provides data needed to evaluate new proposals for standardisation. Open standards motivate a desire to include independent observation and evaluation of performance and deployment data. Independent data helps compare different methods, judge the level of deployment and ensure the wider applicability of the results. This is important when considering when a protocol or mechanism should be standardised for use in the general Internet. This, in turn, demands control/understanding about where and when measurement samples are collected. This requires consideration of the methods used to observe information and the appropriate balance between encrypting all and no transport header information.

There can be performance and operational trade-offs in exposing selected information to network tools. This section explores key implications of tools and procedures that observe transport protocols, but does not endorse or condemn any specific practises.

#### 3.1. Independent Measurement

Encrypting transport header information has implications on the way network data is collected and analysed. Independent observation by multiple actors is currently used by the transport community to maintain an accurate understanding of the network within transport area working groups, IRTF research groups, and the broader research community. This is important to be able to provide accountability, and demonstrate that protocols behave as intended, although when providing or using such information, it is important to consider the

privacy of the user and their incentive for providing accurate and detailed information.

Protocols that expose the state of the transport protocol in their header (e.g., timestamps used to calculate the RTT, packet numbers used to assess congestion and requests for retransmission) provide an incentive for a sending endpoint to provide consistent information, because a protocol will not work otherwise. An on-path observer can have confidence that well-known (and ossified) transport header information represents the actual state of the endpoints, when this information is necessary for the protocol's correct operation.

Encryption of transport header information could reduce the range of actors that can observe useful data. This would limit the information sources available to the Internet community to understand the operation of new transport protocols, reducing information to inform design decisions and standardisation of the new protocols and related operational practises. The cooperating dependence of network, application, and host to provide communication performance on the Internet is uncertain when only endpoints (i.e., at user devices and within service platforms) can observe performance, and when performance cannot be independently verified by all parties.

### 3.2. Measurable Transport Protocols

Transport protocol evolution, and the ability to measure and understand the impact of protocol changes, have to proceed hand-in-hand. A transport protocol that provides observable headers can be used to provide open and verifiable measurement data. Observation of pathologies has a critical role in the design of transport protocol mechanisms and development of new mechanisms and protocols, and aides understanding of the interactions between cooperating protocols and network mechanisms, the implications of sharing capacity with other traffic and the impact of different patterns of usage. The ability of other stakeholders to review transport header traces helps develop insight into the performance and the traffic contribution of specific variants of a protocol.

Development of new transport protocol mechanisms has to consider the scale of deployment and the range of environments in which the transport is used. Experience has shown that it is often difficult to correctly implement new mechanisms [RFC8085], and that mechanisms often evolve as a protocol matures, or in response to changes in network conditions, changes in network traffic, or changes to application usage. Analysis is especially valuable when based on the behaviour experienced across a range of topologies, vendor equipment, and traffic patterns.



Encryption enables a transport protocol to choose which internal state to reveal to devices on the network path, what information to encrypt, and what fields to grease [RFC8701]. A new design can provide summary information regarding its performance, congestion control state, etc., or to make available explicit measurement information. For example, [I-D.ietf-quic-transport] specifies a way for a QUIC endpoint to optionally set the spin-bit to explicitly reveal the RTT of an encrypted transport session to the on-path network devices. There is a choice of what information to expose. For some operational uses, the information has to contain sufficient detail to understand, and possibly reconstruct, the network traffic pattern for further testing. The interpretation of the information needs to consider whether this information reflects the actual transport state of the endpoints. This might require the trust of transport protocol implementers, to correctly reveal the desired information.

New transport protocol formats are expected to facilitate an increased pace of transport evolution, and with it the possibility to experiment with and deploy a wide range of protocol mechanisms. At the time of writing, there has been interest in a wide range of new transport methods, e.g., Larger Initial Window, Proportional Rate Reduction (PRR), congestion control methods based on measuring bottleneck bandwidth and round-trip propagation time, the introduction of AQM techniques and new forms of ECN response (e.g., Data Centre TCP, DCTCP, and methods proposed for L4S). The growth and diversity of applications and protocols using the Internet also continues to expand. For each new method or application, it is desirable to build a body of data reflecting its behaviour under a wide range of deployment scenarios, traffic load, and interactions with other deployed/candidate methods.

### 3.3. Other Sources of Information

Some measurements that traditionally rely on observable transport information could be completed by utilising endpoint-based logging (e.g., based on Quic-Trace [Quic-Trace] and qlog [I-D.marx-qlog-main-schema]). Such information has a diversity of uses, including developers wishing to debug/understand the transport/application protocols with which they work, researchers seeking to spot trends and anomalies, and to characterise variants of protocols. A standard format for endpoint logging could allow these to be shared (after appropriate anonymisation) to understand performance and pathologies.

When measurement datasets are made available by servers or client endpoints, additional metadata, such as the state of the network and conditions in which the system was observed, is often necessary to

interpret this data to answer questions about network performance or understand a pathology. Collecting and coordinating such metadata is more difficult when the observation point is at a different location to the bottleneck or device under evaluation [RFC7799].

Despite being applicable in some scenarios, endpoint logs do not provide equivalent information to on-path measurements made by devices in the network. In particular, endpoint logs contain only a part of the information to understand the operation of network devices and identify issues such as link performance or capacity sharing between multiple flows. An analysis can require coordination between actors at different layers to successfully characterise flows and correlate the performance or behaviour of a specific mechanism with an equipment configuration and traffic using operational equipment along a network path (e.g., combining transport and network measurements to explore congestion control dynamics, to understand the implications of traffic on designs for active queue management or circuit breakers).

Another source of information could arise from operations, administration and management (OAM) (see Section 6) information data records could be embedded into header information at different layers to support functions such as performance evaluation, path-tracing, path verification information, classification and a diversity of other uses.

In-situ OAM (IOAM) data fields [I-D.ietf-ippm-ioam-data] can be encapsulated into a variety of protocols to record operational and telemetry information in an existing packet, while that packet traverses a part of the path between two points in a network (e.g., within a particular IOAM management domain). The IOAM-Data-Fields are independent from the protocols into which the IOAM-Data-Fields are encapsulated. For example, IOAM can provide proof that a certain traffic flow takes a pre-defined path, SLA verification for the live data traffic, and statistics relating to traffic distribution.

#### 4. Encryption and Authentication of Transport Headers

There are several motivations for transport header encryption.

One motive to encrypt transport headers is to prevent network ossification from network devices that inspect well-known transport headers. Once a network device observes a transport header and becomes reliant upon using it, the overall use of that field can become ossified, preventing new versions of the protocol and mechanisms from being deployed. Examples include:

- o During the development of TLS 1.3 [RFC8446], the design needed to function in the presence of deployed middleboxes that relied on the presence of certain header fields exposed in TLS 1.2 [RFC5426].
- o The design of Multipath TCP (MPTCP) [RFC8684] had to account for middleboxes (known as "TCP Normalizers") that monitor the evolution of the window advertised in the TCP header and then reset connections when the window did not grow as expected.
- o TCP Fast Open [RFC7413] can experience problems due to middleboxes that modify the transport header of packets by removing "unknown" TCP options. Segments with unrecognised TCP options can be dropped, segments that contain data and set the SYN bit can be dropped, and some middleboxes that disrupt connections that send data before completion of the three-way handshake.
- o Other examples of TCP ossification have included middleboxes that modify transport headers by rewriting TCP sequence and acknowledgement numbers, but are unaware of the (newer) TCP selective acknowledgement (SACK) option and therefore fail to correctly rewrite the SACK information to match the changes made to the fixed TCP header, preventing correct SACK operation.

In all these cases, middleboxes with a hard-coded, but incomplete, understanding of a specific transport behaviour (i.e., TCP), interacted poorly with transport protocols after the transport behaviour was changed. In some cases, the middleboxes modified or replaced information in the transport protocol header.

Transport header encryption prevents an on-path device from observing the transport headers, and therefore stops ossified mechanisms being used that directly rely on or infer semantics of the transport header information. This encryption is normally combined with authentication of the protected information. RFC 8546 summarises this approach, stating that it is "The wire image, not the protocol's specification, determines how third parties on the network paths among protocol participants will interact with that protocol" (Section 1 of [RFC8546]), and it can be expected that header information that is not encrypted will become ossified.

Encryption does not itself prevent ossification of the network service. People seeking to understand or classify network traffic could still come to rely on pattern inferences and other heuristics or machine learning to derive measurement data and as the basis for network forwarding decisions [RFC8546]. This can also create dependencies on the transport protocol, or the patterns of traffic it can generate, also resulting in ossification of the service.

Another motivation for using transport header encryption is to improve privacy and to decrease opportunities for surveillance. Users value the ability to protect their identity and location, and defend against analysis of the traffic. Revelations about the use of pervasive surveillance [RFC7624] have, to some extent, eroded trust in the service offered by network operators and have led to an increased use of encryption. Concerns have also been voiced about the addition of metadata to packets by third parties to provide analytics, customisation, advertising, cross-site tracking of users, to bill the customer, or to selectively allow or block content.

Whatever the reasons, the IETF is designing protocols that include transport header encryption (e.g., QUIC [I-D.ietf-quic-transport]) to supplement the already widespread payload encryption, and to further limit exposure of transport metadata to the network.

If a transport protocol uses header encryption, the designers have to decide whether to encrypt all, or a part of, the transport layer information. Section 4 of [RFC8558] states: "Anything exposed to the path should be done with the intent that it be used by the network elements on the path".

Certain transport header fields can be made observable to on-path network devices, or can define new fields designed to explicitly expose observable transport layer information to the network. Where exposed fields are intended to be immutable (i.e., can be observed, but not modified by a network device), the endpoints are encouraged to use authentication to provide a cryptographic integrity check that can detect if these immutable fields have been modified by network devices. Authentication can help to prevent attacks that rely on sending packets that fake exposed control signals in transport headers (e.g., TCP RST spoofing). Making a part of a transport header observable or exposing new header fields can lead to ossification of that part of a header as network devices come to rely on observations of the exposed fields.

The use of transport header authentication and encryption therefore exposes a tussle between middlebox vendors, operators, researchers, applications developers, and end-users:

- o On the one hand, future Internet protocols that support transport header encryption assist in the restoration of the end-to-end nature of the Internet by returning complex processing to the endpoints. Since middleboxes cannot modify what they cannot see, the use of transport header encryption can improve application and end-user privacy by reducing leakage of transport metadata to operators that deploy middleboxes.

- o On the other hand, encryption of transport layer information has implications for network operators and researchers seeking to understand the dynamics of protocols and traffic patterns, since it reduces the information that is available to them.

The following briefly reviews some security design options for transport protocols. A Survey of the Interaction between Security Protocols and Transport Services [RFC8922] provides more details concerning commonly used encryption methods at the transport layer.

Security work typically employs a design technique that seeks to expose only what is needed [RFC3552]. This approach provides incentives to not reveal any information that is not necessary for the end-to-end communication. The IETF has provided guidelines for writing Security Considerations for IETF specifications [RFC3552].

Endpoint design choices impacting privacy also need to be considered as a part of the design process [RFC6973]. The IAB has provided guidance for analyzing and documenting privacy considerations within IETF specifications [RFC6973].

**Authenticating the Transport Protocol Header:** Transport layer header information can be authenticated. An example transport authentication mechanism is TCP-Authentication (TCP-AO) [RFC5925]. This TCP option authenticates the IP pseudo header, TCP header, and TCP data. TCP-AO protects the transport layer, preventing attacks from disabling the TCP connection itself and provides replay protection. Such authentication might interact with middleboxes, depending on their behaviour [RFC3234].

The IPsec Authentication Header (AH) [RFC4302] was designed to work at the network layer and authenticate the IP payload. This approach authenticates all transport headers, and verifies their integrity at the receiver, preventing modification by network devices on the path. The IPsec Encapsulating Security Payload (ESP) [RFC4303] can also provide authentication and integrity without confidentiality using the NULL encryption algorithm [RFC2410]. SRTP [RFC3711] is another example of a transport protocol that allows header authentication.

**Integrity Check** Transport protocols usually employ integrity checks on the transport header information. Security method usually employ stronger checks and can combine this with authentication. An integrity check that protects the immutable transport header fields, but can still expose the transport header information in the clear, allows on-path network devices to observe these fields. An integrity check is not able to prevent modification by network devices on the path, but can prevent a receiving endpoint from

accepting changes and avoid impact on the transport protocol operation, including some types of attack.

**Selectively Encrypting Transport Headers and Payload:** A transport protocol design that encrypts selected header fields, allows specific transport header fields to be made observable by network devices on the path. This information is explicitly exposed either in a transport header field or lower layer protocol header. A design that only exposes immutable fields can also perform end-to-end authentication of these fields across the path to prevent undetected modification of the immutable transport headers.

Mutable fields in the transport header provide opportunities where on-path network devices can modify the transport behaviour (e.g., the extended headers described in [I-D.trammell-plus-abstract-mech]). An example of a method that encrypts some, but not all, transport header information is GRE-in-UDP [RFC8086] when used with GRE encryption.

**Optional Encryption of Header Information:** There are implications to the use of optional header encryption in the design of a transport protocol, where support of optional mechanisms can increase the complexity of the protocol and its implementation, and in the management decisions that have to be made to use variable format fields. Instead, fields of a specific type ought to be sent with the same level of confidentiality or integrity protection.

**Greasing:** Protocols often provide extensibility features, reserving fields or values for use by future versions of a specification. The specification of receivers has traditionally ignored unspecified values, however on-path network devices have emerged that ossify to require a certain value in a field, or re-use a field for another purpose. When the specification is later updated, it is impossible to deploy the new use of the field, and forwarding of the protocol could even become conditional on a specific header field value.

A protocol can intentionally vary the value, format, and/or presence of observable transport header fields at random [RFC8701]. This prevents a network device ossifying the use of a specific observable field and can ease future deployment of new uses of the value or code-point. This is not a security mechanism, although the use can be combined with an authentication mechanism.

Different transports use encryption to protect their header information to varying degrees. The trend is towards increased protection.

## 5. Intentionally Exposing Transport Information to the Network

A transport protocol can choose to expose certain transport information to on-path devices operating at the network layer by sending observable fields. One approach is to make an explicit choice not to encrypt certain transport header fields, making this transport information observable by an on-path network device. Another approach is to expose transport information in a network-layer extension header (see Section 5.1). Both are examples of explicit information intended to be used by network devices on the path [RFC8558].

Whatever the mechanism used to expose the information, a decision to expose only specific information places the transport endpoint in control of what to expose outside of the encrypted transport header. This decision can then be made independently of the transport protocol functionality. This can be done by exposing part of the transport header or as a network layer option/extension.

### 5.1. Exposing Transport Information in Extension Headers

At the network-layer, packets can carry optional headers that explicitly expose transport header information to the on-path devices operating at the network layer (Section 2.3.2). For example, an endpoint that sends an IPv6 Hop-by-Hop option [RFC8200] can provide explicit transport layer information that can be observed and used by network devices on the path. New hop-by-hop options are not recommended in RFC 8200 [RFC8200] "because nodes may be configured to ignore the Hop-by-Hop Options header, drop packets containing a Hop-by-Hop Options header, or assign packets containing a Hop-by-Hop Options header to a slow processing path. Designers considering defining new hop-by-hop options need to be aware of this likely behavior."

Network-layer optional headers explicitly indicate the information that is exposed, whereas use of exposed transport header information first requires an observer to identify the transport protocol and its format. (See Section 2.2.)

An arbitrary path can include one or more network devices that drop packets that include a specific header or option used for this purpose (see [RFC7872]). This could impact the proper functioning of the protocols using the path. Protocol methods can be designed to probe to discover whether the specific option(s) can be used along the current path, enabling use on arbitrary paths.

## 5.2. Common Exposed Transport Information

There are opportunities for multiple transport protocols to consistently supply common observable information [RFC8558]. A common approach can result in an open definition of the observable fields. This has the potential that the same information can be utilised across a range of operational and analysis tools.

## 5.3. Considerations for Exposing Transport Information

Considerations concerning what information, if any, it is appropriate to expose include:

- o On the one hand, explicitly exposing derived fields containing relevant transport information (e.g., metrics for loss, latency, etc) can avoid network devices needing to derive this information from other header fields. This could result in development and evolution of transport-independent tools around a common observable header, and permit transport protocols to also evolve independently of this ossified header [RFC8558].
- o On the other hand, protocols and implementations might be designed to avoid consistently exposing external information that corresponds to the actual internal information used by the protocol itself. An endpoint/protocol could choose to expose transport header information to optimise the benefit it gets from the network [RFC8558]. The value of this information for analysing operation of the transport layer would be enhanced if the exposed information could be verified to match the transport protocol's observed behavior.

The motivation to include actual transport header information and the implications of network devices using this information has to be considered when proposing such a method. RFC 8558 summarises this as "When signals from endpoints to the path are independent from the signals used by endpoints to manage the flow's state mechanics, they may be falsified by an endpoint without affecting the peer's understanding of the flow's state. For encrypted flows, this divergence is not detectable by on-path devices [RFC8558].

## 6. Addition of Transport OAM Information to Network-Layer Headers

Even when the transport headers are encrypted, on-path devices can make measurements by utilising additional protocol headers carrying OAM information in an additional packet header. OAM information can be included with packets to perform functions such as identification of transport protocols and flows, to aide understanding of network or



transport performance, or to support network operations or mitigate the effects of specific network segments.

Using network-layer approaches to reveal information has the potential that the same method (and hence same observation and analysis tools) can be consistently used by multiple transport protocols. This approach also could be applied to methods beyond OAM (see Section 5). There can also be less desirable implications from separating the operation of the transport protocol from the measurement framework.

#### 6.1. Use of OAM within a Maintenance Domain

OAM information can be restricted to a maintenance domain, typically owned and operated by a single entity. OAM information can be added at the ingress to the maintenance domain (e.g., an Ethernet protocol header with timestamps and sequence number information using a method such as 802.11ag or in-situ OAM [I-D.ietf-ippm-ioam-data], or as a part of the encapsulation protocol). This additional header information is not delivered to the endpoints and is typically removed at the egress of the maintenance domain.

Although some types of measurements are supported, this approach does not cover the entire range of measurements described in this document. In some cases, it can be difficult to position measurement tools at the appropriate segments/nodes and there can be challenges in correlating the downstream/upstream information when in-band OAM data is inserted by an on-path device.

#### 6.2. Use of OAM across Multiple Maintenance Domains

OAM information can also be added at the network layer by the sender as an IPv6 extension header or an IPv4 option, or in an encapsulation/tunnel header that also includes an extension header or option. This information can be used across multiple network segments, or between the transport endpoints.

One example is the IPv6 Performance and Diagnostic Metrics (PDM) destination option [RFC8250]. This allows a sender to optionally include a destination option that carries header fields that can be used to observe timestamps and packet sequence numbers. This information could be authenticated by a receiving transport endpoint when the information is added at the sender and visible at the receiving endpoint, although methods to do this have not currently been proposed. This needs to be explicitly enabled at the sender.

## 7. Conclusions

Header encryption and strong integrity checks are being incorporated into new transport protocols and have important benefits. The pace of development of transports using the WebRTC data channel, and the rapid deployment of the QUIC transport protocol, can both be attributed to using the combination of UDP as a substrate while providing confidentiality and authentication of the encapsulated transport headers and payload.

This document has described some current practises, and the implications for some stakeholders, when transport layer header encryption is used. It does not judge whether these practises are necessary, or endorse the use of any specific practise. Rather, the intent is to highlight operational tools and practises to consider when designing and modifying transport protocols, so protocol designers can make informed choices about what transport header fields to encrypt, and whether it might be beneficial to make an explicit choice to expose certain fields to devices on the network path. In making such a decision, it is important to balance:

- o User Privacy: The less transport header information that is exposed to the network, the lower the risk of leaking metadata that might have user privacy implications. Transports that chose to expose some header fields need to make a privacy assessment to understand the privacy cost versus benefit trade-off in making that information available. The design of the QUIC spin bit to the network is an example of such considered analysis.
- o Transport Ossification: Unencrypted transport header fields are likely to ossify rapidly, as network devices come to rely on their presence, making it difficult to change the transport in future. This argues that the choice to expose information to the network is made deliberately and with care, since it is essentially defining a stable interface between the transport and the network. Some protocols will want to make that interface as limited as possible; other protocols might find value in exposing certain information to signal to the network, or in allowing the network to change certain header fields as signals to the transport. The visible wire image of a protocol should be explicitly designed.
- o Network Ossification: While encryption can reduce ossification of the transport protocol, it does not itself prevent ossification of the network service. People seeking to understand network traffic could still come to rely on pattern inferences and other heuristics or machine learning to derive measurement data and as the basis for network forwarding decisions [RFC8546]. This

creates dependencies on the transport protocol, or the patterns of traffic it can generate, resulting in ossification of the service.

- o **Impact on Operational Practice:** The network operations community has long relied on being able to understand Internet traffic patterns, both in aggregate and at the flow level, to support network management, traffic engineering, and troubleshooting. Operational practice has developed based on the information available from unencrypted transport headers. The IETF has supported this practice by developing operations and management specifications, interface specifications, and associated Best Current Practises. Widespread deployment of transport protocols that encrypt their information will impact network operations, unless operators can develop alternative practises that work without access to the transport header.
- o **Pace of Evolution:** Removing obstacles to change can enable an increased pace of evolution. If a protocol changes its transport header format (wire image), or its transport behaviour, this can result in the currently deployed tools and methods becoming no longer relevant. Where this needs to be accompanied by development of appropriate operational support functions and procedures, it can incur a cost in new tooling to catch-up with each change. Protocols that consistently expose observable data do not require such development, but can suffer from ossification and need to consider if the exposed protocol metadata has privacy implications. There is no single deployment context, and therefore designers need to consider the diversity of operational networks (ISPs, enterprises, DDoS mitigation and firewall maintainers, etc.).
- o **Supporting Common Specifications:** Common, open, transport specifications can stimulate engagement by developers, users, researchers, and the broader community. Increased protocol diversity can be beneficial in meeting new requirements, but the ability to innovate without public scrutiny risks point solutions that optimise for specific cases, and that can accidentally disrupt operations of/in different parts of the network. The social contract that maintains the stability of the Internet relies on accepting common transport specifications, and on it being possible to detect violations. The existence of independent measurements, transparency, and public scrutiny of transport protocol behaviour, help the community to enforce the social norm that protocol implementations behave fairly and conform (at least mostly) to the specifications. It is important to find new ways of maintaining that community trust as increased use of transport header encryption limits visibility into transport behaviour (see also Section 5.3).

- o Impact on Benchmarking and Understanding Feature Interactions: An appropriate vantage point for observation, coupled with timing information about traffic flows, provides a valuable tool for benchmarking network devices, endpoint stacks, and/or configurations. This can help understand complex feature interactions. An inability to observe transport header information can make it harder to diagnose and explore interactions between features at different protocol layers, a side-effect of not allowing a choice of vantage point from which this information is observed. New approaches might have to be developed.
- o Impact on Research and Development: Hiding transport header information can impede independent research into new mechanisms, measurement of behaviour, and development initiatives. Experience shows that transport protocols are complicated to design and complex to deploy, and that individual mechanisms have to be evaluated while considering other mechanisms, across a broad range of network topologies and with attention to the impact on traffic sharing the capacity. If increased use of transport header encryption results in reduced availability of open data, it could eliminate the independent checks to the standardisation process that have previously been in place from research and academic contributors (e.g., the role of the IRTF Internet Congestion Control Research Group (ICCRG) and research publications in reviewing new transport mechanisms and assessing the impact of their deployment).

Observable transport header information might be useful to various stakeholders. Other sets of stakeholders have incentives to limit what can be observed. This document does not make recommendations about what information ought to be exposed, to whom it ought to be observable, or how this will be achieved. There are also design choices about where observable fields are placed. For example, one location could be a part of the transport header outside of the encryption envelope, another alternative is to carry the information in a network-layer option or extension header. New transport protocol designs ought to explicitly identify any fields that are intended to be observed, consider if there are alternative ways of providing the information, and reflect on the implications of observable fields being used by on-path network devices, and how this might impact user privacy and protocol evolution when these fields become ossified.

As [RFC7258] notes, "Making networks unmanageable to mitigate (pervasive monitoring) is not an acceptable outcome, but ignoring (pervasive monitoring) would go against the consensus documented here." Providing explicit information can help avoid traffic being

inappropriately classified, impacting application performance. An appropriate balance will emerge over time as real instances of this tension are analysed [RFC7258]. This balance between information exposed and information hidden ought to be carefully considered when specifying new transport protocols.

## 8. Security Considerations

This document is about design and deployment considerations for transport protocols. Issues relating to security are discussed throughout this document.

Authentication, confidentiality protection, and integrity protection are identified as Transport Features by [RFC8095]. As currently deployed in the Internet, these features are generally provided by a protocol or layer on top of the transport protocol [RFC8922].

Confidentiality and strong integrity checks have properties that can also be incorporated into the design of a transport protocol or to modify an existing transport. Integrity checks can protect an endpoint from undetected modification of protocol fields by on-path network devices, whereas encryption and obfuscation or greasing can further prevent these headers being utilised by network devices [RFC8701]. Preventing observation of headers provides an opportunity for greater freedom to update the protocols and can ease experimentation with new techniques and their final deployment in endpoints. A protocol specification needs to weigh the costs of ossifying common headers, versus the potential benefits of exposing specific information that could be observed along the network path to provide tools to manage new variants of protocols.

Header encryption can provide confidentiality of some or all of the transport header information. This prevents an on-path device from gaining knowledge of the header field. It therefore prevents mechanisms being built that directly rely on the information or seeks to infer semantics of an exposed header field. Reduced visibility into transport metadata can limit the ability to measure and characterise traffic, and conversely can provide privacy benefits.

Extending the transport payload security context to also include the transport protocol header protects both types of information with the same key. A privacy concern would arise if this key was shared with a third party, e.g., providing access to transport header information to debug a performance issue, would also result in exposing the transport payload data to the same third party. Such risks would be mitigated using a layered security design that provides one domain of protection and associated keys for the transport payload and

encrypted transport headers; and a separate domain of protection and associated keys for any observable transport header fields.

Exposed transport headers are sometimes utilised as a part of the information to detect anomalies in network traffic. "While PM is an attack, other forms of monitoring that might fit the definition of PM can be beneficial and not part of any attack, e.g., network management functions monitor packets or flows and anti-spam mechanisms need to see mail message content." [RFC7258]. This can be used as the first line of defence to identify potential threats from DoS or malware and redirect suspect traffic to dedicated nodes responsible for DoS analysis, malware detection, or to perform packet "scrubbing" (the normalisation of packets so that there are no ambiguities in interpretation by the ultimate destination of the packet). These techniques are currently used by some operators to also defend from distributed DoS attacks.

Exposed transport header fields can also form a part of the information used by the receiver of a transport protocol to protect the transport layer from data injection by an attacker. In evaluating this use of exposed header information, it is important to consider whether it introduces a significant DoS threat. For example, an attacker could construct a DoS attack by sending packets with a sequence number that falls within the currently accepted range of sequence numbers at the receiving endpoint. This would then introduce additional work at the receiving endpoint, even though the data in the attacking packet might not finally be delivered by the transport layer. This is sometimes known as a "shadowing attack". An attack can, for example, disrupt receiver processing, trigger loss and retransmission, or make a receiving endpoint perform unproductive decryption of packets that cannot be successfully decrypted (forcing a receiver to commit decryption resources, or to update and then restore protocol state).

One mitigation to off-path attack is to deny knowledge of what header information is accepted by a receiver or obfuscate the accepted header information, e.g., setting a non-predictable initial value for a sequence number during a protocol handshake, as in [RFC3550] and [RFC6056], or a port value that cannot be predicted (see Section 5.1 of [RFC8085]). A receiver could also require additional information to be used as a part of a validation check before accepting packets at the transport layer (e.g., utilising a part of the sequence number space that is encrypted; or by verifying an encrypted token not visible to an attacker). This would also mitigate against on-path attacks. An additional processing cost can be incurred when decryption is attempted before a receiver discards an injected packet.

The existence of open transport protocol standards, and a research and operations community with a history of independent observation and evaluation of performance data, encourages fairness and conformance to those standards. This suggests careful consideration will be made over where, and when, measurement samples are collected. An appropriate balance between encrypting some or all of the transport header information needs to be considered. Open data, and accessibility to tools that can help understand trends in application deployment, network traffic and usage patterns can all contribute to understanding security challenges.

The Security and Privacy Considerations in the Framework for Large-Scale Measurement of Broadband Performance (LMAP) [RFC7594] contain considerations for Active and Passive measurement techniques and supporting material on measurement context.

Addition of observable transport information to the path increases the information available to an observer and may, when this information can be linked to a node or user, reduce the privacy of the user. See the security considerations of [RFC8558].

## 9. IANA Considerations

This memo includes no request to IANA.

## 10. Acknowledgements

The authors would like to thank Mohamed Boucadair, Spencer Dawkins, Tom Herbert, Jana Iyengar, Mirja Kuehlewind, Kyle Rose, Kathleen Moriarty, Al Morton, Chris Seal, Joe Touch, Brian Trammell, Chris Wood, Thomas Fossati, Mohamed Boucadair, Martin Thomson, David Black, Martin Duke, Joel Halpern and members of TSVWG for their comments and feedback.

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688421, and the EU Stand ICT Call 4. The opinions expressed and arguments employed reflect only the authors' view. The European Commission is not responsible for any use that might be made of that information.

This work has received funding from the UK Engineering and Physical Sciences Research Council under grant EP/R04144X/1.

## 11. Informative References

## [bufferbloat]

Gettys, J. and K. Nichols, "Bufferbloat: dark buffers in the Internet. Communications of the ACM, 55(1):57-65", January 2012.

## [I-D.ietf-6man-ipv6-alt-mark]

Fioccola, G., Zhou, T., Cociglio, M., and F. Qin, "IPv6 Application of the Alternate Marking Method", draft-ietf-6man-ipv6-alt-mark-00 (work in progress), May 2020.

## [I-D.ietf-ippm-ioam-data]

Brockners, F., Bhandari, S., and T. Mizrahi, "Data Fields for In-situ OAM", draft-ietf-ippm-ioam-data-10 (work in progress), July 2020.

## [I-D.ietf-quic-transport]

Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-29 (work in progress), June 2020.

## [I-D.ietf-tls-dtls13]

Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-38 (work in progress), May 2020.

## [I-D.ietf-tsvwg-rtcweb-qos]

Jones, P., Dhesikan, S., Jennings, C., and D. Druta, "DSCP Packet Markings for WebRTC QoS", draft-ietf-tsvwg-rtcweb-qos-18 (work in progress), August 2016.

## [I-D.marx-qlog-main-schema]

Marx, R., "Main logging schema for qlog", draft-marx-qlog-main-schema-02 (work in progress), November 2020.

## [I-D.trammell-plus-abstract-mech]

Trammell, B., "Abstract Mechanisms for a Cooperative Path Layer under Endpoint Control", draft-trammell-plus-abstract-mech-00 (work in progress), September 2016.

## [Latency]

Briscoe, B., "Reducing Internet Latency: A Survey of Techniques and Their Merits, IEEE Comm. Surveys & Tutorials. 26;18(3) p2149-2196", November 2014.

## [Measurement]

Fairhurst, G., Kuehlewind, M., and D. Lopez, "Measurement-based Protocol Design, Eur. Conf. on Networks and Communications, Oulu, Finland.", June 2017.



- [PAM-RTT] Trammell, B. and M. Kuehlewind, "Revisiting the Privacy Implications of Two-Way Internet Latency Data (in Proc. PAM 2018)", March 2018.
- [Quic-Trace]  
"https:QUIC trace utilities //github.com/google/quic-trace".
- [RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <<https://www.rfc-editor.org/info/rfc791>>.
- [RFC2410] Glenn, R. and S. Kent, "The NULL Encryption Algorithm and Its Use With IPsec", RFC 2410, DOI 10.17487/RFC2410, November 1998, <<https://www.rfc-editor.org/info/rfc2410>>.
- [RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <<https://www.rfc-editor.org/info/rfc2474>>.
- [RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, DOI 10.17487/RFC2475, December 1998, <<https://www.rfc-editor.org/info/rfc2475>>.
- [RFC2507] Degermark, M., Nordgren, B., and S. Pink, "IP Header Compression", RFC 2507, DOI 10.17487/RFC2507, February 1999, <<https://www.rfc-editor.org/info/rfc2507>>.
- [RFC2508] Casner, S. and V. Jacobson, "Compressing IP/UDP/RTP Headers for Low-Speed Serial Links", RFC 2508, DOI 10.17487/RFC2508, February 1999, <<https://www.rfc-editor.org/info/rfc2508>>.
- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, DOI 10.17487/RFC2914, September 2000, <<https://www.rfc-editor.org/info/rfc2914>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC3234] Carpenter, B. and S. Brim, "Middleboxes: Taxonomy and Issues", RFC 3234, DOI 10.17487/RFC3234, February 2002, <<https://www.rfc-editor.org/info/rfc3234>>.

- [RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <<https://www.rfc-editor.org/info/rfc3261>>.
- [RFC3393] Demichelis, C. and P. Chimento, "IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)", RFC 3393, DOI 10.17487/RFC3393, November 2002, <<https://www.rfc-editor.org/info/rfc3393>>.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <<https://www.rfc-editor.org/info/rfc3550>>.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <<https://www.rfc-editor.org/info/rfc3552>>.
- [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <<https://www.rfc-editor.org/info/rfc3711>>.
- [RFC4302] Kent, S., "IP Authentication Header", RFC 4302, DOI 10.17487/RFC4302, December 2005, <<https://www.rfc-editor.org/info/rfc4302>>.
- [RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <<https://www.rfc-editor.org/info/rfc4303>>.
- [RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, DOI 10.17487/RFC4566, July 2006, <<https://www.rfc-editor.org/info/rfc4566>>.
- [RFC4585] Ott, J., Wenger, S., Sato, N., Burmeister, C., and J. Rey, "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, DOI 10.17487/RFC4585, July 2006, <<https://www.rfc-editor.org/info/rfc4585>>.
- [RFC4737] Morton, A., Ciavattone, L., Ramachandran, G., Shalunov, S., and J. Perser, "Packet Reordering Metrics", RFC 4737, DOI 10.17487/RFC4737, November 2006, <<https://www.rfc-editor.org/info/rfc4737>>.

- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.
- [RFC5166] Floyd, S., Ed., "Metrics for the Evaluation of Congestion Control Mechanisms", RFC 5166, DOI 10.17487/RFC5166, March 2008, <<https://www.rfc-editor.org/info/rfc5166>>.
- [RFC5218] Thaler, D. and B. Aboba, "What Makes for a Successful Protocol?", RFC 5218, DOI 10.17487/RFC5218, July 2008, <<https://www.rfc-editor.org/info/rfc5218>>.
- [RFC5236] Jayasumana, A., Piratla, N., Banka, T., Bare, A., and R. Whitner, "Improved Packet Reordering Metrics", RFC 5236, DOI 10.17487/RFC5236, June 2008, <<https://www.rfc-editor.org/info/rfc5236>>.
- [RFC5426] Okmianski, A., "Transmission of Syslog Messages over UDP", RFC 5426, DOI 10.17487/RFC5426, March 2009, <<https://www.rfc-editor.org/info/rfc5426>>.
- [RFC5481] Morton, A. and B. Claise, "Packet Delay Variation Applicability Statement", RFC 5481, DOI 10.17487/RFC5481, March 2009, <<https://www.rfc-editor.org/info/rfc5481>>.
- [RFC5795] Sandlund, K., Pelletier, G., and L-E. Jonsson, "The RObusT Header Compression (ROHC) Framework", RFC 5795, DOI 10.17487/RFC5795, March 2010, <<https://www.rfc-editor.org/info/rfc5795>>.
- [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <<https://www.rfc-editor.org/info/rfc5925>>.
- [RFC6056] Larsen, M. and F. Gont, "Recommendations for Transport-Protocol Port Randomization", BCP 156, RFC 6056, DOI 10.17487/RFC6056, January 2011, <<https://www.rfc-editor.org/info/rfc6056>>.
- [RFC6269] Ford, M., Ed., Boucadair, M., Durand, A., Levis, P., and P. Roberts, "Issues with IP Address Sharing", RFC 6269, DOI 10.17487/RFC6269, June 2011, <<https://www.rfc-editor.org/info/rfc6269>>.
- [RFC6294] Hu, Q. and B. Carpenter, "Survey of Proposed Use Cases for the IPv6 Flow Label", RFC 6294, DOI 10.17487/RFC6294, June 2011, <<https://www.rfc-editor.org/info/rfc6294>>.

- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6437] Amante, S., Carpenter, B., Jiang, S., and J. Rajahalme, "IPv6 Flow Label Specification", RFC 6437, DOI 10.17487/RFC6437, November 2011, <<https://www.rfc-editor.org/info/rfc6437>>.
- [RFC6438] Carpenter, B. and S. Amante, "Using the IPv6 Flow Label for Equal Cost Multipath Routing and Link Aggregation in Tunnels", RFC 6438, DOI 10.17487/RFC6438, November 2011, <<https://www.rfc-editor.org/info/rfc6438>>.
- [RFC6846] Pelletier, G., Sandlund, K., Jonsson, L-E., and M. West, "RObust Header Compression (ROHC): A Profile for TCP/IP (ROHC-TCP)", RFC 6846, DOI 10.17487/RFC6846, January 2013, <<https://www.rfc-editor.org/info/rfc6846>>.
- [RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <<https://www.rfc-editor.org/info/rfc6973>>.
- [RFC7098] Carpenter, B., Jiang, S., and W. Tarreau, "Using the IPv6 Flow Label for Load Balancing in Server Farms", RFC 7098, DOI 10.17487/RFC7098, January 2014, <<https://www.rfc-editor.org/info/rfc7098>>.
- [RFC7126] Gont, F., Atkinson, R., and C. Pignataro, "Recommendations on Filtering of IPv4 Packets Containing IPv4 Options", BCP 186, RFC 7126, DOI 10.17487/RFC7126, February 2014, <<https://www.rfc-editor.org/info/rfc7126>>.
- [RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <<https://www.rfc-editor.org/info/rfc7258>>.
- [RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <<https://www.rfc-editor.org/info/rfc7413>>.
- [RFC7414] Duke, M., Braden, R., Eddy, W., Blanton, E., and A. Zimmermann, "A Roadmap for Transmission Control Protocol (TCP) Specification Documents", RFC 7414, DOI 10.17487/RFC7414, February 2015, <<https://www.rfc-editor.org/info/rfc7414>>.

- [RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <<https://www.rfc-editor.org/info/rfc7567>>.
- [RFC7594] Eardley, P., Morton, A., Bagnulo, M., Burbridge, T., Aitken, P., and A. Akhter, "A Framework for Large-Scale Measurement of Broadband Performance (LMAP)", RFC 7594, DOI 10.17487/RFC7594, September 2015, <<https://www.rfc-editor.org/info/rfc7594>>.
- [RFC7605] Touch, J., "Recommendations on Using Assigned Transport Port Numbers", BCP 165, RFC 7605, DOI 10.17487/RFC7605, August 2015, <<https://www.rfc-editor.org/info/rfc7605>>.
- [RFC7624] Barnes, R., Schneier, B., Jennings, C., Hardie, T., Trammell, B., Huitema, C., and D. Borkmann, "Confidentiality in the Face of Pervasive Surveillance: A Threat Model and Problem Statement", RFC 7624, DOI 10.17487/RFC7624, August 2015, <<https://www.rfc-editor.org/info/rfc7624>>.
- [RFC7799] Morton, A., "Active and Passive Metrics and Methods (with Hybrid Types In-Between)", RFC 7799, DOI 10.17487/RFC7799, May 2016, <<https://www.rfc-editor.org/info/rfc7799>>.
- [RFC7872] Gont, F., Linkova, J., Chown, T., and W. Liu, "Observations on the Dropping of Packets with IPv6 Extension Headers in the Real World", RFC 7872, DOI 10.17487/RFC7872, June 2016, <<https://www.rfc-editor.org/info/rfc7872>>.
- [RFC7928] Kuhn, N., Ed., Natarajan, P., Ed., Khademi, N., Ed., and D. Ros, "Characterization Guidelines for Active Queue Management (AQM)", RFC 7928, DOI 10.17487/RFC7928, July 2016, <<https://www.rfc-editor.org/info/rfc7928>>.
- [RFC7983] Petit-Huguenin, M. and G. Salgueiro, "Multiplexing Scheme Updates for Secure Real-time Transport Protocol (SRTP) Extension for Datagram Transport Layer Security (DTLS)", RFC 7983, DOI 10.17487/RFC7983, September 2016, <<https://www.rfc-editor.org/info/rfc7983>>.
- [RFC8033] Pan, R., Natarajan, P., Baker, F., and G. White, "Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem", RFC 8033, DOI 10.17487/RFC8033, February 2017, <<https://www.rfc-editor.org/info/rfc8033>>.

- [RFC8084] Fairhurst, G., "Network Transport Circuit Breakers", BCP 208, RFC 8084, DOI 10.17487/RFC8084, March 2017, <<https://www.rfc-editor.org/info/rfc8084>>.
- [RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <<https://www.rfc-editor.org/info/rfc8085>>.
- [RFC8086] Yong, L., Ed., Crabbe, E., Xu, X., and T. Herbert, "GRE-in-UDP Encapsulation", RFC 8086, DOI 10.17487/RFC8086, March 2017, <<https://www.rfc-editor.org/info/rfc8086>>.
- [RFC8087] Fairhurst, G. and M. Welzl, "The Benefits of Using Explicit Congestion Notification (ECN)", RFC 8087, DOI 10.17487/RFC8087, March 2017, <<https://www.rfc-editor.org/info/rfc8087>>.
- [RFC8095] Fairhurst, G., Ed., Trammell, B., Ed., and M. Kuehlewind, Ed., "Services Provided by IETF Transport Protocols and Congestion Control Mechanisms", RFC 8095, DOI 10.17487/RFC8095, March 2017, <<https://www.rfc-editor.org/info/rfc8095>>.
- [RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <<https://www.rfc-editor.org/info/rfc8200>>.
- [RFC8250] Elkins, N., Hamilton, R., and M. Ackermann, "IPv6 Performance and Diagnostic Metrics (PDM) Destination Option", RFC 8250, DOI 10.17487/RFC8250, September 2017, <<https://www.rfc-editor.org/info/rfc8250>>.
- [RFC8289] Nichols, K., Jacobson, V., McGregor, A., Ed., and J. Iyengar, Ed., "Controlled Delay Active Queue Management", RFC 8289, DOI 10.17487/RFC8289, January 2018, <<https://www.rfc-editor.org/info/rfc8289>>.
- [RFC8290] Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm", RFC 8290, DOI 10.17487/RFC8290, January 2018, <<https://www.rfc-editor.org/info/rfc8290>>.
- [RFC8404] Moriarty, K., Ed. and A. Morton, Ed., "Effects of Pervasive Encryption on Operators", RFC 8404, DOI 10.17487/RFC8404, July 2018, <<https://www.rfc-editor.org/info/rfc8404>>.

- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [RFC8462] Rooney, N. and S. Dawkins, Ed., "Report from the IAB Workshop on Managing Radio Networks in an Encrypted World (MaRNEW)", RFC 8462, DOI 10.17487/RFC8462, October 2018, <<https://www.rfc-editor.org/info/rfc8462>>.
- [RFC8517] Dolson, D., Ed., Snellman, J., Boucadair, M., Ed., and C. Jacquenet, "An Inventory of Transport-Centric Functions Provided by Middleboxes: An Operator Perspective", RFC 8517, DOI 10.17487/RFC8517, February 2019, <<https://www.rfc-editor.org/info/rfc8517>>.
- [RFC8546] Trammell, B. and M. Kuehlewind, "The Wire Image of a Network Protocol", RFC 8546, DOI 10.17487/RFC8546, April 2019, <<https://www.rfc-editor.org/info/rfc8546>>.
- [RFC8548] Bittau, A., Giffin, D., Handley, M., Mazieres, D., Slack, Q., and E. Smith, "Cryptographic Protection of TCP Streams (tcpcrypt)", RFC 8548, DOI 10.17487/RFC8548, May 2019, <<https://www.rfc-editor.org/info/rfc8548>>.
- [RFC8558] Hardie, T., Ed., "Transport Protocol Path Signals", RFC 8558, DOI 10.17487/RFC8558, April 2019, <<https://www.rfc-editor.org/info/rfc8558>>.
- [RFC8684] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 8684, DOI 10.17487/RFC8684, March 2020, <<https://www.rfc-editor.org/info/rfc8684>>.
- [RFC8701] Benjamin, D., "Applying Generate Random Extensions And Sustain Extensibility (GREASE) to TLS Extensibility", RFC 8701, DOI 10.17487/RFC8701, January 2020, <<https://www.rfc-editor.org/info/rfc8701>>.
- [RFC8724] Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and JC. Zuniga, "SCHC: Generic Framework for Static Context Header Compression and Fragmentation", RFC 8724, DOI 10.17487/RFC8724, April 2020, <<https://www.rfc-editor.org/info/rfc8724>>.
- [RFC8837] Jones, P., Dhesikan, S., Jennings, C., and D. Druta, "Differentiated Services Code Point (DSCP) Packet Markings for WebRTC QoS", RFC 8837, DOI 10.17487/RFC8837, January 2021, <<https://www.rfc-editor.org/info/rfc8837>>.

[RFC8922] Enghardt, T., Pauly, T., Perkins, C., Rose, K., and C. Wood, "A Survey of the Interaction between Security Protocols and Transport Services", RFC 8922, DOI 10.17487/RFC8922, October 2020, <<https://www.rfc-editor.org/info/rfc8922>>.



## Appendix A. Revision information

- 00 This is an individual draft for the IETF community.
  - 01 This draft was a result of walking away from the text for a few days and then reorganising the content.
  - 02 This draft fixes textual errors.
  - 03 This draft follows feedback from people reading this draft.
  - 04 This adds an additional contributor and includes significant reworking to ready this for review by the wider IETF community Colin Perkins joined the author list.
- Comments from the community are welcome on the text and recommendations.
- 05 Corrections received and helpful inputs from Mohamed Boucadair.
  - 06 Updated following comments from Stephen Farrell, and feedback via email. Added a draft conclusion section to sketch some strawman scenarios that could emerge.
  - 07 Updated following comments from Al Morton, Chris Seal, and other feedback via email.
  - 08 Updated to address comments sent to the TSVWG mailing list by Kathleen Moriarty (on 08/05/2018 and 17/05/2018), Joe Touch on 11/05/2018, and Spencer Dawkins.
  - 09 Updated security considerations.
  - 10 Updated references, split the Introduction, and added a paragraph giving some examples of why ossification has been an issue.
  - 01 This resolved some reference issues. Updated section on observation by devices on the path.
  - 02 Comments received from Kyle Rose, Spencer Dawkins and Tom Herbert. The network-layer information has also been re-organised after comments at IETF-103.
  - 03 Added a section on header compression and rewriting of sections referring to RTP transport. This version contains author editorial work and removed duplicate section.
  - 04 Revised following SecDir Review

- o Added some text on TLS story (additional input sought on relevant considerations).
- o Section 2, paragraph 8 - changed to be clearer, in particular, added "Encryption with secure key distribution prevents"
- o Flow label description rewritten based on PS/BCP RFCs.
- o Clarify requirements from RFCs concerning the IPv6 flow label and highlight ways it can be used with encryption. (section 3.1.3)
- o Add text on the explicit spin-bit work in the QUIC DT. Added greasing of spin-bit. (Section 6.1)
- o Updated section 6 and added more explanation of impact on operators.
- o Other comments addressed.

-05 Editorial pass and minor corrections noted on TSVWG list.

-06 Updated conclusions and minor corrections. Responded to request to add OAM discussion to Section 6.1.

-07 Addressed feedback from Ruediger and Thomas.

Section 2 deserved some work to make it easier to read and avoid repetition. This edit finally gets to this, and eliminates some duplication. This also moves some of the material from section 2 to reform a clearer conclusion. The scope remains focussed on the usage of transport headers and the implications of encryption - not on proposals for new techniques/specifications to be developed.

-08 Addressed feedback and completed editorial work, including updating the text referring to RFC7872, in preparation for a WGLC.

-09 Updated following WGLC. In particular, thanks to Joe Touch (specific comments and commentary on style and tone); Dimitri Tikonov (editorial); Christian Huitema (various); David Black (various). Amended privacy considerations based on SECDIR review. Emile Stephan (inputs on operations measurement); Various others.

Added summary text and refs to key sections. Note to editors: The section numbers are hard-linked.

-10 Updated following additional feedback from 1st WGLC. Comments from David Black; Tommy Pauly; Ian Swett; Mirja Kuehlewind; Peter

Gutmann; Ekr; and many others via the TSVWG list. Some people thought that "needed" and "need" could

represent requirements in the document, etc. this has been clarified.

-11 Updated following additional feedback from Martin Thomson, and corrections from other reviewers.

-12 Updated following additional feedback from reviewers.

-13 Updated following 2nd WGLC with comments from D.L.Black; T. Herbert; Ekr; and other reviewers.

-14 Update to resolve feedback to rev -13. This moves the general discussion of adding fields to transport packets to section 6, and discusses with reference to material in RFC8558.

-15 Feedback from D.L. Black, T. Herbert, J. Touch, S. Dawkins and M. Duke. Update to add reference to RFC7605. Clarify a focus on immutable transport fields, rather than modifying middleboxes with Tom H. Clarified Header Compression discussion only provides a list of examples of HC methods for transport. Clarified port usage with Tom H/Joe T. Removed some duplicated sentences, and minor edits. Added NULL-ESP. Improved after initial feedback from Martin Duke.

-16 Editorial comments from Mohamed Boucadair. Added DTLS 1.3.

-17 Revised to satisfy ID-NITs and updates REFs to latest rev, updated HC Refs; cited IAB guidance on security and privacy within IETF specs.

-18 Revised based on AD review.

-19 Revised after additional AD review request, and request to restructure.

-20 Revised after directorate reviews and IETF LC comments.

Gen-ART:

- o While section 2 does include a discussion of traffic mis-ordering, it does not include a discussion of ECMP, and the dependence of ECMP on flow identification to avoid significant packet mis-ordering.:: ECMP added as example.
- o Section 5.1 of this document discusses the use of Hop-by-Hop IPv6 options. It seems that it should acknowledge and discuss the applicability of the sentence "New hop-by-hop options are not

recommended..." from section 4.8 of RFC 8200. I think a good argument can be made in this case as to why (based on the rest of the sentence from 8200) the recommendation does not apply to this proposal. The document should make the argument.:: Quoted RFC sentences directly to avoid interpreting them.

- o I found the discussion of header compression slightly confusing. Given that the TCP / UDP header is small even compared to the IP header, it is difficult to see why encrypting it would have a significant impact on header compression efficacy. :: Added a preface that explains that HC methods are most effective for bit-congestive links.
- o The wording in section 6.2 on adding header information to an IP packet has the drawback of seeming to imply that one could add (or remove) such information in the network, without adding an encapsulating header. That is not permitted by RFC 8200 (IPv6). It would be good to clarify the first paragraph. (The example, which talks about the sender putting in the information is, of course, fine.) :: Unintended - added a sentence of preface.

SECDIR:: Previous revisions were updated following Early Review comments.

OPSEC:: No additional changes were requested in the OPSEC review.

IETF LC:: Tom Herbert: Please refer to 8200 on EH :: addressed in response to Joel above. Michael Richardson, Fernando Gont, Tom Herbert: Continuation of discussion on domains where EH might be (or not) useful and the tussle on what information to reveal. Unclear yet what additional text should be changed within this ID.

-----

- 21 Revised after IESG review:

Revision 21 includes revised text after comments from Zahed, Erik Kline, Rob Wilton, Eric Vyncke, Roman Danyliw, and Benjamin Kaduk.

Authors' Addresses

Godred Fairhurst  
University of Aberdeen  
Department of Engineering  
Fraser Noble Building  
Aberdeen AB24 3UE  
Scotland

EMail: [gorry@erg.abdn.ac.uk](mailto:gorry@erg.abdn.ac.uk)  
URI: <http://www.erg.abdn.ac.uk/>

Colin Perkins  
University of Glasgow  
School of Computing Science  
Glasgow G12 8QQ  
Scotland

EMail: [csp@csperkins.org](mailto:csp@csperkins.org)  
URI: <https://csperkins.org/>

TSVWG  
Internet Draft  
Intended status: Standards Track  
Intended updates: 768  
Expires: September 2022

J. Touch  
Independent Consultant  
March 26, 2022

Transport Options for UDP  
draft-ietf-tsvwg-udp-options-18.txt

Abstract

Transport protocols are extended through the use of transport header options. This document extends UDP by indicating the location, syntax, and semantics for UDP transport layer options.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at <https://www.ietf.org/shadow.html>

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 26, 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction.....	3
2. Conventions used in this document.....	3
3. Terminology.....	3
4. Background.....	4
5. The UDP Option Area.....	5
6. The UDP Surplus Area Structure.....	8
7. The Option Checksum (OCS).....	8
8. UDP Options.....	10
9. Safe UDP Options.....	13
9.1. End of Options List (EOL).....	13
9.2. No Operation (NOP).....	14
9.3. Alternate Payload Checksum (APC).....	14
9.4. Fragmentation (FRAG).....	16
9.5. Maximum Datagram Size (MDS).....	19
9.6. Maximum Reassembled Datagram Size (MRDS).....	20
9.7. Echo request (REQ) and echo response (RES).....	21
9.8. Timestamps (TIME).....	21
9.9. Authentication (AUTH).....	22
9.10. Experimental (EXP).....	23
10. UNSAFE Options.....	24
10.1. UNSAFE Encryption (UENC).....	25
10.2. UNSAFE Experimental (UEXP).....	25
11. Rules for designing new options.....	25
12. Option inclusion and processing.....	26
13. UDP API Extensions.....	28
14. UDP Options are for Transport, Not Transit.....	29
15. UDP options vs. UDP-Lite.....	29
16. Interactions with Legacy Devices.....	30
17. Options in a Stateless, Unreliable Transport Protocol.....	30
18. UDP Option State Caching.....	31
19. Updates to RFC 768.....	31
20. Interactions with other RFCs (and drafts).....	32
21. Multicast Considerations.....	33
22. Security Considerations.....	33
23. IANA Considerations.....	34
24. References.....	35
24.1. Normative References.....	35

24.2. Informative References.....	35
25. Acknowledgments.....	38
Appendix A. Implementation Information.....	39

## 1. Introduction

Transport protocols use options as a way to extend their capabilities. TCP [RFC793], SCTP [RFC4960], and DCCP [RFC4340] include space for these options but UDP [RFC768] currently does not. This document defines an extension to UDP that provides space for transport options including their generic syntax and semantics for their use in UDP's stateless, unreliable message protocol.

## 2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

In this document, the characters ">>" preceding an indented line(s) indicates a statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the portions of this RFC covered by these key words.

## 3. Terminology

The following terminology is used in this document:

- o IP datagram [RFC791][RFC8200] - an IP packet, composed of the IP header and an IP payload area
- o User datagram - a UDP packet, composed of a UDP header and UDP payload; as discussed herein, that payload need not extend to the end of the IP datagram
- o UDP packet - the more contemporary term used herein to refer to a user datagram [RFC768]
- o Surplus area - the area of an IP payload that follows a UDP packet; this area is used for UDP options in this document



- o UDP fragment - one or more components of a UDP packet and its UDP options that enables transmission as IP payloads larger than permitted by IP datagram maximum sizes; note that each UDP fragment is itself transmitted as a UDP packet with its own options
- o (UDP) User data - the user data field of a UDP packet [RFC768]
- o UDP Length - the length field of a UDP header [RFC768]
- o Must-support options - UDP options that all implementations are required to support. Their use in individual UDP packets is optional.

#### 4. Background

Many protocols include a default, invariant header and an area for header options that varies from packet to packet. These options enable the protocol to be extended for use in particular environments or in ways unforeseen by the original designers. Examples include TCP's Maximum Segment Size, Window Scale, Timestamp, and Authentication Options [RFC793][RFC5925][RFC7323].

Header options are used both in stateful (connection-oriented, e.g., TCP [RFC793], SCTP [RFC4960], DCCP [RFC4340]) and stateless (connectionless, e.g., IPv4 [RFC791], IPv6 [RFC8200]) protocols. In stateful protocols they can help extend the way in which state is managed. In stateless protocols their effect is often limited to individual packets, but they can have an aggregate effect on a sequence of packets as well.

UDP is one of the most popular protocols that lacks space for header options [RFC768]. The UDP header was intended to be a minimal addition to IP, providing only ports and a checksum for error detection. This document extends UDP to provide a trailer area for such options, located after the UDP user data.

UDP options are possible because UDP includes its own length field, separate from that of the IP header. Other transport protocols infer transport payload length from the IP datagram length (TCP, DCCP, SCTP). There are a number of reasons why Internet historians suggest that UDP includes this field, e.g., to support multiple UDP packets within the same IP datagram or to indicate the length of the UDP user data as distinct from zero padding required for systems that require writes that are not byte-aligned. These suggestions are not consistent with earlier versions of UDP or with concurrent design of multi-segment multiplexing protocols, however, so the real reason

remains unknown. Regardless, this field presents an opportunity to differentiate the UDP user data from the implied transport payload length, which this document leverages to support a trailer options field.

There are other ways to include additional header fields or options in protocols that otherwise are not extensible. In particular, in-band encoding can be used to differentiate transport payload from additional fields, such as was proposed in [Hi15]. This approach can cause complications for interactions with legacy devices, and is thus not considered further in this document.

IPv6 Teredo [RFC6081] uses values of the UDP Length that are larger than the IP payload as an additional type of signal, as noted in Section 20. UDP options uses a value smaller than the IP payload to enable backwards compatibility with existing UDP implementations, i.e., to deliver the UDP Length of UDP user data to the application and silently ignore the additional surplus area data. Using a value larger than the IP payload could either be considered malformed (and ought to be silently dropped by UDP processing) or could cause buffer overruns, and so is not considered silently and safely backward compatible.

## 5. The UDP Option Area

The UDP transport header includes demultiplexing and service identification (port numbers), an error detection checksum, and a field that indicates the UDP datagram length (including UDP header). The UDP Length field is typically redundant with the size of the maximum space available as a transport protocol payload, as determined by the IP header (see detail in Section 16). The UDP Option area is created when the UDP Length indicates a smaller transport payload than implied by the IP header.

For IPv4, IP Total Length field indicates the total IP datagram length (including IP header) and the size of the IP options is indicated in the IP header (in 4-byte words) as the "Internet Header Length" (IHL), as shown in Figure 1 [RFC791]. As a result, the typical (and largest valid) value for UDP Length is:

$$\text{UDP\_Length} = \text{IPv4\_Total\_Length} - \text{IPv4\_IHL} * 4$$

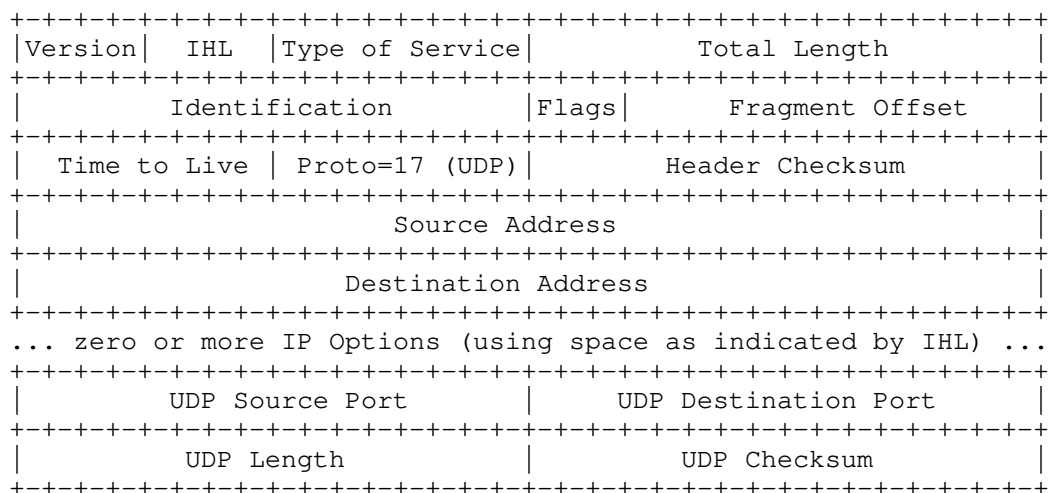


Figure 1 IPv4 datagram with UDP header

For IPv6, the IP Payload Length field indicates the transport payload after the base IPv6 header, which includes the IPv6 extension headers and space available for the transport protocol, as shown in Figure 2 [RFC8200]. Note that the Next HDR field in IPv6 might not indicate UDP (i.e., 17), e.g., when intervening IP extension headers are present. For IPv6, the lengths of any additional IP extensions are indicated within each extension [RFC8200], so the typical (and largest valid) value for UDP Length is:

$$\text{UDP\_Length} = \text{IPv6\_Payload\_Length} - \text{sum}(\text{extension header lengths})$$

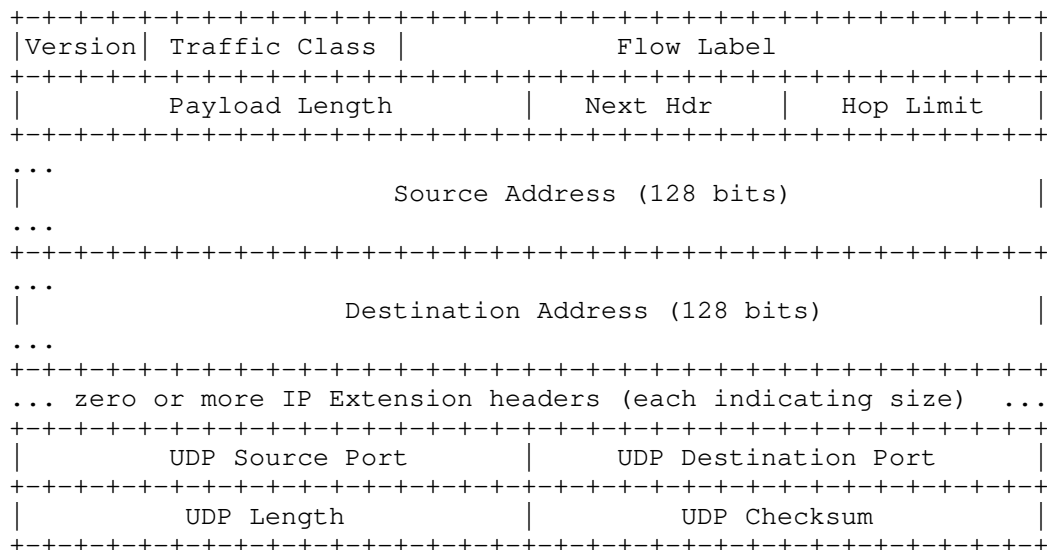


Figure 2 IPv6 datagram with UDP header

In both cases, the space available for the UDP packet is indicated by IP, either directly in the base header (for IPv4) or by adding information in the extensions (for IPv6). In either case, this document will refer to this available space as the "IP transport payload".

As a result of this redundancy, there is an opportunity to use the UDP Length field as a way to break up the IP transport payload into two areas - that intended as UDP user data and an additional "surplus area" (as shown in Figure 3).

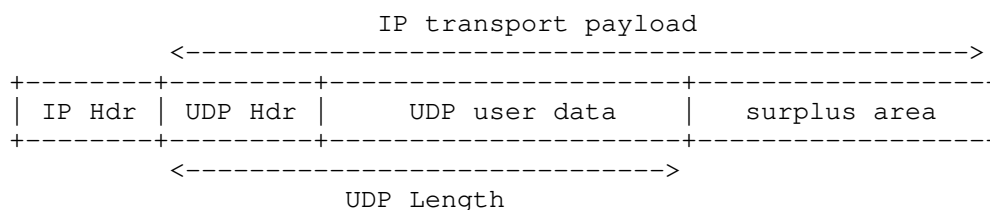


Figure 3 IP transport payload vs. UDP Length

In most cases, the IP transport payload and UDP Length point to the same location, indicating that there is no surplus area. This is not

a requirement of UDP [RFC768] (discussed further in Section 16). This document uses the surplus area for UDP options.

The surplus area can commence at any valid byte offset, i.e., it need not be 16-bit or 32-bit aligned. In effect, this document redefines the UDP "Length" field as a "trailer options offset".

## 6. The UDP Surplus Area Structure

UDP options use the entire surplus area, i.e., the contents of the IP payload after the last byte of the UDP payload. They commence with a 2-byte Option Checksum (OCS) field aligned to the first 2-byte boundary (relative to the start of the IP datagram) of that area, using zeroes for alignment. The UDP option area can be used with any UDP payload length (including zero), as long as there remains enough space for the aligned OCS and the options used.

>> UDP options MAY begin at any UDP length offset.

>> Option area bytes used for alignment before the OCS MUST be zero.

The OCS contains an optional ones-complement sum that detects errors in the surplus area, which is not otherwise covered by the UDP checksum, as detailed in Section 7.

The remainder of the surplus area consists of options defined using a TLV (type, length, and optional value) syntax similar to that of TCP [RFC793], as detailed in Section 8. These options continue until the end of the surplus area or can end earlier using the EOL (end of list) option, followed by zeroes.

## 7. The Option Checksum (OCS)

The Option Checksum (OCS) option is conventional Internet checksum [RFC791] that detects errors in the surplus area. The OCS option contains a 16-bit checksum that is aligned to the first 2-byte boundary, preceded by zeroes for padding (if needed), as shown in Figure 4.

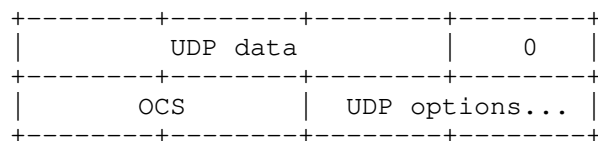


Figure 4 UDP OCS format, here using one zero for alignment

The OCS consists of a 16-bit Internet checksum [RFC1071], computed over the surplus area and including the length of the surplus area as an unsigned 16-bit value. The OCS protects the surplus area from errors in a similar way that the UDP checksum protects the UDP user data (when not zero).

The primary purpose of the OCS is to detect non-standard (i.e., non-option) uses of that area and accidental errors. It is not intended to detect attacks, as discussed further in Section 22.

The design enables traversal of errant middleboxes that incorrectly compute the UDP checksum over the entire IP payload [Fal8], rather than only the UDP header and UDP payload (as indicated by the UDP header length). Because the OCS is computed over the surplus area and its length and then inverted, OCS effectively negates the effect that incorrectly including the surplus has on the UDP checksum. As a result, when OCS is non-zero, the UDP checksum is the same in either case.

>> OCS MUST be non-zero when the UDP checksum is non-zero.

>> When the UDP checksum is zero, the OCS MAY be unused, and is then indicated by a zero OCS value.

Like the UDP checksum, the OCS is optional under certain circumstances and contains zero when not used. UDP checksums can be zero for IPv4 [RFC791] and for IPv6 [RFC8200] when UDP payload already covered by another checksum, as might occur for tunnels [RFC6935]. The same exceptions apply to the OCS when used to detect bit errors; an additional exception occurs for its use in the UDP datagram prior to fragmentation or after reassembly (see Section 9.4).

The OCS covers the surplus area as formatted for transmission and is processed immediately upon reception.

>> If the OCS fails, all options MUST be ignored and the surplus area silently discarded.

>> UDP user data that is validated by a correct UDP checksum MUST be delivered to the application layer, even if the OCS fails, unless the endpoints have negotiated otherwise for this UDP packet's socket pair.

When not used (i.e., containing zero), the OCS is assumed to be "correct" for the purpose of accepting UDP datagrams at a receiver (see Section 12).

## 8. UDP Options

UDP options are typically a minimum of two bytes in length as shown in Figure 5, excepting only the one byte options "No Operation" (NOP) and "End of Options List" (EOL) described below.

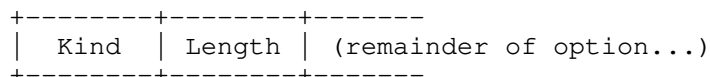


Figure 5 UDP option default format

The Kind field is always one byte. The Length field is one byte for all lengths below 255 (including the Kind and Length bytes). A Length of 255 indicates use of the UDP option extended format shown in Figure 6. The Extended Length field is a 16-bit field in network standard byte order.

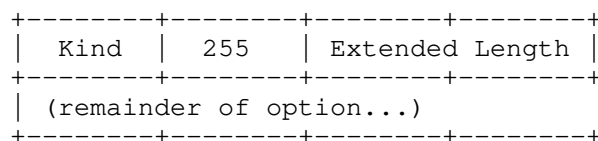


Figure 6 UDP option extended format

>> The UDP length MUST be at least as large as the UDP header (8) and no larger than the IP transport payload. Datagrams with length values outside this range MUST be silently dropped as invalid and logged where rate-limiting permits.

>> Option Lengths (or Extended Lengths, where applicable) smaller than the minimum for the corresponding Kind MUST be treated as an error. Such errors call into question the remainder of the surplus area and thus MUST result in all UDP options being silently discarded.

>> Any UDP option other than EOL and NOP MAY use either the default or extended option formats.

>> Any UDP option whose length is larger than 254 MUST use the UDP option extended format shown in Figure 6.

>> For compactness, UDP options SHOULD use the smallest option format possible.

>> UDP options MUST be interpreted in the order in which they occur in the surplus area.

The following UDP options are currently defined:

Kind	Length	Meaning
-----		
0*	-	End of Options List (EOL)
1*	-	No operation (NOP)
2*	6	Alternate payload checksum (APC)
3*	10/12	Fragmentation (FRAG)
4*	4	Maximum datagram size (MDS)
5*	4	Maximum reassembled datagram size (MRDS)
6*	6	Request (REQ)
7*	6	Response (RES)
8	10	Timestamps (TIME)
9	(varies)	Authentication (AUTH)
10-126	(varies)	UNASSIGNED (assignable by IANA)
127	(varies)	RFC 3692-style experiments (EXP)
128-191		RESERVED
192	(varies)	Encryption (UENC)
193-253		UNASSIGNED-UNSAFE (assignable by IANA)
254	(varies)	RFC 3692-style experiments (UEXP)
255		RESERVED

Options indicated by Kind values in the range 0..127 are known as SAFE options because they do not alter the UDP data payload and thus do not interfere with use of that data by legacy endpoints. Options indicated by Kind values in the range 192..254 are known as UNSAFE options because they do alter the UDP data payload and thus would interfere with legacy endpoints. UNSAFE option nicknames are expected to begin with "U", which should be avoided for safe option nicknames (see Section 23). Kind values 128-191 and 255 are RESERVED and not otherwise defined at this time.

>> RESERVED Kind values MUST NOT be assumed to be either SAFE nor UNSAFE until defined.

Although the FRAG option modifies the original UDP payload contents (i.e., is UNSAFE with respect to the original UDP payload), it is used only in subsequent fragments with zero UDP payloads, thus is SAFE in actual use, as discussed further in Section 9.4.

These options are defined in the following subsections. Options 0 and 1 use the same values as for TCP.



>> An endpoint supporting UDP options MUST support those marked with a "\*" above: EOL, NOP, APC, FRAG, MDS, MRDS, REQ, and RES. This includes both recognizing and being able to generate these options if configured to do so. These are called "must-support" options.

>> An endpoint supporting UDP options MUST treat unsupported options in the UNSAFE range as terminating all option processing.

>> All other SAFE options (without a "\*") MAY be implemented, and their use SHOULD be determined either out-of-band or negotiated, notably if needed to detect when options are silently ignored by legacy receivers.

>> Receivers supporting UDP options MUST silently ignore unknown SAFE options (i.e., in the same way a legacy receiver would). That includes options whose length does not indicate the specified value(s), as long as the length is not inherently invalid (i.e., smaller than 2 for the default and 4 for the extended formats).

>> UNSAFE options are used only in with the FRAG option, in a manner that prevents them from being silently ignored but passing the UDP payload to the user when not supported. This ensures their safe use in environments that might include legacy receivers (See Section 10).

>> Receivers supporting UDP options MUST silently drop all UDP options in a datagram containing an UNSAFE option when any UNSAFE option it contains is unknown. See Section 10 for further discussion of UNSAFE options.

>> Except for NOP, EXP, and UEXP, each option SHOULD NOT occur more than once in a single UDP datagram. If an option other than these occurs more than once, a receiver MUST interpret only the first instance of that option and MUST ignore all others.

>> EXP and UEXP MAY occur more than once, but SHOULD NOT occur more than once using the same ExID (see Sections 9.10 and 10.2).

>> Only the OCS and the AUTH and UENC options depend on the contents of the surplus area. AUTH and UENC are never used together, as UENC would serve both purposes. AUTH and UENC are always computed as if their hash and the OCS are zero; the OCS is always computed as if its contents are zero and after the AUTH or UENC hash has been computed. Future options MUST NOT be defined as having a value dependent on the contents of the surplus area. Otherwise, interactions between those values, the OCS, and the AUTH and UENC options could be unpredictable.

Receivers cannot generally treat unexpected option lengths as invalid, as this would unnecessarily limit future revision of options (e.g., defining a new APC that is defined by having a different length). The exception is only for lengths that imply a physical impossibility, e.g., smaller than two for conventional options and four for extended length options. Impossible lengths should indicate a malformed surplus area and all options silently discarded. Lengths other than those expected should result in safe options being ignored and skipped over, as with any other unknown safe option.

>> Option lengths MUST NOT exceed the IP length of the overall IP datagram. If this occurs, the options MUST be treated as malformed and all options dropped, and the event MAY be logged for diagnostics (logging SHOULD be rate limited).

>> "Must-support" options other than NOP and EOL MUST come before other options.

The requirement that must-support options come before others is intended to allow for endpoints to implement DOS protection, as discussed further in Section 22.

## 9. Safe UDP Options

Safe UDP options can be silently ignored by legacy receivers without affecting the meaning of the UDP user data. They stand in contrast to Unsafe options, which modify UDP user data in ways that render it unusable by legacy receivers (Section 10). The following subsections describe safe options defined in this document.

### 9.1. End of Options List (EOL)

The End of Options List (EOL, Kind=0) option indicates that there are no more options. It is used to indicate the end of the list of options without needing to use NOP options (see the following section) as padding to fill all available option space.

```
+-----+
| Kind=0 |
+-----+
```

Figure 7 UDP EOL option format

>> When the UDP options do not consume the entire surplus area, the last non-NOP option MUST be EOL.

>> NOPs SHOULD NOT be used as padding before the EOL option. As a one byte option, it need not be otherwise aligned.

>> All bytes in the surplus area after EOL MUST be set to zero on transmit.

>> Bytes after EOL in the surplus area MAY be checked as being zero on receipt but MUST be treated as zero regardless of their content and are not passed to the user (e.g., as part of the surplus area).

Requiring the post-option surplus area to be zero prevents side-channel uses of this area, requiring instead that all use of the surplus area be UDP options supported by both endpoints. It is useful to allow this area to be used for zero padding to increase the UDP datagram length without affecting the UDP user data length, e.g., for UDP DPLPMTUD (Section 4.1 of [Fa22]).

## 9.2. No Operation (NOP)

The No Operation (NOP, Kind=1) option is a one-byte placeholder, intended to be used as padding, e.g., to align multi-byte options along 16-bit, 32-bit, or 64-bit boundaries.

```
+-----+
| Kind=1 |
+-----+
```

Figure 8 UDP NOP option format

>> UDP packets SHOULD NOT use more than seven consecutive NOPs, i.e., to support alignment up to 8-byte boundaries. UDP packets SHOULD NOT use NOPs at the end of the options area as a substitute for EOL followed by zero-fill. NOPs are intended to assist with alignment, not as other padding or fill.

This issue is discussed further in Section 22.

## 9.3. Alternate Payload Checksum (APC)

The Alternate Payload Checksum (APC, Kind=2) option provides a stronger alternative to the checksum in the UDP header, using a 32-bit CRC of the conventional UDP user data payload only (excluding the IP pseudoheader, UDP header, and surplus area). It is an "alternate" to the UDP checksum that covers the user data - not to the OCS (the latter covers the surplus area only). Unlike the UDP checksum, APC does not include the IP pseudoheader or UDP header, thus it does not need to be updated by NATs when IP addresses or UDP

ports are rewritten. Its purpose is to detect user data errors that the UDP checksum, when used, might not detect.

A CRC32c has been chosen because of its ubiquity and use in other Internet protocols, including iSCSI and SCTP. The option contains the CRC32c in network standard byte order, as described in [RFC3385].

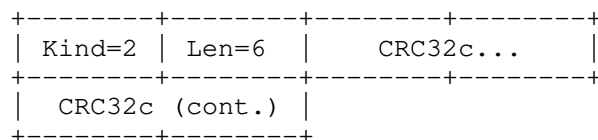


Figure 9 UDP APC option format

When present, the APC always contains a valid CRC checksum. There are no reserved values, including the value of zero. If the CRC is zero, this must indicate a valid checksum (i.e., it does not indicate that the APC is not used; instead, the option would simply not be included if that were the desired effect).

APC does not protect the UDP pseudoheader; only the current UDP checksum provides that protection (when used). APC cannot provide that protection because it would need to be updated whenever the UDP pseudoheader changed, e.g., during NAT address and port translation; because this is not the case, APC does not cover the pseudoheader.

>> UDP packets with incorrect APC checksums MUST be passed to the application by default, e.g., with a flag indicating APC failure.

Like all safe UDP options, APC needs to be silently ignored when failing by default, unless the receiver has been configured to do otherwise. Although all UDP option-aware endpoints support APC (being in the required set), this silently-ignored behavior ensures that option-aware receivers operate the same as legacy receivers unless overridden.

>> UDP packets with unrecognized APC lengths MUST be receive the same treatment as UDP packets with incorrect APC checksums.

Ensuring that unrecognized APC lengths are treated as incorrect checksums enables future variants of APC to be treated as APC-like.

#### 9.4. Fragmentation (FRAG)

The Fragmentation (FRAG, Kind=3) option supports UDP fragmentation and reassembly, which can be used to transfer UDP messages larger than limited by the IP receive MTU (EMTU\_R [RFC1122]). FRAG includes a copy of the same UDP transport ports in each fragment, enabling them to traverse Network Address (and port) Translation (NAT) devices, in contrast to the behavior of IP fragments. FRAG is typically used with the UDP MDS and MRDS options to enable more efficient use of large messages, both at the UDP and IP layers. FRAG is designed similar to the IPv6 Fragmentation Header [RFC8200], except that the UDP variant uses a 16-bit Offset measured in bytes, rather than IPv6's 13-bit Fragment Offset measured in 8-byte units. This UDP variant avoids creating reserved fields.

>> When FRAG is present, it SHOULD come as early as possible in the UDP options list.

>> When FRAG is present, the UDP user data MUST be empty. If the user data is not empty, all UDP options MUST be silently ignored and the user data received sent to the user.

Legacy receivers interpret FRAG messages as zero-length user data UDP packets (i.e., UDP Length field is 8, the length of just the UDP header), which would not affect the receiver unless the presence of the UDP packet itself were a signal (see Section 5 of [RFC8085]). In this manner, the FRAG option also helps hide UNSAFE options so they can be used more safely in the presence of legacy receivers.

The FRAG option has two formats; non-terminal fragments use the shorter variant (Figure 10) and terminal fragments use the longer (Figure 11). The latter includes stand-alone fragments, i.e., when data is contained in the FRAG option but reassembly is not required.

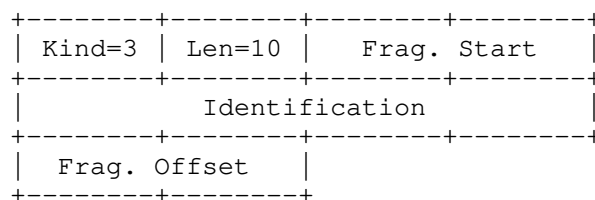


Figure 10 UDP non-terminal FRAG option format

In the non-terminal FRAG option format, Frag. Start indicates the location of the beginning of the fragment data, measured from the beginning of the UDP header of the fragment. The fragment data

follows the remainder of the UDP options and continues to the end of the IP datagram (i.e., the end of the surplus area). Those options are applied to this UDP fragment. Non-terminal fragments never have options after the fragment.

The Frag. Offset field indicates the location of this fragment relative to the original UDP datagram (prior to fragmentation), measured from the start of the original UDP datagram's UDP header.

The FRAG option does not need a "more fragments" bit because it provides the same indication by using the longer, 12-byte variant, as shown in Figure 11.

>> The FRAG option MAY be used on a single fragment, in which case the Frag. Offset would be zero and the option would have the 12-byte format.

>> Endpoints supporting UDP options MUST be capable of fragmenting and reassembling at least 2 fragments, for a total of at least 3,000 bytes (see MRDS in Section 9.6).

Use of the single fragment variant can be helpful in supporting use of UNSAFE options without undesirable impact to receivers that do not support either UDP options or the specific UNSAFE options.

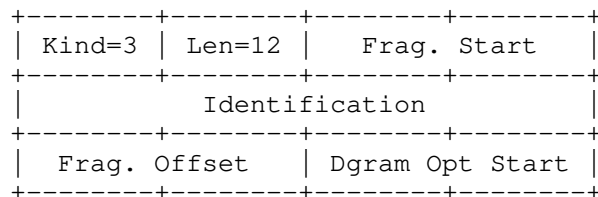


Figure 11 UDP terminal FRAG option format

The terminal FRAG option format adds a Datagram Option Start pointer, measured from the start of the original UDP datagram header, indicating the end of the reassembled data and the start of the surplus area after the original UDP datagram. In this variant, UDP options that apply to the reassembled datagram may occur after the terminal fragment data. UDP options that occur before the FRAG data are processed on the fragment; UDP options after the FRAG data are processed after reassembly, such that the reassembled data represents the original UDP user data. This allows either pre-reassembly or post-reassembly UDP option effects, such as using UENC on each fragment while also using TIME on the reassembled datagram for round-trip latency measurements.

>> During fragmentation, the UDP header checksum of each fragment remains constant and does not depend on the fragment data (which appears in the surplus area), because all fragments have a zero-length user data field.

The Fragment Offset is 16 bits and indicates the location of the UDP payload fragment in bytes from the beginning of the original unfragmented payload. The option Len field indicates whether there are more fragments (Len=10) or no more fragments (Len=12).

>> The Identification field is a 32-bit value that MUST be unique over the expected fragment reassembly timeout.

>> The Identification field SHOULD be generated in a manner similar to that of the IPv6 Fragment ID [RFC8200].

>> UDP fragments MUST NOT overlap.

Similar to IPv6 reassembly [RFC8200], if any of the fragments being reassembled overlap with any other fragments being reassembled for the same UDP packet, reassembly of that UDP packet must be abandoned and all the fragments that have been received for that UDP packet must be discarded, and no ICMP error messages should be sent.

It should be noted that fragments may be duplicated in the network. Instead of treating these exact duplicate fragments as overlapping fragments, an implementation may choose to detect this case and drop exact duplicate fragments while keeping the other fragments belonging to the same UDP packet.

UDP fragmentation relies on a fragment expiration timer, which can be preset or could use a value computed using the UDP Timestamp option.

>> The default UDP reassembly SHOULD be no more than 2 minutes.

>> UDP reassembly space SHOULD be limited to reduce the impact of DOS attacks on resource use.

>> UDP reassembly space limits SHOULD NOT be computed as a shared resource across multiple sockets, to avoid cross-socketpair DOS attacks.

>> Individual UDP fragments MUST NOT be forwarded to the user. The reassembled datagram is received only after complete reassembly, checksum validation, and continued processing of the remaining UDP options.

Any per-datagram UDP options, if used, follow the FRAG option in the final fragment and would be included in the reassembled UDP packet. Processing of those options would commence after reassembly. This is especially important for UNSAFE options, which are interpreted only after FRAG.

In general, UDP packets are fragmented as follows:

1. Create a UDP packet with data and UDP options, which we will call "D". Note that the UDP options treat the data area as UDP user data and thus must follow that data.

Process these UDP options before the rest of the fragmentation steps below. Note that the OCS value of the original packet SHOULD be zero if each fragment will have a non-zero OCS value (as will be the case if the UDP checksum is non-zero).

2. Identify the desired fragment size, which we will call "S". This value should take into account the path MTU (if known) and allow space for per-fragment options.
3. Fragment "D" into chunks of size no larger than "S"-10 each, with one final chunk no larger than "S"-12. Note that all the non-FRAG options in step #1 need not be limited to the terminal fragment, i.e., the Dgram Opt. Start pointer can indicate the start of the original surplus area anywhere in the reassembled data.
4. For each chunk of "D" in step #3, create a zero-data UDP packet followed by the word-aligned OCS, the FRAG option, and any additional UDP options, followed by the FRAG data chunk.

The last chunk includes the non-FRAG options noted in step #1 after the end of the FRAG data. These UDP options apply to the reassembled user data as a whole when received.

5. Process the pre-reassembly UDP options of each fragment.

Receivers reverse the above sequence. They process all received options in each fragment. When the FRAG option is encountered, the FRAG data is used in reassembly. After all fragments are received, the entire UDP packet is processed with any trailing UDP options applying to the reassembled user data.

#### 9.5. Maximum Datagram Size (MDS)

The Maximum Datagram Size (MDS, Kind=4) option is a 16-bit hint of the largest unfragmented UDP packet that an endpoint believes can be



received. As with the TCP Maximum Segment Size (MSS) option [RFC793], the size indicated is the IP layer MTU decreased by the fixed IP and UDP headers only [RFC6691]. The space needed for IP and UDP options need to be adjusted by the sender when using the value indicated. The value transmitted is based on EMTU\_R, the largest IP datagram that can be received (i.e., reassembled at the receiver) [RFC1122]. However, as with TCP, this value is only a hint at what the receiver believes; it does not indicate a known path MTU and thus MUST NOT be used to limit transmissions.

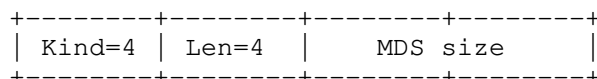


Figure 12 UDP MDS option format

The UDP MDS option MAY be used as a hint for path MTU discovery [RFC1191][RFC8201], but this may be difficult because of known issues with ICMP blocking [RFC2923] as well as UDP lacking automatic retransmission. It is more likely to be useful when coupled with IP source fragmentation or UDP fragmentation to limit the largest reassembled UDP message as indicated by MRDS (see Section 9.6), e.g., when EMTU\_R is larger than the required minimums (576 for IPv4 [RFC791] and 1500 for IPv6 [RFC8200]). It can also be used with DPLPMTUD [RFC8899] to provide a hint to maximum DPLPMTU, though it MUST NOT prohibit transmission of larger UDP packets (or fragments) used as DPLPMTU probes.

#### 9.6. Maximum Reassembled Datagram Size (MRDS)

The Maximum Reassembled Segment Size (MRDS, Kind=5) option is a 16-bit indicator of the largest reassembled UDP segment that can be received. MRDS is the UDP equivalent of IP's EMTU\_R but the two are not related [RFC1122]. Using the FRAG option (Section 9.4), UDP packets can be transmitted as transport fragments, each in their own (presumably not fragmented) IP datagram and be reassembled at the UDP layer.

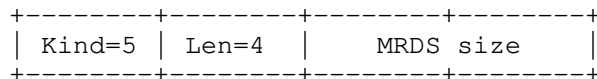


Figure 13 UDP MRDS option format

>> Endpoints supporting UDP options MUST support a local MRDS of at least 3,000 bytes.

### 9.7. Echo request (REQ) and echo response (RES)

The echo request (REQ, Kind=6) and echo response (RES, Kind=7) options provide a means for UDP options to be used to provide UDP packet-level acknowledgements. One such use is described as part of the UDP options variant of packetization layer path MTU discovery (PLPMTUD) [Fa22]. The options both have the format indicated in Figure 14, in which the token has no internal structure or meaning.

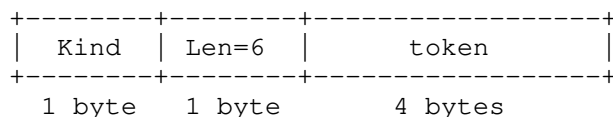


Figure 14 UDP REQ and RES options format

Each of these option kinds appears at most once in each UDP packet, as with other options. Note also that the FRAG option is not used when sending DPLPMTUD probes to determine a PLPMTU [Fa22].

### 9.8. Timestamps (TIME)

The Timestamp (TIME, Kind=8) option exchanges two four-byte unsigned timestamp fields. It serves a similar purpose to TCP's TS option [RFC7323], enabling UDP to estimate the round trip time (RTT) between hosts. For UDP, this RTT can be useful for establishing UDP fragment reassembly timeouts or transport-layer rate-limiting [RFC8085].

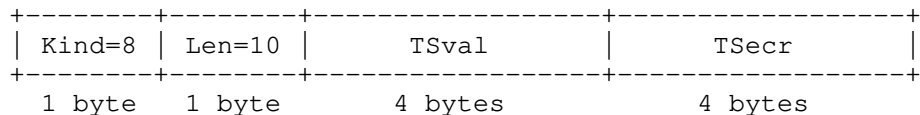


Figure 15 UDP TIME option format

TS Value (TSval) and TS Echo Reply (TSecr) are used in a similar manner to the TCP TS option [RFC7323]. On transmitted UDP packets using the option, TS Value is always set based on the local "time" value. Received TSval and TSecr values are provided to the application, which can pass the TSval value to be used as TSecr on UDP messages sent in response (i.e., to echo the received TSval). A received TSecr of zero indicates that the TSval was not echoed by the transmitter, i.e., from a previously received UDP packet.

>> TIME MAY use an RTT estimate based on nonzero Timestamp values as a hint for fragmentation reassembly, rate limiting, or other mechanisms that benefit from such an estimate.

>> an application MAY use TIME to compute this RTT estimate for further use by the user.

UDP timestamps are modeled after TCP timestamps and have similar expectations. In particular, they are expected to be:

- o Values are monotonic and non-decreasing except for anticipated number-space rollover events
- o Values should "increase" (allowing for rollover) according to a typical 'tick' time
- o A request is defined as TSval being non-zero and a reply is defined as TSecr being non-zero.
- o A receiver should always respond to a request with the highest TSval received (allowing for rollover), which is not necessarily the most recently received.

Rollover can be handled as a special case or more completely using sequence number extension [RFC9187], however zero values need to be avoided explicitly.

>> TIME values MUST NOT use zeros as valid time values, because they are used as indicators of requests and responses.

### 9.9. Authentication (AUTH)

The Authentication (AUTH, Kind=9) option is intended to allow UDP to provide a similar type of authentication as the TCP Authentication Option (TCP-AO) [RFC5925]. AUTH covers the UDP user data. AUTH supports NAT traversal in a similar manner as TCP-AO [RFC6978]. Figure 16 shows the UDP AUTH format, whose contents are identical to that of the TCP-AO option.

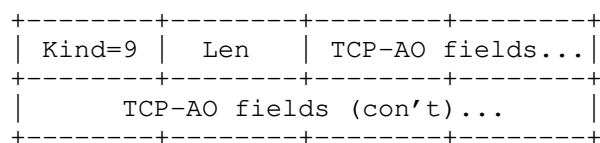


Figure 16 UDP AUTH option format

Like TCP-AO, AUTH is not negotiated in-band. Its use assumes both endpoints have populated Master Key Tuples (MKTs), used to exclude non-protected traffic.

TCP-AO generates unique traffic keys from a hash of TCP connection parameters. UDP lacks a three-way handshake to coordinate connection-specific values, such as TCP's Initial Sequence Numbers (ISNs) [RFC793], thus AUTH's Key Derivation Function (KDF) uses zeroes as the value for both ISNs. This means that the AUTH reuses keys when socket pairs are reused, unlike TCP-AO.

>> UDP packets with incorrect AUTH HMACs MUST be passed to the application by default, e.g., with a flag indicating AUTH failure.

Like all non-UNSAFE UDP options, AUTH needs to be silently ignored when failing. This silently-ignored behavior ensures that option-aware receivers operate the same as legacy receivers unless overridden.

In addition to the UDP user data (which is always included), AUTH can be configured to either include or exclude the surplus area, in a similar way as can TCP-AO can optionally exclude TCP options. When UDP options are covered, the OCS value and AUTH (and later, UENC) hash areas are zeroed before computing the AUTH hash. It is important to consider that options not yet defined might yield unpredictable results if not confirmed as supported, e.g., if they were to contain other hashes or checksums that depend on the surplus area contents. This is why such dependencies are not permitted except as defined for the OCS and the AUTH (and later, UENC) option.

Similar to TCP-AO-NAT, AUTH (and later, UENC) can be configured to support NAT traversal, excluding (by zeroing out) one or both of the UDP ports and corresponding IP addresses [RFC6978].

#### 9.10. Experimental (EXP)

The Experimental option (EXP, Kind=127) is reserved for experiments [RFC3692]. Only one such value is reserved because experiments are expected to use an Experimental ID (ExIDs) to differentiate concurrent use for different purposes, using UDP ExIDs registered with IANA according to the approach developed for TCP experimental options [RFC6994].

Kind=127	Len	UDP ExID
(option contents, as defined)...		

Figure 17 UDP EXP option format

>> The length of the experimental option MUST be at least 4 to account for the Kind, Length, and the minimum 16-bit UDP ExID identifier (similar to TCP ExIDs [RFC6994]).

The UDP EXP option also includes an extended length format, where the option LEN is 255 followed by two bytes of extended length.

Kind=127	255	Extended Length
UDP ExID.	(option contents...)	

Figure 18 UDP EXP option format

Assigned UDP experimental IDs (ExIDs) assigned from a single registry managed by IANA (see Section 23). Assigned ExIDs can be used in either the EXP or UEXP options (see Section 10.2 for the latter).

## 10. UNSAFE Options

UNSAFE options are not safe to ignore and can be used unidirectionally or without soft-state confirmation of UDP option capability. They are always used only when the user data occurs inside a reassembled set of one or more UDP fragments, such that if UDP fragmentation is not supported, the enclosed UDP user data would be silently dropped anyway.

>> Applications using UNSAFE options SHOULD NOT also use zero-length UDP packets as signals, because they will arrive when UNSAFE options fail. Those that choose to allow such packets MUST account for such events.

>> UNSAFE options MUST be used only as part of UDP fragments, used either per-fragment or after reassembly.

>> Receivers supporting UDP options MUST silently drop the UDP user data of the reassembled datagram if any fragment or the entire

datagram includes an UNSAFE option whose UKind is not supported. Note that this still results in the receipt of a zero-length UDP datagram.

#### 10.1. UNSAFE Encryption (UENC)

UNSAFE encryption (UENC, Kind=192) has the same format as AUTH (Section 9.9), except that it encrypts (modifies) the user data. It provides a similar encryption capability as TCP-AO-ENC, in a similar manner [Tol8]. Its fields, coverage, and processing are the same as for AUTH, except that UENC encrypts only the user data, although it can (optionally) depend on the surplus area (with certain fields zeroed, as per AUTH, e.g., providing authentication over the surplus area). Like AUTH, UENC can be configured to be compatible with NAT traversal.

#### 10.2. UNSAFE Experimental (UEXP)

The UNSAFE Experimental option (UEXP, Kind=254) is reserved for experiments [RFC3692]. As with EXP, only one such UEXP value is reserved because experiments are expected to use an Experimental ID (ExIDs) to differentiate concurrent use for different purposes, using UDP ExIDs registered with IANA according to the approach developed for TCP experimental options [RFC6994].

Assigned ExIDs can be used with either the UEXP or EXP options.

### 11. Rules for designing new options

The UDP option Kind space allows for the definition of new options, however the currently defined options do not allow for arbitrary new options. The following is a summary of rules for new options and their rationales:

>> New options MUST NOT modify other option content.

>> New options MUST NOT depend on the content of other options.

>> UNSAFE options can both depend on and vary user data content because they are contained only inside UDP fragments and thus are processed only by UDP option capable receivers.

>> New options MUST NOT declare their order relative to other options, whether new or old.

>> At the sender, new options MUST NOT modify UDP packet content anywhere except within their option field, excepting only those

contained within the UNSAFE option; areas that need to remain unmodified include the IP header, IP options, the UDP user data, and the surplus area (i.e., other options).

>> Options MUST NOT be modified in transit. This includes those already defined as well as new options.

>> New options MUST NOT require or intend optionally for modification of any UDP options, including their new areas, in transit.

Note that only certain of the initially defined options violate these rules:

- o >> Only FRAG and UNSAFE options are permitted to modify the UDP body.

The following recommendation helps enable efficient zero-copy processing:

- o >> FRAG SHOULD be the first option, when present.

## 12. Option inclusion and processing

The following rules apply to option inclusion by senders and processing by receivers.

>> Senders MAY add any option, as configured by the API.

>> All "must-support" options MUST be processed by receivers, if present (presuming UDP options are supported at that receiver).

>> Non-"must-support" options MAY be ignored by receivers, if present, e.g., based on API settings.

>> All options MUST be processed by receivers in the order encountered in the options area.

>> All options except UNSAFE options MUST result in the UDP user data being passed to the application layer, regardless of whether all options are processed, supported, or succeed.

The basic premise is that, for options-aware endpoints, the sender decides what options to add and the receiver decides what options to handle. Simply adding an option does not force work upon a receiver, with the exception of the "must-support" options.

Upon receipt, the receiver checks various properties of the UDP packet and its options to decide whether to accept or drop the UDP packet and whether to accept or ignore some its options as follows (in order):

```
if the UDP checksum fails then
    silently drop the entire UDP packet (per RFC1122)
if the UDP checksum passes then
    if OCS != 0 and fails or is zero when UDP CS != 0 then
        deliver the UDP user data but ignore other options
        (this is required to emulate legacy behavior)
    if OCS is nonzero and passes or is zero then
        deliver the UDP user data after parsing
        and processing the rest of the options,
        regardless of whether each is supported or succeeds
        (again, this is required to emulate legacy behavior)
```

The design of the UNSAFE options as used only inside the FRAG area ensures that the resulting UDP data will be silently dropped in both legacy and options-aware receivers. Again, note that this still results in the delivery of a zero-length UDP packet.

Options-aware receivers can drop UDP packets with option processing errors via either an override of the default UDP processing or at the application layer.

I.e., all options are treated the same, in that the transmitter can add it as desired and the receiver has the option to require it or not. Only if it is required (e.g., by API configuration) should the receiver require it being present and correct.

I.e., for all options:

- o if the option is not required by the receiver, then UDP packets missing the option are accepted.
- o if the option is required (e.g., by override of the default behavior at the receiver) and missing or incorrectly formed, silently drop the UDP packet.
- o if the UDP packet is accepted (either because the option is not required or because it was required and correct), then pass the option with the UDP packet via the API.



Any options whose length exceeds that of the UDP packet (i.e., intending to use data that would have been beyond the surplus area) should be silently ignored (again to model legacy behavior).

### 13. UDP API Extensions

UDP currently specifies an application programmer interface (API), summarized as follows (with Unix-style command as an example) [RFC768]:

- o Method to create new receive ports
  - o E.g., `bind(handle, recvaddr(optional), recvport)`
- o Receive, which returns data octets, source port, and source address
  - o E.g., `recvfrom(handle, srcaddr, srcport, data)`
- o Send, which specifies data, source and destination addresses, and source and destination ports
  - o E.g., `sendto(handle, destaddr, destport, data)`

This API is extended to support options as follows:

- o Extend the method to create receive ports to include per-packet and per-fragment receive options that are required as indicated by the application. Datagrams not containing these required options MUST be silently dropped and MAY be logged. This includes a minimum datagram length, such that the options list ends in EOL and additional space is zero-filled as needed.
- o WG QUESTION: DO WE ALSO WANT A MIN FRAG SIZE? OR MAX?
- o Extend the receive function to indicate the per-packet options and their parameters as received with the corresponding received datagram. Note that per-fragment options are handled within the processing of each fragment.
- o WG QUESTION: SHOULD WE ACCUMULATE THOSE OPTIONS? OR DISCARD THEM?
- o Extend the send function to indicate the options to be added to the corresponding sent datagram. This includes indicating which options apply to individual fragments vs. which apply to the UDP packet prior to fragmentation, if fragmentation is enabled.

Examples of API instances for Linux and FreeBSD are provided in Appendix A, to encourage uniform cross-platform implementations.

#### 14. UDP Options are for Transport, Not Transit

UDP options are indicated in the surplus area of the IP payload that is not used by UDP. That area is really part of the IP payload, not the UDP payload, and as such, it might be tempting to consider whether this is a generally useful approach to extending IP.

Unfortunately, the surplus area exists only for transports that include their own transport layer payload length indicator. TCP and SCTP include header length fields that already provide space for transport options by indicating the total length of the header area, such that the entire remaining area indicated in the network layer (IP) is transport payload. UDP-Lite already uses the UDP Length field to indicate the boundary between data covered by the transport checksum and data not covered, and so there is no remaining area where the length of the UDP-Lite payload as a whole can be indicated [RFC3828].

UDP options are intended for use only by the transport endpoints. They are no more (or less) appropriate to be modified in-transit than any other portion of the transport datagram.

UDP options are transport options. Generally, transport headers, options, and data are not intended to be modified in-transit. UDP options are no exception and here are specified as "MUST NOT" be altered in transit. However, the UDP option mechanism provides no specific protection against in-transit modification of the UDP header, UDP payload, or surplus area, except as provided by the OCS or the options selected (e.g., AUTH, or UENC).

#### 15. UDP options vs. UDP-Lite

UDP-Lite provides partial checksum coverage, so that UDP packets with errors in some locations can be delivered to the user [RFC3828]. It uses a different transport protocol number (136) than UDP (17) to interpret the UDP Length field as the prefix covered by the UDP checksum.

UDP (protocol 17) already defines the UDP Length field as the limit of the UDP checksum, but by default also limits the data provided to the application as that which precedes the UDP Length. A goal of UDP-Lite is to deliver data beyond UDP Length as a default, which is why a separate transport protocol number was required.

UDP options do not use or need a separate transport protocol number because the data beyond the UDP Length offset (surplus data) is not provided to the application by default. That data is interpreted exclusively within the UDP transport layer.

UDP-Lite cannot support UDP options, either as proposed here or in any other form, because the entire payload of the UDP packet is already defined as user data and there is no additional field in which to indicate a surplus area for options. The UDP Length field in UDP-Lite is already used to indicate the boundary between user data covered by the checksum and user data not covered.

#### 16. Interactions with Legacy Devices

It has always been permissible for the UDP Length to be inconsistent with the IP transport payload length [RFC768]. Such inconsistency has been utilized in UDP-Lite using a different transport number. There are no known systems that use this inconsistency for UDP [RFC3828]. It is possible that such use might interact with UDP options, i.e., where legacy systems might generate UDP datagrams that appear to have UDP options. The OCS provides protection against such events and is stronger than a static "magic number".

UDP options have been tested as interoperable with Linux, macOS, and Windows Cygwin, and worked through NAT devices. These systems successfully delivered only the user data indicated by the UDP Length field and silently discarded the surplus area.

One reported embedded device passes the entire IP datagram to the UDP application layer. Although this feature could enable application-layer UDP option processing, it would require that conventional UDP user applications examine only the UDP user data. This feature is also inconsistent with the UDP application interface [RFC768] [RFC1122].

It has been reported that Alcatel-Lucent's "Brick" Intrusion Detection System has a default configuration that interprets inconsistencies between UDP Length and IP Length as an attack to be reported. Note that other firewall systems, e.g., CheckPoint, use a default "relaxed UDP length verification" to avoid falsely interpreting this inconsistency as an attack.

#### 17. Options in a Stateless, Unreliable Transport Protocol

There are two ways to interpret options for a stateless, unreliable protocol -- an option is either local to the message or intended to

affect a stream of messages in a soft-state manner. Either interpretation is valid for defined UDP options.

It is impossible to know in advance whether an endpoint supports a UDP option.

>> All UDP options other than UNSAFE ones MUST be ignored if not supported or upon failure (e.g., APC).

>> All UDP options that fail MUST result in the UDP data still being sent to the application layer by default, to ensure equivalence with legacy devices.

>> UDP options that rely on soft-state exchange MUST allow for message reordering and loss.

The above requirements prevent using any option that cannot be safely ignored unless it is hidden inside the FRAG area (i.e., UNSAFE options). Legacy systems also always need to be able to interpret the transport fragments as individual UDP packets.

#### 18. UDP Option State Caching

Some TCP connection parameters, stored in the TCP Control Block, can be usefully shared either among concurrent connections or between connections in sequence, known as TCP Sharing [RFC9040]. Although UDP is stateless, some of the options proposed herein may have similar benefit in being shared or cached. We call this UCB Sharing, or UDP Control Block Sharing, by analogy. Just as TCB sharing is not a standard because it is consistent with existing TCP specifications, UCB sharing would be consistent with existing UDP specifications, including this one. Both are implementation issues that are outside the scope of their respective specifications, and so UCB sharing is outside the scope of this document.

#### 19. Updates to RFC 768

This document updates RFC 768 as follows:

- o This document defines the meaning of the IP payload area beyond the UDP length but within the IP length as the surplus area used herein for UDP options.
- o This document extends the UDP API to support the use of UDP options.

## 20. Interactions with other RFCs (and drafts)

This document clarifies the interaction between UDP Length and IP length that is not explicitly constrained in either UDP or the host requirements [RFC768] [RFC1122].

Teredo extensions (TE) define use of a similar difference between these lengths for trailers [RFC6081]. TE defines the UDP length pointing beyond (larger) than the location indicated by the IP length rather than shorter (as used herein):

"..the IPv6 packet length (i.e., the Payload Length value in the IPv6 header plus the IPv6 header size) is less than or equal to the UDP payload length (i.e., the Length value in the UDP header minus the UDP header size)"

As a result, UDP options are not compatible with TE, but that is also why this document does not update TE. Additionally, it is not at all clear how TE operates, as it requires network processing of the UDP length field to understand the total message including TE trailers.

TE updates Teredo NAT traversal [RFC4380]. The NAT traversal document defined "consistency" of UDP length and IP length as:

"An IPv6 packet is deemed valid if it conforms to [RFC2460]: the protocol identifier should indicate an IPv6 packet and the payload length should be consistent with the length of the UDP datagram in which the packet is encapsulated."

IPv6 is clear on the meaning of this consistency, in which the pseudoheader used for UDP checksums is based on the UDP length, not inferred from the IP length, using the same text in the current specification [RFC8200]:

"The Upper-Layer Packet Length in the pseudo-header is the length of the upper-layer header and data (e.g., TCP header plus TCP data). Some upper-layer protocols carry their own length information (e.g., the Length field in the UDP header); for such protocols, that is the length used in the pseudo-header."

This document is consistent the UDP profile for Robust Header Compression (ROHC) [RFC3095], noted here:

"The Length field of the UDP header MUST match the Length field(s) of the preceding subheaders, i.e., there must not

be any padding after the UDP payload that is covered by the IP Length."

ROHC compresses UDP headers only when this match succeeds. It does not prohibit UDP headers where the match fails; in those cases, ROHC default rules (Section 5.10) would cause the UDP header to remain uncompressed. Upon receipt of a compressed UDP header, Section A.1.3 of that document indicates that the UDP length is "INFERRED"; in uncompressed packets, it would simply be explicitly provided.

This issue of handling UDP header compression is more explicitly described in more recent specifications, e.g., Sec. 10.10 of Static Context Header Compression [RFC8724].

## 21. Multicast Considerations

UDP options are primarily intended for unicast use. Using these options over multicast IP requires careful consideration, e.g., to ensure that the options used are safe for different endpoints to interpret differently (e.g., either to support or silently ignore) or to ensure that all receivers of a multicast group confirm support for the options in use.

## 22. Security Considerations

There are a number of security issues raised by the introduction of options to UDP. Some are specific to this variant, but others are associated with any packet processing mechanism; all are discussed in this section further.

The use of UDP packets with inconsistent IP and UDP Length fields has the potential to trigger a buffer overflow error if not properly handled, e.g., if space is allocated based on the smaller field and copying is based on the larger. However, there have been no reports of such vulnerability and it would rely on inconsistent use of the two fields for memory allocation and copying.

UDP options are not covered by DTLS (datagram transport-layer security). Despite the name, neither TLS [RFC8446] (transport layer security, for TCP) nor DTLS [RFC6347] (TLS for UDP) protect the transport layer. Both operate as a shim layer solely on the user data of transport packets, protecting only their contents. Just as TLS does not protect the TCP header or its options, DTLS does not protect the UDP header or the new options introduced by this document. Transport security is provided in TCP by the TCP Authentication Option (TCP-AO [RFC5925]) or in UDP by the Authentication (AUTH) option (Section 9.9) and UNSAFE Encryption

(UENC) option (Section 10). Transport headers are also protected as payload when using IP security (IPsec) [RFC4301].

UDP options use the TLV syntax similar to that of TCP. This syntax is known to require serial processing and may pose a DOS risk, e.g., if an attacker adds large numbers of unknown options that must be parsed in their entirety, as is the case for IPv6 [RFC8504].

>> Implementations concerned with the potential for this vulnerability MAY implement only the required UDP options and MAY also limit processing of TLVs, either in number of non-padding options or total length, or both. The number of non-zero TLVs allowed in such cases MUST be at least 8.

Because required options come first and at most once each (with the exception of NOPs, which should never need to come in sequences of more than seven in a row), this limits their DOS impact. Note that TLV formats for options does require serial processing, but any format that allows future options, whether ignored or not, could introduce a similar DOS vulnerability.

UDP security should never rely solely on transport layer processing of options. UNSAFE options are the only type that share fate with the UDP data, because of the way that data is hidden in the surplus area until after those options are processed. All other options default to being silently ignored at the transport layer but may be dropped either if that default is overridden (e.g., by configuration) or discarded at the application layer (e.g., using information about the options processed that are passed along with the UDP packet).

UDP fragmentation introduces its own set of security concerns, which can be handled in a manner similar to IP reassembly or TCP segment reordering [CERT18]. In particular, the number of UDP packets pending reassembly and effort used for reassembly is typically limited. In addition, it may be useful to assume a reasonable minimum fragment size, e.g., that non-terminal fragments should never be smaller than 500 bytes.

### 23. IANA Considerations

Upon publication, IANA is hereby requested to create a new registry for UDP Option Kind numbers, similar to that for TCP Option Kinds. Initial values of this registry are as listed in Section 8. Additional values in this registry are to be assigned from the UNASSIGNED values in Section 8 by IESG Approval or Standards Action

[RFC8126]. Those assignments are subject to the conditions set forth in this document, particularly (but not limited to) those in Section 11.

Although option nicknames are not used in-band, IANA should require UNSAFE safe option values to commence with the letter "U" and avoid that letter as commencing safe options.

Upon publication, IANA is hereby requested to create a new registry for UDP Experimental Option Experiment Identifiers (UDP ExIDs) for use in a similar manner as TCP ExIDs [RFC6994]. UDP ExIDs can be used in either (or both) the EXP or UEXP options. This registry is initially empty. Values in this registry are to be assigned by IANA using first-come, first-served (FCFS) rules [RFC8126]. Options using these ExIDs are subject to the same conditions as new options, i.e., they too are subject to the conditions set forth in this document, particularly (but not limited to) those in Section 11.

## 24. References

### 24.1. Normative References

- [Fa22] Fairhurst, G., T. Jones, "Datagram PLPMTUD for UDP Options," draft-ietf-tsvwg-udp-options-dplpmtud, Feb. 2022.
- [RFC768] Postel, J., "User Datagram Protocol," RFC 768, August 1980.
- [RFC791] Postel, J., "Internet Protocol," RFC 791, Sept. 1981.
- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts -- Communication Layers," RFC 1122, Oct. 1989.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words," RFC 2119, May 2017.

### 24.2. Informative References

- [Fa18] Fairhurst, G., T. Jones, R. Zullo, "Checksum Compensation Options for UDP Options", draft-fairhurst-udp-options-cco, Oct. 2018.



- [Hil15] Hildebrand, J., B. Trammel, "Substrate Protocol for User Datagrams (SPUD) Prototype," draft-hildebrand-spud-prototype-03, Mar. 2015.
- [RFC793] Postel, J., "Transmission Control Protocol" RFC 793, September 1981.
- [RFC1071] Braden, R., D. Borman, C. Partridge, "Computing the Internet Checksum," RFC 1071, Sept. 1988.
- [RFC1191] Mogul, J., S. Deering, "Path MTU discovery," RFC 1191, November 1990.
- [RFC2923] Lahey, K., "TCP Problems with Path MTU Discovery," RFC 2923, September 2000.
- [RFC3095] Bormann, C. (Ed), et al., "RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed," RFC 3095, July 2001.
- [RFC3385] Sheinwald, D., J. Satran, P. Thaler, V. Cavanna, "Internet Protocol Small Computer System Interface (iSCSI) Cyclic Redundancy Check (CRC)/Checksum Considerations," RFC 3385, Sep. 2002.
- [RFC3692] Narten, T., "Assigning Experimental and Testing Numbers Considered Useful," RFC 3692, Jan. 2004.
- [RFC3828] Larzon, L-A., M. Degermark, S. Pink, L-E. Jonsson (Ed.), G. Fairhurst (Ed.), "The Lightweight User Datagram Protocol (UDP-Lite)," RFC 3828, July 2004.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, Dec. 2005.
- [RFC4340] Kohler, E., M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, March 2006.
- [RFC4380] Huitema, C., "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)," RFC 4380, Feb. 2006.
- [RFC4960] Stewart, R. (Ed.), "Stream Control Transmission Protocol", RFC 4960, September 2007.
- [RFC5925] Touch, J., A. Mankin, R. Bonica, "The TCP Authentication Option," RFC 5925, June 2010.

- [RFC6081] Thaler, D., "Teredo Extensions," RFC 6081, Jan 2011.
- [RFC6347] Rescorla, E., N. Modadugu, "Datagram Transport Layer Security Version 1.2," RFC 6347, Jan. 2012.
- [RFC6691] Borman, D., "TCP Options and Maximum Segment Size (MSS)," RFC 6691, July 2012.
- [RFC6935] Eubanks, M., P. Chimento, M. Westerlund, "IPv6 and UDP Checksums for Tunneled Packets," RFC 6935, April 2013.
- [RFC6978] Touch, J., "A TCP Authentication Option Extension for NAT Traversal", RFC 6978, July 2013.
- [RFC6994] Touch, J., "Shared Use of Experimental TCP Options," RFC 6994, Aug. 2013.
- [RFC7323] Borman, D., R. Braden, V. Jacobson, R. Scheffenegger (Ed.), "TCP Extensions for High Performance," RFC 7323, Sep. 2014.
- [RFC8085] Eggert, L., G. Fairhurst, G. Shepherd, "UDP Usage Guidelines," RFC 8085, Feb. 2017.
- [RFC8126] Cotton, M., B. Leiba, T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs," RFC 8126, June 2017.
- [RFC8200] Deering, S., R. Hinden, "Internet Protocol Version 6 (IPv6) Specification," RFC 8200, Jul. 2017.
- [RFC8201] McCann, J., S. Deering, J. Mogul, R. Hinden (Ed.), "Path MTU Discovery for IP version 6," RFC 8201, Jul. 2017.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018.
- [RFC8504] Chown, T., J. Loughney, T. Winters, "IPv6 Node Requirements," RFC 8504, Jan. 2019.
- [RFC8724] Minaburo, A., L. Toutain, C. Gomez, D. Barthel, JC., "SCHC: Generic Framework for Static Context Header Compression and Fragmentation," RFC 8724, Apr. 2020.
- [RFC8899] Fairhurst, G., T. Jones, M. Tuxen, I. Rungeler, T. Volker, "Packetization Layer Path MTU Discovery for Datagram Transports," RFC 8899, Sep. 2020.

- [RFC9040] Touch, J., M. Welzl, S. Islam, "TCP Control Block Interdependence," RFC 9040, Jul. 2021.
- [RFC9187] Touch, J., "Sequence Number Extension for Windowed Protocols," RFC 9187, Jan. 2022.
- [CERT18] CERT Coordination Center, "TCP implementations vulnerable to Denial of Service," Vulnerability Note VU 962459, Software Engineering Institute, CMU, 2018, <https://www.kb.cert.org/vuls/id/962459>.
- [To18] Touch, J., "A TCP Authentication Option Extension for Payload Encryption," draft-touch-tcp-ao-encrypt, Jul. 2018.

## 25. Acknowledgments

This work benefitted from feedback from Erik Auerswald, Bob Briscoe, Ken Calvert, Ted Faber, Gorry Fairhurst (including OCS for misbehaving middlebox traversal), C. M. Heard (including combining previous FRAG and LITE options into the new FRAG), Tom Herbert, Mark Smith, and Raffaele Zullo, as well as discussions on the IETF TSVWG and SPUD email lists.

This work was partly supported by USC/ISI's Postel Center.

This document was prepared using 2-Word-v2.0.template.dot.

## Authors' Addresses

Joe Touch  
Manhattan Beach, CA 90266 USA

Phone: +1 (310) 560-0334  
Email: [touch@strayalpha.com](mailto:touch@strayalpha.com)

## Appendix A. Implementation Information

The following information is provided to encourage interoperable API implementations.

System-level variables (sysctl):

Name	default	meaning
net.ipv4.udp_opt	0	UDP options available
net.ipv4.udp_opt_ocs	1	Default use OCS
net.ipv4.udp_opt_apc	0	Default include APC
net.ipv4.udp_opt_frag	0	Default fragment
net.ipv4.udp_opt_mds	0	Default include MDS
net.ipv4.udp_opt_mrds	0	Default include MRDS
net.ipv4.udp_opt_req	0	Default include REQ
net.ipv4.udp_opt_resp	0	Default include RES
net.ipv4.udp_opt_time	0	Default include TIME
net.ipv4.udp_opt_auth	0	Default include AUTH
net.ipv4.udp_opt_exp	0	Default include EXP
net.ipv4.udp_opt_uenc	0	Default include UENC
net.ipv4.udp_opt_uexp	0	Default include UEXP

Socket options (sockopt), cached for outgoing datagrams:

Name	meaning
UDP_OPT	Enable UDP options (at all)
UDP_OPT_OCS	Use UDP OCS
UDP_OPT_APC	Enable UDP APC option
UDP_OPT_FRAG	Enable UDP fragmentation
UDP_OPT_MDS	Enable UDP MDS option
UDP_OPT_MRDS	Enable UDP MRDS option
UDP_OPT_REQ	Enable UDP REQ option
UDP_OPT_RES	Enable UDP RES option
UDP_OPT_TIME	Enable UDP TIME option
UDP_OPT_AUTH	Enable UDP AUTH option
UDP_OPT_EXP	Enable UDP EXP option
UDP_OPT_UENC	Enable UDP UENC option
UDP_OPT_UEXP	Enable UDP UEXP option

Send/sendto parameters:

Connection parameters (per-socketpair cached state, part UCB):

Name	Initial value
-----	
opts_enabled	net.ipv4.udp_opt
ocs_enabled	net.ipv4.udp_opt_ocs

The following option is included for debugging purposes, and MUST NOT be enabled otherwise.

#### System variables

net.ipv4.udp\_opt\_junk    0

## System-level variables (sysctl):

Name	default	meaning
-----		
net.ipv4.udp_opt_junk	0	Default use of junk

## Socket options (sockopt):

Name	params	meaning
-----		
UDP_JUNK	-	Enable UDP junk option
UDP_JUNK_VAL	fillval	Value to use as junk fill
UDP_JUNK_LEN	length	Length of junk payload in bytes

## Connection parameters (per-socketpair cached state, part UCB):

Name	Initial value
-----	
junk_enabled	net.ipv4.udp_opt_junk
junk_value	0xABCD
junk_len	4



Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: May 7, 2020

E. Kinnear  
T. Pauly  
Apple Inc.  
November 4, 2019

Using HTTP/2 as a Transport for Arbitrary Bytestreams  
draft-kinnear-httpbis-http2-transport-02

Abstract

HTTP/2 provides multiplexing of HTTP requests over a single underlying transport connection. HTTP/2 Transport defines the use of the bidirectional extended CONNECT handshake to negotiate the use of application protocols using streams of an HTTP/2 connection as transport.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 7, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.



## Table of Contents

1. Introduction . . . . .	2
1.1. Notational Conventions . . . . .	2
2. The SETTINGS_ENABLE_BIDIRECTIONAL_CONNECT Parameter . . . . .	3
3. Negotiating Bidirectional Transport . . . . .	3
3.1. Initiating the Extended CONNECT Handshake . . . . .	4
3.2. Responding to the Extended CONNECT Handshake . . . . .	4
4. Using Tunnels Established via the Extended CONNECT Handshake . . . . .	5
4.1. Example . . . . .	5
5. IANA Considerations . . . . .	6
6. Security Considerations . . . . .	7
7. Acknowledgments . . . . .	7
8. Normative References . . . . .	7
Authors' Addresses . . . . .	8

## 1. Introduction

HTTP/2 [RFC7540] provides a framing layer that describes the exchange of HTTP messages. This framing layer includes multiplexing of multiple streams on a single underlying transport connection, flow control, stream dependencies and priorities, and exchange of configuration information between endpoints.

Section 8.3 of [RFC7540] defines the HTTP CONNECT method for HTTP/2, which converts a HTTP/2 stream into a tunnel for arbitrary data. [RFC8441] describes the use of the extended CONNECT method to negotiate the use of the WebSocket Protocol [RFC6455] on an HTTP/2 stream.

This document extends the CONNECT handshake to allow both endpoints of an HTTP/2 connection to establish streams that tunnel data. It also defines a protocol name for use in the extended CONNECT handshake that allow negotiation of HTTP/2 streams that transport arbitrary bytestreams. Being able to transport application protocol data on individual HTTP/2 streams allows an underlying connection to be shared by multiple protocols and allows all protocols to benefit from the features provided by HTTP/2 framing.

## 1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 2. The SETTINGS\_ENABLE\_BIDIRECTIONAL\_CONNECT Parameter

As described in Section 5.5 of [RFC7540], SETTINGS parameters allow endpoints to negotiate use of protocol extensions that would otherwise generate protocol errors. Use of the CONNECT method extension defined in [RFC6455] requires the SETTINGS\_ENABLE\_CONNECT\_PROTOCOL parameter to be received by a client prior to its use.

This document introduces another SETTINGS parameter, SETTINGS\_ENABLE\_BIDIRECTIONAL\_CONNECT, which MUST have a value of 0 or 1.

Once a SETTINGS\_ENABLE\_BIDIRECTIONAL\_CONNECT parameter has been sent with a value of 1, an endpoint MUST NOT send the parameter with a value of 0.

Upon receipt of SETTINGS\_ENABLE\_BIDIRECTIONAL\_CONNECT with a value of 1, an endpoint MAY use the extended CONNECT defined in [RFC6455] with the protocol values defined in this document. An endpoint that supports receiving the extended CONNECT method SHOULD send this setting with a value of 1.

Note that [RFC6455] restricts SETTINGS\_ENABLE\_CONNECT\_PROTOCOL to have no effect if received by a server. This document modifies that restriction and allows both SETTINGS\_ENABLE\_CONNECT\_PROTOCOL and SETTINGS\_ENABLE\_BIDIRECTIONAL\_CONNECT to take effect if received by either endpoint of an HTTP/2 connection.

## 3. Negotiating Bidirectional Transport

[RFC6455] defines the pseudo-header field :protocol which can indicate the protocol intended to be used on the tunnel established by the CONNECT method. Values for the :protocol pseudo-header field are maintained in an Upgrade Token Registry established by [RFC7230] for protocol-name tokens.

After receiving both SETTINGS\_ENABLE\_CONNECT\_PROTOCOL and SETTINGS\_ENABLE\_BIDIRECTIONAL\_CONNECT, either endpoint of an HTTP/2 connection can send a request in HEADERS frames to establish a new stream via the extended CONNECT method. Similarly, either endpoint may be required to respond to an incoming CONNECT request seeking to establish such a stream.

### 3.1. Initiating the Extended CONNECT Handshake

Endpoints using this mechanism to establish bidirectional transport over HTTP/2 streams follow the CONNECT handshake procedure defined in [RFC6455]. However, instead of supplying "websocket" for the :protocol psuedo-header field to indicate a WebSocket connection, they negotiate the use of a specific application protocol by specifying an appropriate value. This document registers "bytestream" as a value to be used when an out-of-band negotiation has already occurred and an application protocol wishes to transport arbitrary bytes on an HTTP/2 stream. Any endpoint supplying "bytestream" as a value for the :protocol psuedo-header MUST have previously negotiated the use of this value via another mechanism.

The :scheme and :path psuedo-headers are required by [RFC6455]. The scheme of the target URI MUST be set to "https" for all :protocol values. The path is used in the same manner as for the WebSocket protocol, and MAY be set to "/" (an empty path component) if not desired for use.

Implementations should note that the Origin, Sec-WebSocket-Version, Sec-WebSocket-Protocol, and Sec-WebSocket-Extensions header fields are not included in the CONNECT request and response header fields, since this handshake mechanism is not being used to negotiate a WebSocket connection.

If the response to the extended CONNECT request indicates success of the handshake, then all further data sent or received on the new HTTP/2 stream is considered to be that of the supplied :protocol value and follows the semantics defined by that protocol.

### 3.2. Responding to the Extended CONNECT Handshake

A recipient of the extended CONNECT method follows the same procedure outlined by [RFC8441].

If the recipient encounters a :protocol psuedo-header with an unknown value or a value corresponding to a protocol they do not support, or if the recipient encounters violations of the extended CONNECT handshake protocol, they MUST return an HTTP response with an appropriate error code, such as 400 Bad Request. Otherwise, unknown header fields are ignored.

Once the handshake has been validated and is considered successful, the responder sends a HTTP response with status 200. After that response, all further data sent or received on the new HTTP/2 stream is considered to be of the supplied :protocol value.

#### 4. Using Tunnels Established via the Extended CONNECT Handshake

DATA frames are used as usual on the stream established by the CONNECT handshake to transmit data.

If the application negotiated the "bytestream" protocol, then individual DATA frames represent segments of an in-order byte stream and are delivered to the application as a stream of bytes. Implementations can deliver data to the application as soon as it becomes available, since there are no message boundaries to preserve.

The same considerations around intermediaries as defined in Section 7 of [RFC6455] apply to the extended CONNECT method. A client that connects via HTTP/2 to an HTTP proxy SHOULD use a traditional CONNECT request to tunnel through that proxy to the destination server.

Streams created via the extended CONNECT method participate in flow control, stream prioritization, and other HTTP/2 features in the same manner as request and response streams defined in [RFC7540]. Stream closure continues to be interpreted as defined in Section 5 of [RFC8441].

Note that the frame type restrictions defined in Section 8.3 of [RFC7540] remain in effect: only DATA, RST\_STREAM, WINDOW\_UPDATE, and PRIORITY frames are allowed on the connected streams and any other frame types MUST be treated as a stream error (Section 5.4.2 of [RFC7540]) if received.

##### 4.1. Example

An example of negotiating a "bytestream" stream on an HTTP/2 connection follows. This example is intended to closely follow the example in Section 5.1 of [RFC8441] to help illustrate the minor differences defined in this document.

```
[[ From Client ]]
```

```
[[ From Server ]]
```

```
SETTINGS
```

```
SETTINGS_ENABLE_CONNECT[..] = 1
```

```
SETTINGS_ENABLE_BIDIRECTIONAL[..] = 1
```

```
SETTINGS
```

```
SETTINGS_ENABLE_CONNECT[..] = 1
```

```
SETTINGS_ENABLE_BIDIRECTIONAL[..] = 1
```

```
HEADERS + END_HEADERS
```

```
:method = CONNECT
```

```
:protocol = bytestream
```

```
:scheme = https
```

```
:path = /
```

```
:authority = server.example.com
```

```
HEADERS + END_HEADERS
```

```
:status = 200
```

```
DATA
```

```
Bytestream Data
```

```
DATA + END_STREAM
```

```
Bytestream Data
```

```
DATA + END_STREAM
```

```
Bytestream Data
```

## 5. IANA Considerations

This specification registers an entry in the "HTTP Upgrade Tokens" registry that was established by [RFC7230].

A new token, "bytestream", for arbitrary bytestream data.

- o Value: bytestream
- o Description: Arbitrary bidirectional bytestream data
- o Expected Version Tokens:
- o References: [[RFC Editor: Please fill in this value with the RFC number for this document.]]

## 6. Security Considerations

The tunnels established by the CONNECT handshake are expected to be protected with a TLS connection. They inherit the security properties of this cryptographic context.

The security considerations of [RFC8441] Section 8 and [RFC7540] Section 10, and Section 10.5.2 especially, apply to this use of the CONNECT method.

## 7. Acknowledgments

Thanks to Anthony Chivetta, Joshua Otto, and Valentin Pistol for their contributions in the design and implementation of this work.

## 8. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<https://www.rfc-editor.org/info/rfc6455>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<https://www.rfc-editor.org/info/rfc7540>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8441] McManus, P., "Bootstrapping WebSockets with HTTP/2", RFC 8441, DOI 10.17487/RFC8441, September 2018, <<https://www.rfc-editor.org/info/rfc8441>>.

Authors' Addresses

Eric Kinnear  
Apple Inc.  
One Apple Park Way  
Cupertino, California 95014  
United States of America

Email: [ekinnear@apple.com](mailto:ekinnear@apple.com)

Tommy Pauly  
Apple Inc.  
One Apple Park Way  
Cupertino, California 95014  
United States of America

Email: [tpauly@apple.com](mailto:tpauly@apple.com)

TSVWG  
Internet-Draft  
Intended status: Informational  
Expires: January 14, 2021

Y. Li  
X. Zhou  
Huawei  
M. Boucadair  
Orange  
J. Wang  
China Telecom  
F. Qin  
China Mobile  
July 13, 2020

LOOPS (Localized Optimizations on Path Segments) Problem Statement and  
Opportunities for Network-Assisted Performance Enhancement  
draft-li-tsvwg-loops-problem-opportunities-06

Abstract

In various network deployments, end to end forwarding paths are partitioned into multiple segments. For example, in some cloud-based WAN communications, stitching multiple overlay tunnels are used for traffic policy enforcement matters such as to optimize traffic distribution or to select paths exposing a lower latency. Likewise, in satellite communications, the communication path is decomposed into two terrestrial segments and a satellite segment. Such long-haul paths are naturally composed of multiple network segments with various encapsulation schemes. Packet loss may show different characteristics on different segments.

Traditional transport protocols (e.g., TCP) respond to packet loss slowly especially in long-haul networks: they either wait for some signal from the receiver to indicate a loss and then retransmit from the sender or rely on sender's timeout which is often quite long. With the increase of end-to-end transport encryption (e.g., QUIC), traditional PEP (performance enhancing proxy) techniques such as TCP splitting are no longer applicable.

LOOPS (Local Optimizations on Path Segments) is a network-assisted performance enhancement over path segment and it aims to provide local in-network recovery to achieve better data delivery by making packet loss recovery faster. In an overlay network scenario, LOOPS can be performed over a variety of the existing, or purposely created, tunnel-based path segments.



## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 14, 2021.

## Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. The Problem and Opportunity Overview . . . . .	3
1.2. Sketching a Work Direction: Rationale & Goals . . . . .	4
2. Terminology . . . . .	5
3. Usage Scenarios . . . . .	6
3.1. Cloud-Internet Overlay Network . . . . .	6
3.2. Satellite Communication . . . . .	8
3.3. Branch Office WAN Connection . . . . .	9
4. Impact of Packet loss . . . . .	10
4.1. Tail Loss or Loss in Short Flows . . . . .	10
4.2. Packet Loss in Real Time Media Streams . . . . .	11
5. Features to be Considered for LOOPS . . . . .	11
5.1. Local Recovery . . . . .	11
5.2. Congestion Control Interaction . . . . .	12

5.3. Overlay Protocol Extensions . . . . .	12
6. Local in-network Recovery and End-to-end Retransmission . . .	13
7. Summary . . . . .	14
8. Security Considerations . . . . .	15
9. IANA Considerations . . . . .	15
10. Acknowledgements . . . . .	15
11. Informative References . . . . .	15
Authors' Addresses . . . . .	18

## 1. Introduction

### 1.1. The Problem and Opportunity Overview

Packet loss is ubiquitous in Internet. A reliable transport layer normally employs some end-to-end retransmission mechanisms which also address congestion control [RFC0793] [RFC5681]. The sender either waits for the receiver to send some signals on a packet loss or sets some form of timeout for retransmission. For unreliable transport protocols such as RTP [RFC3550], optional and limited usage of end-to-end retransmission is employed to recover from packet loss [RFC4585] [RFC4588]. End-to-end retransmission to recover lost packets is slow especially when the network is long-haul. For short-lived flows and transactional flows, latency suffers a lot from tail loss.

Tunnels are widely deployed within many networks to achieve various engineering goals, including long-haul WAN interconnection or enterprise wireless access networks. A connection between two endpoints can be decomposed into many connection legs. As such, the corresponding forwarding path can be partitioned into multiple path segments that some of them are using network overlays by means of tunnels. This design serves a number of purposes such as steering the traffic, optimizing egress/ingress link utilization, optimizing traffic performance metrics (such as delay, delay variation, or loss), optimizing resource utilization by invoking resource bonding, provide high-availability, etc.

When a path is partitioned into multiple path segments that are realized typically as overlay tunnels, LOOPS (Local Optimizations on Path Segments) aims to provide in-network recovery over segments to achieve better data delivery by making packet loss recovery faster. In an overlay network scenario, LOOPS can be performed over the existing, or purposely created, overlay tunnel based path segments. Figure 1 show an overall usage scenarios of LOOPS.

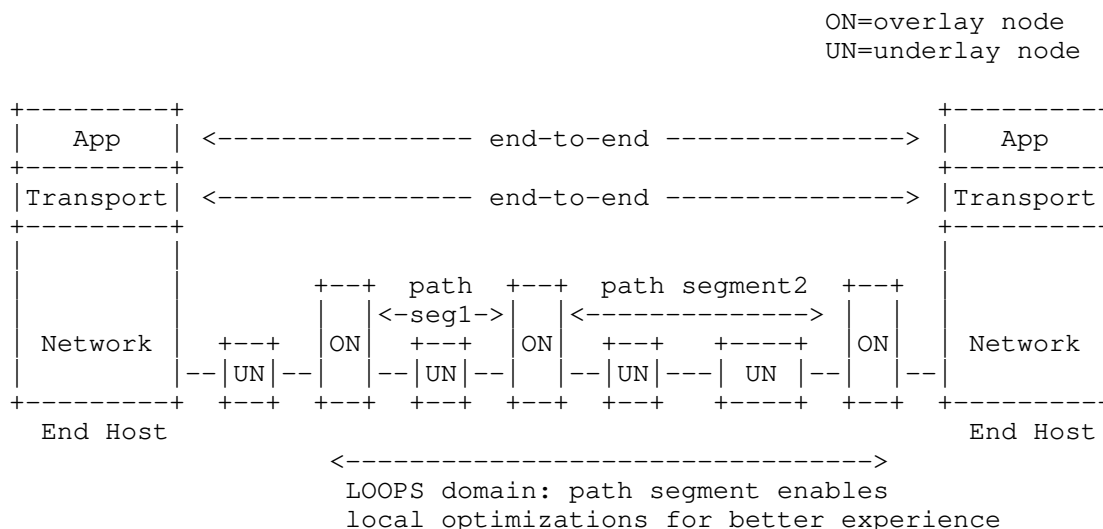


Figure 1: LOOPS Usage Scenario

## 1.2. Sketching a Work Direction: Rationale & Goals

This document sketches a proposal that is meant to experimentally investigate to what extent a network-assisted approach can contribute to increase the overall perceived quality of experience in specific situations (e.g., Sections 3.5 and 3.6 of [RFC8517]) without requiring access to internal transport primitives. The rationale beneath this approach is that some information (loss detection and segment characteristics, etc.) can be used to trigger local in-network recovery actions which have a faster effect while not impacting the end-to-end congestion control loop.

To that aim, the work is structured into two (2) phased stages:

- o Stage 1: Network-assisted optimization. This one assumes that optimizations can be implemented at the network without requiring defining new interaction with the endpoint. Existing tools such as ECN will be used. Loss signal would be converted to CE (congestion experienced) signal to interact with the end-to-end control loop.
- o Stage 2: Collaborative networking optimization. This one requires more interaction between the network and an endpoint to implement coordinated and more surgical network-assisted optimizations based on information/instructions shared by an endpoint or sharing locally-visible information with endpoint for better and faster recovery.

The document focuses on the first stage. Effort related to the second stage is out of scope of the initial planned work.

The proposed mechanism is not meant to be applied to all traffic, but only to a subset which is particularly benefits from, and has been selected for the network-assisted optimization service.

Which traffic is selected is deployment-specific and policy-based. For example, techniques for dynamic information about optimization function (e.g., SFC) may be leveraged to unambiguously identify the aggregate of traffic that is eligible to the service. Such identification may be triggered by subscription actions made by customers or be provided by a network provider (e.g., specific applications, during specific events such as during severe DDoS attack or flash crowds events).

Likewise, whether the optimization function is permanently instantiated or on-demand is deployment-specific.

This document does not intend to provide a comprehensive list of target deployment cases. Sample scenarios are described to illustrate some LOOPS potentials. Similar issues and optimizations may be helpful in other deployments such as enhancing the reliability of data transfer when a fleet of drones are used for specific missions (e.g., site inspection, live streaming, and emergency service). Captured data should be reliably transmitted via paths involving radio connections.

It is not required that all segments are LOOPS-aware to benefit from LOOPS advantages.

Section 3 presents the issues and opportunities found in some multiple path segments scenarios. Section 3 describes the impact of packet loss for different traffic. Section 5 describes the LOOPS desired features and their impact on existing network technologies. Section 6 shows the analysis on local retransmission and end-to-end retransmission. Section 7 summarizes LOOPS key elements.

## 2. Terminology

This document makes use of the following terms:

**LOOPS:** Local Optimizations on Path Segments. LOOPS includes the local in-network (i.e., non end-to-end) recovery functions and other supporting features such as local measurement, loss detection, and congestion feedback.

**LOOPS Node:** A node supporting LOOPS functions.

Overlay Node (ON): A node having overlay functions (e.g., overlay protocol encapsulation/decapsulation, header modification, TLV inspection) and LOOPS functions in the LOOPS overlay network usage scenario.

Overlay Tunnel: A tunnel with designated ingress and egress nodes using some network overlay protocol as encapsulation, optionally with a specific traffic type.

Path segment: A LOOPS enabled tunnel-based network subpath. It is used interchangeably with overlay segment in this document when the context wants to emphasize on its overlay encapsulated nature. It is also called segment for simplicity in this document.

Overlay segment: Refers to path segment.

Underlay Node (UN): A node not participating in the overlay network.

### 3. Usage Scenarios

#### 3.1. Cloud-Internet Overlay Network

CSPs (Cloud Service Providers) are connecting their data centers using the Internet or via self-constructed networks/links. This expands the traditional Internet's infrastructure and, together with the original ISP's infrastructure, forms the Internet underlay.

Automation techniques and NFV (Network Function Virtualization) make it easier to dynamically provision a new virtual node/function as a workload in a cloud for CPU/storage intensive functions. Virtual nodes can be in form of virtual machines or containers hosting the workloads sharing a physical node's infrastructure. With the aid of various mechanisms such as kernel bypassing and Virtual IO, forwarding based on virtual nodes is becoming more and more effective. The interconnection among the purposely positioned virtual nodes and/or the existing nodes with virtualization functions potentially form an overlay infrastructure. It is called the Cloud-Internet Overlay Network (CION) in this document for short.

This architecture scenario makes use of overlay technologies to direct the traffic going through the specific overlay path in order to achieve better service delivery. It purposely creates or selects overlay nodes (ON) from providers. By continuously measuring the delay of path segments and use them as metrics for path selection, when the number of overlay nodes is sufficiently large, there is a high chance that a better path could be found [DOI\_10.1109\_ICDCS.2016.49] [DOI\_10.1145\_3038912.3052560]. [DOI\_10.1145\_3038912.3052560] further shows all cloud providers

experience random loss episodes and random loss accounts for more than 35% of total loss.

Figure 2 shows an example of an overlay path over large geographic distances. An overlay node (ON) is usually a virtual node, though it does not have to be. Three path segments, i.e., ON1-ON2, ON2-ON3, ON3-ON4 are shown. Each segment transmits packets using some form of network overlay protocol encapsulation. ON has the computing and memory resources that can be used for some functions like packet loss detection, network measurement and feedback, packet retransmission and FEC (Forward Error Correction) computation. ONs here are managed by a single administrator though they can be workloads created from different CSPs.

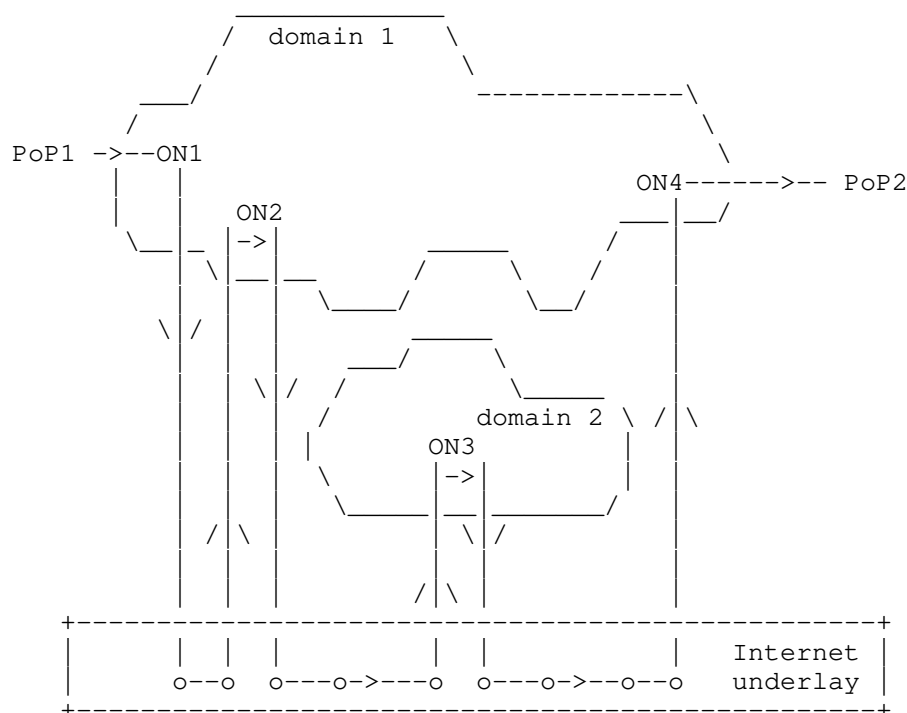


Figure 2: Cloud-Internet Overlay Network (CION)

We tested based on 37 overlay nodes from multiple cloud providers globally. Each pair of the overlay nodes are used as sender and receiver. When the traffic is not intentionally directed to go through any intermediate virtual nodes, we call the path followed by the traffic in the test the default path. When any of the virtual nodes is intentionally used as an intermediate node to forward the

traffic, the path that the traffic takes is called an overlay path. The preliminary experiments showed that the delay of a specifically selected overlay path has lower latency than the one of the default path in 69% of cases at 99% percentile and improvement is 17.5% at 99% percentile when we probe Ping packets every second for a week. The average number of hops for an overlay path is 3.02. More experimental information can be found in [DOI\_10.1109\_INFCOMW.2019.8845208].

Lower average delay does not necessarily mean less or no packet loss. Different path segments have different packet loss rates. Loss rate is another major factor impacting the user experience, especially for the short-lived or transactional flows. From some customer requirements, the target loss rate is set in the test to be less than 1% at 99% percentile and 99.9% percentile, respectively. The loss was measured between any two overlay nodes, i.e., any potential path segment. Two thousand Ping packets were sent every 20 seconds between two overlay nodes for 55 hours. This preliminary experiment showed that the packet loss rate satisfaction are only 44.27% and 29.51% at the 99% and 99.9% percentiles, respectively.

As CION naturally consists of multiple overlay segments, LOOPS can leverage this to perform local optimizations on a single hop between two overlay nodes. ("Local" here is a concept relative to end-to-end, it does not mean such optimization is limited to LAN networks.)

### 3.2. Satellite Communication

Traditionally, satellite communications deploy PEP (performance enhancing proxy [RFC3135]) nodes around the satellite link to enhance end-to-end performance. TCP splitting is a common approach employed by such PEPs, where the TCP connection is split into three: the segment before the satellite hop, the satellite section (uplink, downlink), and the segment behind the satellite hop. This requires heavy interactions with the end-to-end transport protocols, usually without the explicit consent of the end hosts. Unfortunately, this is indistinguishable from a man-in-the-middle attack on TCP. With end-to-end encryption moving under the transport (QUIC), this approach is no longer useful.

Geosynchronous Earth Orbit (GEO) satellites have a one-way delay (up to the satellite and back) on the order of 250 milliseconds. This does not include queueing, coding and other delays in the satellite ground equipment. The Round Trip Time for a TCP or QUIC connection going over a satellite hop in both directions, in the best case, will be on the order of 600 milliseconds. And, it may be considerably longer. RTTs on this order of magnitude have significant performance implications.

Packet loss recovery is an area where splitting the TCP connection into different parts helps. Packets lost on the terrestrial links can be recovered at terrestrial latencies. Packet loss on the satellite link can be recovered more quickly by an optimized satellite protocol between the PEPs and/or link layer FEC than they could be end to end. Again, encryption makes TCP splitting no longer applicable. Enhanced error recovery at the satellite link layer helps for the loss on the satellite link but doesn't help for the terrestrial links. Even when the terrestrial segments are short, any loss must be recovered across the satellite link delay. And, there are cases when a satellite ground station connects to the general Internet with a potentially larger terrestrial segment (e.g., to a correspondent host in another country). Faster recovery over such long terrestrial segments is desirable.

There are two high level classes of solutions for making encrypted transport traffic like QUIC work well over satellite:

- o Hooks in the transport protocol which can adapt to large BDPS where both the bandwidth and the latency are large. This would require end to end enhancement.
- o Capabilities (such as LOOPS) under the transport protocol to improve performance over specific segments of the path. In particular, separating the terrestrial from the satellite losses. Fixing the terrestrial loss quickly.

This document focuses on the latter.

### 3.3. Branch Office WAN Connection

Enterprises usually require network connections between the branch offices, or between branch office and cloud data center over geographic distances. With the increasing deployment of vCPE (virtual CPE), services hosted on the CPE are moved to the provider network from the customer site. Such vCPE approach enables some value added service to be provided such as WAN optimization and traffic steering.

Figure 3 shows a branch office access to public cloud via a selected PoP (point of presence) for service access or reaching another branch office via vPC (Virtual Private Cloud) interconnect. vCPE connects to the PoP which can be hundreds of kilometers away via Internet. From vCPE1 to vCPE2, it can consist of three segments, vCPE1-PoP1, PoP1-PoP2 and PoP2-vCPE2. Packet loss can happen on any of them. Segment based in-network recovery can be employed here to improve the WAN connection quality.



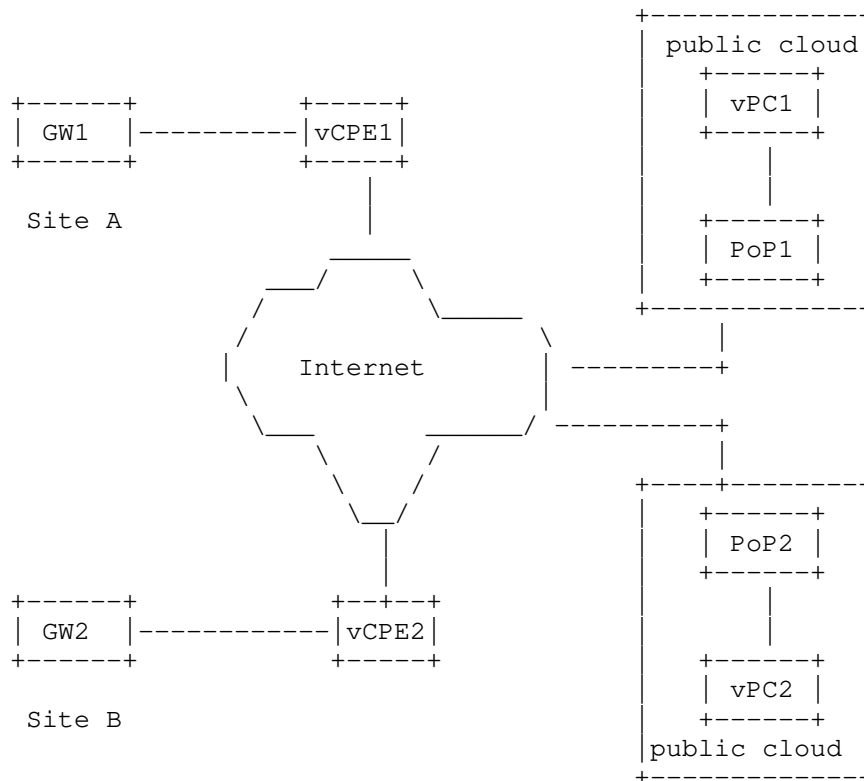


Figure 3: Enterprise Cloud Access

#### 4. Impact of Packet loss

##### 4.1. Tail Loss or Loss in Short Flows

When the lost segments are at the end of a transaction, TCP's fast retransmit algorithm does not work as there are no ACKs to trigger it. When a sender does not receive an ACK for a given segment within a certain amount of time called retransmission timeout (RTO), it re-sends the segment [RFC6298]. RTO can be as long as several seconds. Hence the recovery of lost segments triggered by RTO is lengthy. [I-D.dukkipati-tcpm-tcp-loss-probe] indicates that large RTOs make a significant contribution to the long tail on the latency statistics of short flows such as loading web pages.

The short-lived flows often complete in one or two RTTs. Even when the lost packet is not an exact tail, it can possibly add another RTT

because there may not be enough packets in flight to trigger the fast retransmit). In long-haul networks, it can result in extra time of tens or hundreds of milliseconds. For ant short lived or transactional flows, it affects the latency greatly.

An overlay segment transmits the aggregated flows from ON to ON. As short-lived flows are aggregated, the probability of tail loss over this specific overlay segment decreases compared to an individual flow. The overlay segment is much shorter than the end-to-end path, hence loss recovery over an overlay segment helps to obtain low latency.

#### 4.2. Packet Loss in Real Time Media Streams

The Real-time transport protocol (RTP) is widely used in interactive audio and video. Packet loss degrades the quality of the received media. When the latency tolerance of the application is sufficiently large, the RTP sender may use RTCP NACK feedback from the receiver [RFC4585] to trigger the retransmission of the lost packets before the playout time is reached at the receiver.

The end-to-end path over WAN can be hundreds of milliseconds, so the end-to-end feedback based retransmission may be not be very useful when applications can not tolerate one more RTT. Loss recovery over an overlay segment can then be used for the scenarios in which a shorter delayed retransmission can catch up with the playout time.

### 5. Features to be Considered for LOOPS

This section provides an overview of the LOOPS features. This section is not meant to document a detailed specification, but it is meant to highlight some design choices that may be followed during the solution design phase.

#### 5.1. Local Recovery

LOOPS (Local Optimizations on Path Segments) aims to provide in-network recovery over segments to achieve better data delivery by making packet loss recovery faster. This is viable because LOOPS nodes will be instantiated to partition the path into segments. At the same time, LOOPS does not replace the end-to-end loss recovery (if any). With the advent of automation and technologies like NFV and virtual IO, it is possible to dynamically instantiate functions to nodes. The enabling of LOOPS is expected to be dynamic. When to enable this function is out of scope. The operator or administrator can make the decision based on their historical experience or real-time monitoring.

There are two ways to recover packet, retransmission and Forward Error Correction (FEC). A document to specify the generic elements for loss detection, sequence number space, acknowledgment generation and state transition is available in [I-D.welzl-loops-gen-info].

## 5.2. Congestion Control Interaction

When a TCP-like transport layer protocol is used, local recovery in LOOPS has to interact with the upper layer transport congestion control. Classic TCP adjusts the congestion window when a loss is detected and then fast retransmit is invoked. LOOPS performs in-network recovery which may cause a loss event not being observed by the TCP sender. Then TCP sender may overshoot then.

To solve this issue, LOOPS needs to report the loss to end-to-end congestion control LOOPS. LOOPS can CE (Congestion Experienced) marks its recovered packets as the loss signal to end-to-end. Converting a packet loss signal to CE marking signal brings the benefits of reducing Head-of-Line blocking and probability of RTO expiry [RFC8087] without affecting TCP sender's loss based congestion control behaviour while enjoying the faster local recovery. ECN based indication is equivalent to a loss event at the TCP sender [RFC3168]. In this way, a requirement is set for applying LOOPS. Only ECT (ECN-Capable Transport) flows should be directed to an LOOPS enabled path segment.

## 5.3. Overlay Protocol Extensions

Some tunnel protocols such as VXLAN [RFC7348], GENEVE [I-D.ietf-nvo3-geneve], LISP [RFC6830] or CAPWAP [RFC5415] are employed in overlay network. They are used in various ways. A path can have single overlay tunnel as a sub-path or stitch multiple segments together, like VXLAN [RFC7348] or GENEVE [I-D.ietf-nvo3-geneve], or specify a sequence of intermediate nodes, as in SRv6 [RFC8754].

LOOPS does not look into the inner packet. LOOPS information is required to be embedded in the overlay protocol header. An example shown in Figure 4. The current protocol focus is GENEVE [I-D.ietf-nvo3-geneve]. The specific information is to be defined in separate documents.

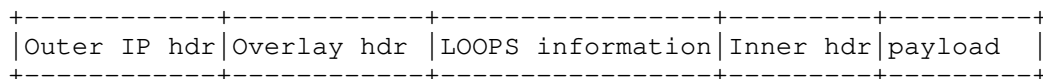


Figure 4: LOOPS Extension Header Example

## 6. Local in-network Recovery and End-to-end Retransmission

Most transport layer protocols have their own end-to-end retransmission to recover the lost packet. When LOOPS is in use, its local recovery can affect the end-to-end one. This section talks about such impacts.

There are two ways to perform local recovery, retransmission and FEC (Forward Error Correction). They are possibly used together in some cases. Such approaches between two overlay nodes recover the lost packet in relatively shorter distance and thus shorter latency. Therefore the local recovery is generally faster compared to end-to-end.

End-to-end retransmission is normally triggered by a NACK as in RTCP, multiple duplicate ACKs as in traditional TCP or time based detection as in RACK [I-D.ietf-tcpm-rack].

When FEC is used for local recovery, it may come with a buffer to make sure the recovered packets delivered are in order subsequently. Therefore the receiver side is unlikely to see the out-of-order packets and then send a NACK or multiple duplicate ACKs. The side effect to unnecessarily trigger end-to-end retransmit is minimum. When FEC is used in this way, if redundancy and block size are determined, extra latency required to recover lost packets is also bounded. Then RTT variation caused by it is predictable. In some extreme case like a large number of packet loss caused by persistent burst, FEC may not be able to recover it. Then end-to-end retransmit will work as a last resort. In summary, when FEC is used as local recovery, the impact on end-to-end retransmission is limited.

When local retransmission is used, it has the following impacts on the end-to-end retransmission.

For packet loss in RTP streaming, local retransmission can recover those packets which would not be retransmitted end-to-end otherwise due to long RTT. Therefore when the segment(s) being retransmitted on is a small portion of the whole end to end path, the retransmission will have a significant effect of improving the quality at receiver.

When the sender also re-transmits the packet based on a NACK received, the receiver may receive the duplicated retransmitted packets.

For packet loss in TCP flows, TCP RENO and CUBIC use duplicate ACKs as a loss signal to trigger the fast retransmit. Though we are not

standardize the buffering feature of a LOOPS egress, an introductory analysis is given as follows.

- o The egress overlay node can buffer the out-of-order packets for a while, giving a limited time for a packet being retransmitted somewhere in the overlay path to reach it. The retransmitted packet and the buffered packets caused by it may increase the RTT variation at the sender. When the retransmitted latency is a small portion of RTT or the loss is rare, such RTT variation will be smoothed without much impact.

The buffer management is nontrivial in this case. It has to be determined how many out-of-order packets can be buffered at the egress overlay node before it gives up waiting for a successful local retransmission. In some extreme case the lost packet is not recovered successfully locally, the sender may invoke end-to-end fast retransmit slower than it would be in classic TCP.

- o If LOOPS network does not buffer the out-of-order packets caused by packet loss, TCP sender which uses a time based loss detection like RACK [I-D.ietf-tcpm-rack] will perform well here. It uses the notion of time to replace the conventional DUPACK threshold approach to detect losses. Hence it prevents the TCP sender from invoking fast retransmit too early. Local retransmission will not interfere the sender's retransmission generally in this case. If time based loss detection is not supported at the sender, end to end retransmission may be invoked as usual. It consumes extra bandwidth Because the lost packets (i.e. recovered packet) is normally a very small percentage of the total packets. Then extra bandwidth cost is not significant.

## 7. Summary

LOOPS will extend the existing overlay protocols in data plane, potential starting from GENEVE [I-D.ietf-nvo3-geneve] which has good extensibility. Path or segment selection can be feature provided by the overlay protocols via SDN techniques [RFC7149] or other approaches and is not a part of LOOPS. LOOPS is a set of functions to be implemented on Overlay Nodes as a tunnel transport with best effort reliability. LOOPS targets the following features.

1. Local recovery: Local recovery: Retransmission, FEC, or combination thereof can be used as local recovery method. Such recovery mechanism is in-network. It is performed by two network nodes with computing and memory resources.

2. Local measurement: Ingress/Egress overlay nodes measure the local segment RTT, loss and/or throughput to immediately get the overlay segment status for loss detection.
3. Interact with end-to-end congestion control: Convert a packet loss signal to an ECN-marking signal to notify the end host sender.

## 8. Security Considerations

LOOPS does not require access to the traffic payload in clear, so encrypted payload does not affect functionality of LOOPS.

The use of LOOPS introduces some issues which impact security. ON with LOOPS function represents a point in the network where the traffic can be potentially manipulated and intercepted by malicious nodes. Means to ensure that only legitimate nodes are involved should be considered.

Denial of service attack can be launched from an ON. A rogue ON might be able to spoof packets as if it come from a legitimate ON. It may also modify the ECN CE marking in packets to influence the sender's rate. In order to protected from such attacks, the overlay protocol itself should have some built-in security protection which is used by LOOPS. The operator should use some authentication mechanism to make sure ONs are valid and non-compromised.

## 9. IANA Considerations

No IANA action is required.

## 10. Acknowledgements

Thanks to etosat mailing list about the discussion about the SatCom and LOOPS use case.

## 11. Informative References

- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC3135] Border, J., Kojo, M., Griner, J., Montenegro, G., and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations", RFC 3135, DOI 10.17487/RFC3135, June 2001, <<https://www.rfc-editor.org/info/rfc3135>>.

- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <<https://www.rfc-editor.org/info/rfc3550>>.
- [RFC4585] Ott, J., Wenger, S., Sato, N., Burmeister, C., and J. Rey, "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, DOI 10.17487/RFC4585, July 2006, <<https://www.rfc-editor.org/info/rfc4585>>.
- [RFC4588] Rey, J., Leon, D., Miyazaki, A., Varsa, V., and R. Hakenberg, "RTP Retransmission Payload Format", RFC 4588, DOI 10.17487/RFC4588, July 2006, <<https://www.rfc-editor.org/info/rfc4588>>.
- [RFC5415] Calhoun, P., Ed., Montemurro, M., Ed., and D. Stanley, Ed., "Control And Provisioning of Wireless Access Points (CAPWAP) Protocol Specification", RFC 5415, DOI 10.17487/RFC5415, March 2009, <<https://www.rfc-editor.org/info/rfc5415>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/info/rfc5681>>.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <<https://www.rfc-editor.org/info/rfc6298>>.
- [RFC6830] Farinacci, D., Fuller, V., Meyer, D., and D. Lewis, "The Locator/ID Separation Protocol (LISP)", RFC 6830, DOI 10.17487/RFC6830, January 2013, <<https://www.rfc-editor.org/info/rfc6830>>.
- [RFC7149] Boucadair, M. and C. Jacquenet, "Software-Defined Networking: A Perspective from within a Service Provider Environment", RFC 7149, DOI 10.17487/RFC7149, March 2014, <<https://www.rfc-editor.org/info/rfc7149>>.

- [RFC7348] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", RFC 7348, DOI 10.17487/RFC7348, August 2014, <<https://www.rfc-editor.org/info/rfc7348>>.
- [RFC8087] Fairhurst, G. and M. Welzl, "The Benefits of Using Explicit Congestion Notification (ECN)", RFC 8087, DOI 10.17487/RFC8087, March 2017, <<https://www.rfc-editor.org/info/rfc8087>>.
- [RFC8517] Dolson, D., Ed., Snellman, J., Boucadair, M., Ed., and C. Jacquenet, "An Inventory of Transport-Centric Functions Provided by Middleboxes: An Operator Perspective", RFC 8517, DOI 10.17487/RFC8517, February 2019, <<https://www.rfc-editor.org/info/rfc8517>>.
- [RFC8754] Filsfils, C., Ed., Dukes, D., Ed., Previdi, S., Leddy, J., Matsushima, S., and D. Voyer, "IPv6 Segment Routing Header (SRH)", RFC 8754, DOI 10.17487/RFC8754, March 2020, <<https://www.rfc-editor.org/info/rfc8754>>.
- [I-D.dukkipati-tcpm-tcp-loss-probe]  
Dukkipati, N., Cardwell, N., Cheng, Y., and M. Mathis, "Tail Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses", draft-dukkipati-tcpm-tcp-loss-probe-01 (work in progress), February 2013.
- [I-D.ietf-nvo3-geneve]  
Gross, J., Ganga, I., and T. Sridhar, "Geneve: Generic Network Virtualization Encapsulation", draft-ietf-nvo3-geneve-16 (work in progress), March 2020.
- [I-D.ietf-tcpm-rack]  
Cheng, Y., Cardwell, N., Dukkipati, N., and P. Jha, "RACK: a time-based fast loss detection algorithm for TCP", draft-ietf-tcpm-rack-08 (work in progress), March 2020.
- [I-D.welzl-loops-gen-info]  
Welzl, M. and C. Bormann, "LOOPS Generic Information Set", draft-welzl-loops-gen-info-03 (work in progress), March 2020.



[DOI\_10.1109\_ICDCS.2016.49]

Cai, C., Le, F., Sun, X., Xie, G., Jamjoom, H., and R. Campbell, "CRONets: Cloud-Routed Overlay Networks", 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS), DOI 10.1109/icdcs.2016.49, June 2016.

[DOI\_10.1145\_3038912.3052560]

Haq, O., Raja, M., and F. Dogar, "Measuring and Improving the Reliability of Wide-Area Cloud Paths", Proceedings of the 26th International Conference on World Wide Web, DOI 10.1145/3038912.3052560, April 2017.

[DOI\_10.1109\_INFCOMW.2019.8845208]

Xu, Z., Ju, R., Gu, L., Wang, W., Li, J., Li, F., and L. Han, "Using Overlay Cloud Network to Accelerate Global Communications", IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), DOI 10.1109/infcomw.2019.8845208, April 2019.

#### Authors' Addresses

Yizhou Li  
Huawei Technologies

Email: liyizhou@huawei.com

Xingwang Zhou  
Huawei Technologies

Email: zhouxingwang@huawei.com

Mohamed Boucadair  
Orange

Email: mohamed.boucadair@orange.com

Jianglong Wang  
China Telecom

Email: wangjll.bri@chinatelecom.cn

Fengwei Qin  
China Mobile

Email: qinfengwei@chinamobile.com

Transport Working Group  
Internet-Draft  
Updates: 3168, 8311 (if approved)  
Intended status: Standards Track  
Expires: September 11, 2019

J. Morton  
Bufferbloat.net  
D. Taeht  
TekLibre  
March 10, 2019

The Some Congestion Experienced ECN Codepoint  
draft-morton-taht-tsvwg-sce-00

Abstract

This memo reclassifies ECT(1) to be an early notification of congestion on ECT(0) marked packets, which can be used by AQM algorithms and transports as an earlier signal of congestion than CE. It is a simple, transparent, and backward compatible upgrade to existing IETF-approved AQMs, RFC3168, and nearly all congestion control algorithms.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 11, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Terminology . . . . .	2
2. Introduction . . . . .	2
3. Background . . . . .	2
4. Some Congestion Experienced . . . . .	3
5. Examples of use . . . . .	5
5.1. Cubic . . . . .	5
5.2. TCP receiver side handling . . . . .	5
5.3. Other . . . . .	5
6. Related Work . . . . .	5
7. IANA Considerations . . . . .	5
8. Security Considerations . . . . .	6
9. Acknowledgements . . . . .	6
10. References . . . . .	6
10.1. Normative References . . . . .	6
10.2. Informative References . . . . .	6
Authors' Addresses . . . . .	7

## 1. Terminology

The keywords **MUST**, **MUST NOT**, **REQUIRED**, **SHALL**, **SHALL NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **MAY**, and **OPTIONAL**, when they appear in this document, are to be interpreted as described in [RFC2119].

## 2. Introduction

This memo reclassifies ECT(1) to be an early notification of congestion on ECT(0) marked packets, which can be used by AQM algorithms and transports as an earlier signal of congestion than CE ("Congestion Experienced").

This memo limits its scope to the redefinition of the ECT(1) codepoint as SCE, "Some Congestion Experienced", with a few brief illustrations of how it may be used.

## 3. Background

[RFC3168] defines the lower two bits of the (former) TOS byte in the IPv4/6 header as the ECN field. This may take four values: Not-ECT, ECT(0), ECT(1) or CE.

Binary Keyword References

-----

00	Not-ECT (Not ECN-Capable Transport)	[RFC 3168]
01	ECT(1) (ECN-Capable Transport(1))	[RFC 3168]
10	ECT(0) (ECN-Capable Transport(0))	[RFC 3168]
11	CE (Congestion Experienced)	[RFC 3168]

Research has shown that the ECT(1) codepoint goes essentially unused, with the "Nonce Sum" extension to ECN having not been implemented in practice and thus subsequently obsoleted by [RFC8311] (section 3). Additionally, known [RFC3168] compliant senders do not emit ECT(1), and compliant middleboxes do not alter the field to ECT(1), while compliant receivers all interpret ECT(1) identically to ECT(0). These are useful properties which represent an opportunity for improvement.

Experience gained with 7 years of [RFC8290] deployment in the field suggests that it remains difficult to maintain the desired 100% link utilisation, whilst simultaneously strictly minimising induced delay due to excess queue depth - irrespective of whether ECN is in use. This leads to a reluctance amongst hardware vendors to implement the most effective AQM schemes because their headline benchmarks are throughput-based.

The underlying cause is the very sharp "multiplicative decrease" reaction required of transport protocols to congestion signalling (whether that be packet loss or CE marks), which tends to leave the congestion window significantly smaller than the ideal BDP when triggered at only slightly above the ideal value. The availability of this sharp response is required to assure network stability (AIMD principle), but there is presently no standardised and backwards-compatible means of providing a less drastic signal.

#### 4. Some Congestion Experienced

As consensus has arisen that some form of ECN signaling should be an earlier signal than drop, this Internet Draft changes the meaning of ECT(1) to be SCE, meaning "Some Congestion Experienced". The above ECN-field codepoint table then becomes:

##### Binary Keyword References

---

00	Not-ECT (Not ECN-Capable Transport)	[@RFC3168]
01	SCE (Some Congestion Experienced)	[This Internet-draft]
10	ECT (ECN-Capable Transport)	[@RFC3168]
11	CE (Congestion Experienced)	[@RFC3168]

This permits middleboxes implementing AQM to signal incipient congestion, below the threshold required to justify setting CE, by converting some proportion of ECT codepoints to SCE ("SCE marking"). Existing [RFC3168] compliant receivers MUST transparently ignore this new signal, and both existing and SCE-aware middleboxes MAY convert SCE to CE in the same circumstances as for ECT, thus ensuring backwards compatibility with [RFC3168] ECN endpoints.

Permitted ECN codepoint packet transitions by middleboxes are:

Not-ECT	->	Not-ECT or DROP
ECT	->	ECT or SCE or CE or DROP
SCE	->	SCE or CE or DROP
CE	->	CE or DROP

In other words, for ECN-aware flows, the ECN marking of an individual packet MAY be increased by a middlebox to signal congestion, but MUST NOT be decreased, and packets SHALL NOT be altered to appear to be ECN-aware if they were not originally, nor vice versa. Note however that SCE is numerically less than ECT, but semantically greater, and the latter definition applies for this rule.

New SCE-aware receivers and transport protocols SHALL continue to apply the [RFC3168] interpretation of the CE codepoint, that is, to signal the sender to back off send rate to the same extent as if a packet loss were detected. This maintains compatibility with existing middleboxes, senders and receivers.

New SCE-aware receivers and transport protocols SHOULD interpret the SCE codepoint as an indication of mild congestion, and respond accordingly by applying send rates intermediate between those resulting from a continuous sequence of ECT codepoints, and those resulting from a CE codepoint. The ratio of ECT and SCE codepoints received indicates the relative severity of such congestion, such that 100% SCE is very close to the threshold of CE marking, 100% ECT indicates that the bottleneck link may not be fully utilised, and a 1:1 balance of ECT and SCE codepoints indicates that the present send rate is a good match to the bottleneck link.

Details of how to implement SCE awareness at the transport layer will be left to additional Internet Drafts yet to be submitted.

To maximise the benefit of SCE, middleboxes SHOULD produce SCE markings sooner than they produce CE markings, when the level of congestion increases.

## 5. Examples of use

### 5.1. Cubic

Consider a TCP transport implementing CUBIC congestion control. This presently exhibits exponential cwnd growth during slow-start, polynomial cwnd growth in steady-state, and multiplicative decrease upon detecting a single CE marking or packet loss in one RTT cycle.

With SCE awareness, it might exit slow-start upon detecting a single SCE marking, switch from polynomial to Reno-linear cwnd growth when the SCE:ECT ratio exceeds 1:2, halt cwnd growth entirely when it exceeds 1:1, and implement a Reno-linear decline when it exceeds 2:1, in addition to retaining the sharp 40% decrease on detecting CE.

In ideal circumstances, the above behaviour would result in the send rate stabilising at a level which produces between 50% and 66% SCE marking at some bottleneck on the path. The middlebox performing this marking can thus control the send rate smoothly to an ideal value, maximising throughput with minimum average queue length.

### 5.2. TCP receiver side handling

SCE can potentially be handled entirely by the receiver and be entirely independent of any of the dozens of [RFC3168] compliant congestion control algorithms, for example by manipulating the TCP receive window in a similar manner to the sender's congestion window.

Alternatively, some mechanism may be defined to feed back SCE signals to the sender explicitly. Details of this are left to future I-Ds.

### 5.3. Other

New transports under development such as QUIC SHOULD implement a multi-bit, sub-RTT, and finer grained signal back to the sender based on SCE.

## 6. Related Work

[RFC8087] [RFC7567] [RFC7928] [RFC8290] [RFC8289] [RFC8033] [RFC8034]

## 7. IANA Considerations

There are no IANA considerations.

## 8. Security Considerations

There are no security considerations.

## 9. Acknowledgements

Many thanks to John Gilmore, the members of the ecn-sane project and the cake@lists.bufferbloat.net mailing list, and the former IETF AQM working group.

## 10. References

### 10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <<https://www.rfc-editor.org/info/rfc8311>>.

### 10.2. Informative References

- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <<https://www.rfc-editor.org/info/rfc7567>>.
- [RFC7928] Kuhn, N., Ed., Natarajan, P., Ed., Khademi, N., Ed., and D. Ros, "Characterization Guidelines for Active Queue Management (AQM)", RFC 7928, DOI 10.17487/RFC7928, July 2016, <<https://www.rfc-editor.org/info/rfc7928>>.
- [RFC8033] Pan, R., Natarajan, P., Baker, F., and G. White, "Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem", RFC 8033, DOI 10.17487/RFC8033, February 2017, <<https://www.rfc-editor.org/info/rfc8033>>.



- [RFC8034] White, G. and R. Pan, "Active Queue Management (AQM) Based on Proportional Integral Controller Enhanced PIE) for Data-Over-Cable Service Interface Specifications (DOCSIS) Cable Modems", RFC 8034, DOI 10.17487/RFC8034, February 2017, <<https://www.rfc-editor.org/info/rfc8034>>.
- [RFC8087] Fairhurst, G. and M. Welzl, "The Benefits of Using Explicit Congestion Notification (ECN)", RFC 8087, DOI 10.17487/RFC8087, March 2017, <<https://www.rfc-editor.org/info/rfc8087>>.
- [RFC8289] Nichols, K., Jacobson, V., McGregor, A., Ed., and J. Iyengar, Ed., "Controlled Delay Active Queue Management", RFC 8289, DOI 10.17487/RFC8289, January 2018, <<https://www.rfc-editor.org/info/rfc8289>>.
- [RFC8290] Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm", RFC 8290, DOI 10.17487/RFC8290, January 2018, <<https://www.rfc-editor.org/info/rfc8290>>.

## Authors' Addresses

Jonathan Morton  
Bufferbloat.net  
Koekkoenranta 21  
PITKAeJAeRVI 31520  
FINLAND

Phone: +358 44 927 2377  
Email: [chromatix99@gmail.com](mailto:chromatix99@gmail.com)

David M. Taeht  
TekLibre  
20600 Aldercroft Heights Rd  
Los Gatos, Ca 95033  
USA

Phone: +18312059740  
Email: [dave@taht.net](mailto:dave@taht.net)

Internet Area Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: May 6, 2021

V. Olteanu  
D. Niculescu  
University Politehnica of Bucharest  
November 02, 2020

SOCKS Protocol Version 6  
draft-olteanu-intarea-socks-6-11

## Abstract

The SOCKS protocol is used primarily to proxy TCP connections to arbitrary destinations via the use of a proxy server. Under the latest version of the protocol (version 5), it takes 2 RTTs (or 3, if authentication is used) before data can flow between the client and the server.

This memo proposes SOCKS version 6, which reduces the number of RTTs used, takes full advantage of TCP Fast Open, and adds support for 0-RTT authentication.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 6, 2021.

## Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Revision log . . . . .	4
2. Requirements language . . . . .	11
3. Mode of operation . . . . .	11
4. Requests . . . . .	12
5. Version Mismatch Replies . . . . .	14
6. Authentication Replies . . . . .	14
7. Operation Replies . . . . .	15
7.1. Handling CONNECT . . . . .	17
7.2. Handling BIND . . . . .	17
7.3. Handling UDP ASSOCIATE . . . . .	17
7.3.1. Proxying UDP servers . . . . .	21
7.3.2. Proxying multicast traffic . . . . .	21
7.3.3. Reporting ICMP Errors . . . . .	21
8. SOCKS Options . . . . .	22
8.1. Stack options . . . . .	23
8.1.1. IP TOS options . . . . .	24
8.1.2. Happy Eyeballs options . . . . .	24
8.1.3. TTL options . . . . .	25
8.1.4. No Fragmentation options . . . . .	26
8.1.5. TFO options . . . . .	26
8.1.6. Multipath options . . . . .	27
8.1.7. Listen Backlog options . . . . .	27
8.1.8. UDP Error options . . . . .	28
8.1.9. Port Parity options . . . . .	29
8.2. Authentication Method options . . . . .	30
8.3. Authentication Data options . . . . .	32
8.4. Session options . . . . .	32
8.4.1. Session initiation . . . . .	33
8.4.2. Further SOCKS Requests . . . . .	34
8.4.3. Tearing down the session . . . . .	34
8.5. Idempotence options . . . . .	35
8.5.1. Requesting a token window . . . . .	35
8.5.2. Spending a token . . . . .	36
8.5.3. Shifting windows . . . . .	38
8.5.4. Out-of-order Window Advertisements . . . . .	38
9. Username/Password Authentication . . . . .	38
10. TCP Fast Open on the Client-Proxy Leg . . . . .	39
11. False Starts . . . . .	39
12. DNS provided by SOCKS . . . . .	40
13. Security Considerations . . . . .	40

13.1. Large requests . . . . .	40
13.2. Replay attacks . . . . .	41
13.3. Resource exhaustion . . . . .	41
14. Privacy Considerations . . . . .	41
15. IANA Considerations . . . . .	41
16. Acknowledgments . . . . .	42
17. References . . . . .	43
17.1. Normative References . . . . .	43
17.2. Informative References . . . . .	43
Authors' Addresses . . . . .	44

## 1. Introduction

Versions 4 and 5 [RFC1928] of the SOCKS protocol were developed two decades ago and are in widespread use for circuit level gateways or as circumvention tools, and enjoy wide support and usage from various software, such as web browsers, SSH clients, and proxifiers. However, their design needs an update in order to take advantage of the new features of transport protocols, such as TCP Fast Open [RFC7413], or to better assist newer transport protocols, such as MPTCP [RFC6824].

One of the main issues faced by SOCKS version 5 is that, when taking into account the TCP handshake, method negotiation, authentication, connection request and grant, it may take up to 5 RTTs for a data exchange to take place at the application layer. This is especially costly in networks with a large delay at the access layer, such as 3G, 4G, or satellite.

The desire to reduce the number of RTTs manifests itself in the design of newer security protocols. TLS version 1.3 [RFC8446] defines a zero round trip (0-RTT) handshake mode for connections if the client and server had previously communicated.

TCP Fast Open [RFC7413] is a TCP option that allows TCP to send data in the SYN and receive a response in the first ACK, and aims at obtaining a data response in one RTT. The SOCKS protocol needs to concern itself with at least two TFO deployment scenarios: First, when TFO is available end-to-end (at the client, at the proxy, and at the server); second, when TFO is active between the client and the proxy, but not at the server.

This document describes the SOCKS protocol version 6. The key improvements over SOCKS version 5 are:

- o The client sends as much information upfront as possible, and does not wait for the authentication process to conclude before requesting the creation of a socket.

- o The connection request also mimics the semantics of TCP Fast Open [RFC7413]. As part of the connection request, the client can supply the potential payload for the initial SYN that is sent out to the server.
- o The protocol can be extended via options without breaking backward-compatibility.
- o The protocol can leverage the aforementioned options to support 0-RTT authentication schemes.

#### 1.1. Revision log

Typos and minor clarifications are not listed.

draft-11

- o Changed intended status to Standards Track
- o Renamed Vendor-specific option range to Experimental
- o Stack options:
  - \* Fixed some instances where an unsupported option was indistinguishable from a case where the proxy couldn't or wouldn't honor it (offenders: Happy Eyeballs, IP Fragmentation, UDP Error, Port Parity)
  - \* MPTCP: changed semantics w.r.t. TCP BIND: the absence of such an option SHOULD no longer lead the proxy to refuse MPTCP
  - \* Port Parity: relaxed restrictions in case the client supplies a specific port

draft-10

- o Removed untrusted sessions
- o IP DF
- o UDP relay:
  - \* Support ICMPv6 Too Big
  - \* Shifted some fields in the error messages
  - \* RTP support

draft-09

- o Revamped UDP relay
  - \* Support for ICMP errors: host/net unreachable, TTL exceeded
  - \* Datagrams can be sent over TCP
  - \* Timeout for the receipt of the initial datagram
- o TTL stack option (intended use: traceroute)
- o Added the "Privacy Considerations" section
- o SOCKS-provided DNS: the proxy may provide a valid bind address and port

draft-08

- o Removed Address Resolution options
- o Happy Eyeballs options
- o DNS provided by SOCKS

draft-07

- o All fields are now aligned.
- o Eliminated version minors
- o Lots of changes to options
  - \* 2-byte option kinds
  - \* Flattened option kinds/types/reply codes; also renamed some options
  - \* Socket options
    - + Proxies MUST always answer them (Clients can probe for support)
    - + MPTCP Options: expanded functionality ("please do/don't do MPTCP on my behalf")
    - + MPTCP Scheduler options removed

- + Listen Backlog options: code changed to 0x03
  - \* Revamped Idempotence options
  - \* Auth data options limited to one per method
  - o Authentication Reply: all authentication-related information is now in the options
    - \* Authentication replies no longer have a field indicating the chosen auth. method
    - \* Method that must proceed (or whereby authentication succeeded) indicated in options
    - \* Username/password authentication: proxy now sends reply in option
  - o Removed requirements w.r.t. caching authentication methods by multihomed clients
  - o UDP: 8-byte association IDs
  - o Sessions
    - \* The proxy is now free to terminate ongoing connections along with the session.
    - \* The session-terminating request is not part of the session that it terminated.
  - o Address Resolution options
- draft-06
- o Session options
  - o Options now have a 2-byte length field.
  - o Stack options
    - \* Stack options can no longer contain duplicate information.
    - \* TFO: Better payload size semantics
    - \* TOS: Added missing code field.
    - \* MPTCP Scheduler options:

- + Removed support for round-robin
- + "Default" renamed to "Lowest latency first"
- \* Listen Backlog options: now tied to sessions, instead of an authenticated user
- o Idempotence options
  - \* Now used in the context of a session (no longer tied to an authenticated user)
  - \* Idempotence options have a different codepoint: 0x05. (Was 0x04.)
  - \* Clarified that implementations that support Idempotence Options must support all Idempotence Option Types.
  - \* Shifted Idempotence Option Types by 1. (Makes implementation easier.)
- o Shrunk vendor-specific option range to 32 (down from 64).
- o Removed reference to dropping initial data. (It could no longer be done as of -05.)
- o Initial data size capped at 16KB.
- o Application data is never encrypted by SOCKS 6. (It can still be encrypted by the TLS layer under SOCKS.)
- o Messages now carry the total length of the options, rather than the number of options. Limited options length to 16KB.
- o Security Considerations
  - \* Updated the section to reflect the smaller maximum message size.
  - \* Added a subsection on resource exhaustion.

draft-05

- o Limited the "slow" authentication negotiations to one (and Authentication Replies to 2)
- o Revamped the handling of the first bytes in the application data stream



- \* False starts are now recommended. (Added the "False Start" section.)
  - \* Initial data is only available to clients willing to do "slow" authentication. Moved the "Initial data size" field from Requests to Authentication Method options.
  - \* Initial data size capped at  $2^{13}$ . Initial data can no longer be dropped by the proxy.
  - \* The TFO option can hint at the desired SYN payload size.
  - o Request: clarified the meaning of the Address and Port fields.
  - o Better reverse TCP proxy support: optional listen backlog for TCP BIND
  - o TFO options can no longer be placed inside Operation Replies.
  - o IP TOS stack option
  - o Suggested a range for vendor-specific options.
  - o Revamped UDP functionality
    - \* Now using fixed UDP ports
    - \* DTLS support
  - o Stack options: renamed Proxy-Server leg to Proxy-Remote leg
- draft-04
- o Moved Token Expenditure Replies to the Authentication Reply.
  - o Shifted the Initial Data Size field in the Request, in order to make it easier to parse.

draft-03

- o Shifted some fields in the Operation Reply to make it easier to parse.
- o Added connection attempt timeout response code to Operation Replies.
- o Proxies send an additional Authentication Reply after the authentication phase. (Useful for token window advertisements.)

- o Renamed the section "Connection Requests" to "Requests"
- o Clarified the fact that proxies don't need to support any command in particular.
- o Added the section "TCP Fast Open on the Client-Proxy Leg"
- o Options:
  - \* Added constants for option kinds
  - \* Salt options removed, along with the relevant section from Security Considerations. (TLS 1.3 Makes AEAD mandatory.)
  - \* Limited Authentication Data options to one per method.
  - \* Relaxed proxy requirements with regard to handling multiple Authentication Data options. (When the client violates the above bullet point.)
  - \* Removed interdependence between Authentication Method and Authentication Data options.
  - \* Clients SHOULD omit advertising the "No authentication required" option. (Was MAY.)
  - \* Idempotence options:
    - + Token Window Advertisements are now part of successful Authentication Replies (so that the proxy-server RTT has no impact on their timeliness).
    - + Proxies can't advertise token windows of size 0.
    - + Tweaked token expenditure response codes.
    - + Support no longer mandatory on the proxy side.
  - \* Revamped Socket options
    - + Renamed Socket options to Stack options.
    - + Banned contradictory socket options.
    - + Added socket level for generic IP. Removed the "socket" socket level.
    - + Stack options no longer use option codes from `setsockopt()`.

- + Changed MPTCP Scheduler constants.

#### draft-02

- o Made support for Idempotence options mandatory for proxies.
- o Clarified what happens when proxies can not or will not issue tokens.
- o Limited token windows to  $2^{31} - 1$ .
- o Fixed definition of "less than" for tokens.
- o NOOP commands now trigger Operation Replies.
- o Renamed Authentication options to Authentication Data options.
- o Authentication Data options are no longer mandatory.
- o Authentication methods are now advertised via options.
- o Shifted some Request fields.
- o Option range for vendor-specific options.
- o Socket options.
- o Password authentication.
- o Salt options.

#### draft-01

- o Added this section.
- o Support for idempotent commands.
- o Removed version numbers from operation replies.
- o Request port number for SOCKS over TLS. Deprecate encryption/encapsulation within SOCKS.
- o Added Version Mismatch Replies.
- o Renamed the AUTH command to NOOP.
- o Shifted some fields to make requests and operation replies easier to parse.

## 2. Requirements language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

## 3. Mode of operation

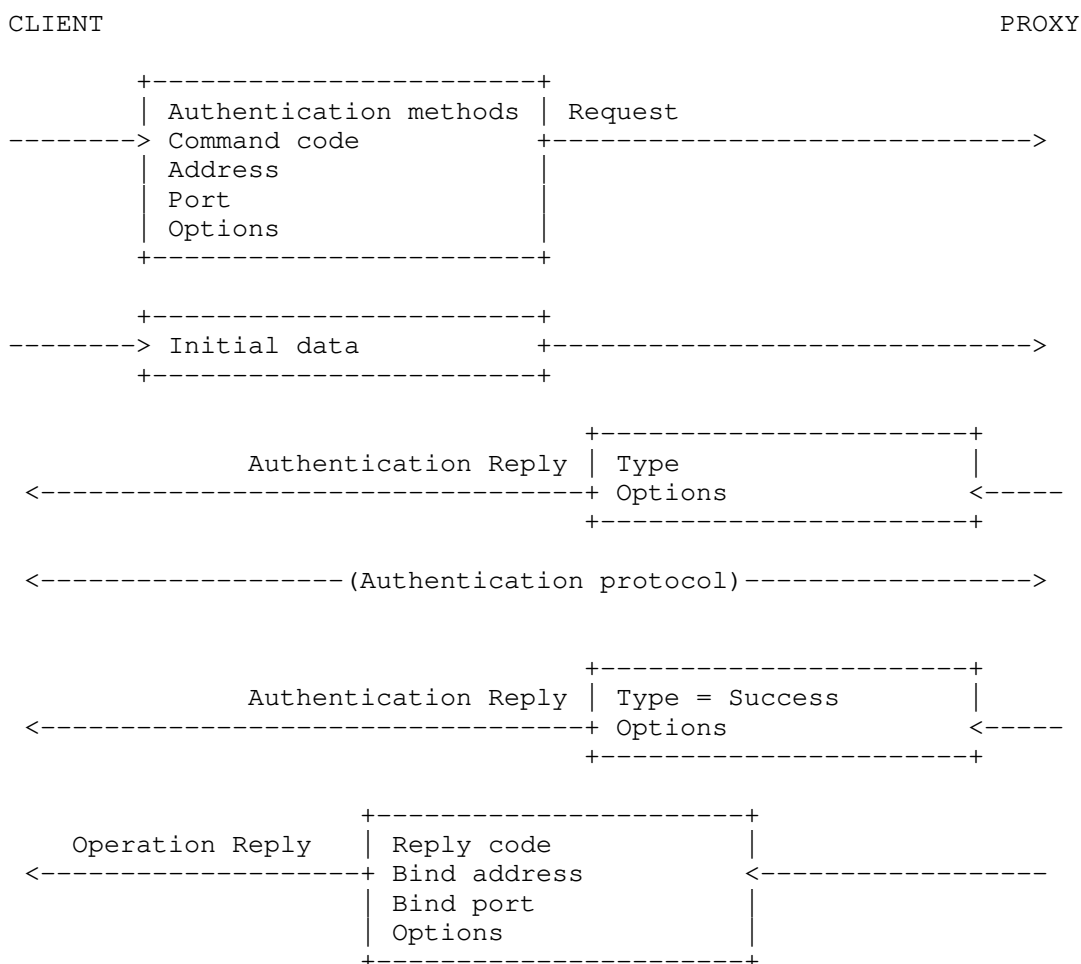


Figure 1: The SOCKS version 6 protocol message exchange

When a TCP-based client wishes to establish a connection to a server, it must open a TCP connection to the appropriate SOCKS port on the

SOCKS proxy. The client then enters a negotiation phase, by sending the request in Figure 1, that contains, in addition to fields present in SOCKS 5 [RFC1928], fields that facilitate low RTT usage and faster authentication negotiation.

Next, the server sends an authentication reply. If the request did not contain the necessary authentication information, the proxy indicates an authentication method that must proceed. This may trigger a longer authentication sequence that could include tokens for ulterior faster authentications. The part labeled "Authentication protocol" is specific to the authentication method employed and is not expected to be employed for every connection between a client and its proxy server. The authentication protocol typically takes up 1 RTT or more.

If the authentication is successful, an operation reply is generated by the proxy. It indicates whether the proxy was successful in creating the requested socket or not.

In the fast case, when authentication is properly set up, the proxy attempts to create the socket immediately after the receipt of the request, thus achieving an operational connection in one RTT (provided TFO functionality is available at the client, proxy, and server).

#### 4. Requests

The client starts by sending a request to the proxy.

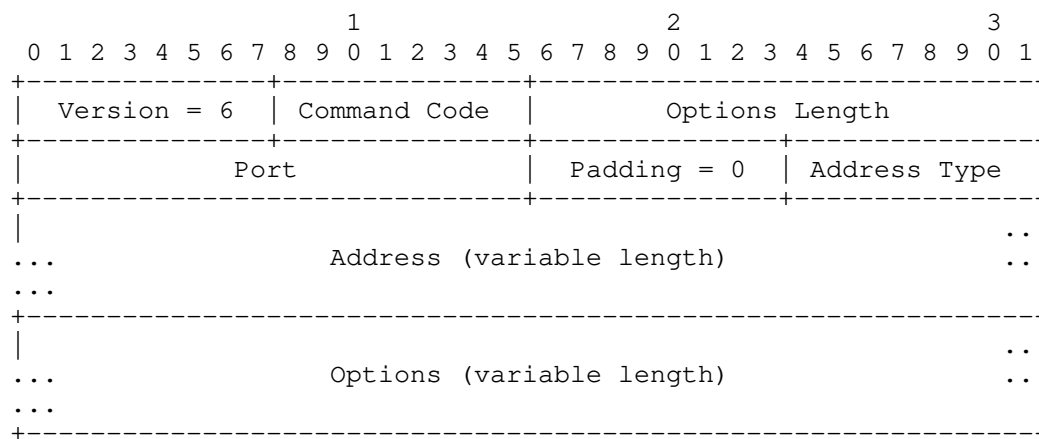


Figure 2: SOCKS 6 Request

- o Version: 6
- o Command Code:
  - \* 0x00 NOOP: does nothing.
  - \* 0x01 CONNECT: requests the establishment of a TCP connection. TFO MUST NOT be used unless explicitly requested.
  - \* 0x02 BIND: requests the establishment of a TCP port binding.
  - \* 0x03 UDP ASSOCIATE: requests a UDP port association.
- o Address Type:
  - \* 0x01: IPv4
  - \* 0x03: Domain Name
  - \* 0x04: IPv6
- o Address: this field's format depends on the address type:
  - \* IPv4: a 4-byte IPv4 address
  - \* Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated, but padded by NUL characters, if needed.
  - \* IPv6: a 16-byte IPv6 address
- o Port: the port in network byte order.
- o Padding: set to 0
- o Options Length: the total size of the SOCKS options that appear in the Options field. MUST NOT exceed 16KB.
- o Options: see Section 8.

The Address and Port fields have different meanings based on the Command Code:

- o NOOP: The fields have no meaning. The Address Type field MUST be either 0x01 (IPv4) or 0x04 (IPv6). The Address and Port fields MUST be 0.

- o CONNECT: The fields signify the address and port to which the client wishes to connect.
- o BIND, UDP ASSOCIATE: The fields indicate the desired bind address and port. If the client does not require a certain address, it can set the Address Type field to 0x01 (IPv4) or 0x04 (IPv6), and the Address field to 0. Likewise, if the client does not require a certain port, it can set the Port field to 0.

Clients can advertise their supported authentication methods by including an Authentication Method Advertisement option (see Section 8.2).

## 5. Version Mismatch Replies

Upon receipt of a request starting with a version number other than 6, the proxy sends the following response:

```

 0 1 2 3 4 5 6 7
+-----+
| Version = 6 |
+-----+
```

Figure 3: SOCKS 6 Version Mismatch Reply

- o Version: 6

A client **MUST** close the connection after receiving such a reply.

## 6. Authentication Replies

Upon receipt of a valid request, the proxy sends an Authentication Reply:

```

          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+
| Version = 6 | Type | Options Length |
+-----+-----+-----+
|                                     |
| Options (variable length)         |
|                                     |
+-----+-----+-----+
```

Figure 4: SOCKS 6 Authentication Reply

- o Version: 6
- o Type:
  - \* 0x00: authentication successful.
  - \* 0x01: authentication failed.
- o Options Length: the total size of the SOCKS options that appear in the Options field. MUST NOT exceed 16KB.
- o Options: see Section 8.

If the server signals that the authentication has failed and does not signal that any authentication negotiation can continue (via an Authentication Method Selection option), the client MUST close the connection.

The client and proxy begin a method-specific negotiation. During such negotiations, the proxy MAY supply information that allows the client to authenticate a future request using an Authentication Data option. Application data is not subject to any encryption negotiated during this phase. Descriptions of such negotiations are beyond the scope of this memo.

When the negotiation is complete (either successfully or unsuccessfully), the proxy sends a second Authentication Reply. The second Authentication Reply MUST NOT allow for further negotiations.

## 7. Operation Replies

After the authentication negotiations are complete, the proxy sends an Operation Reply:



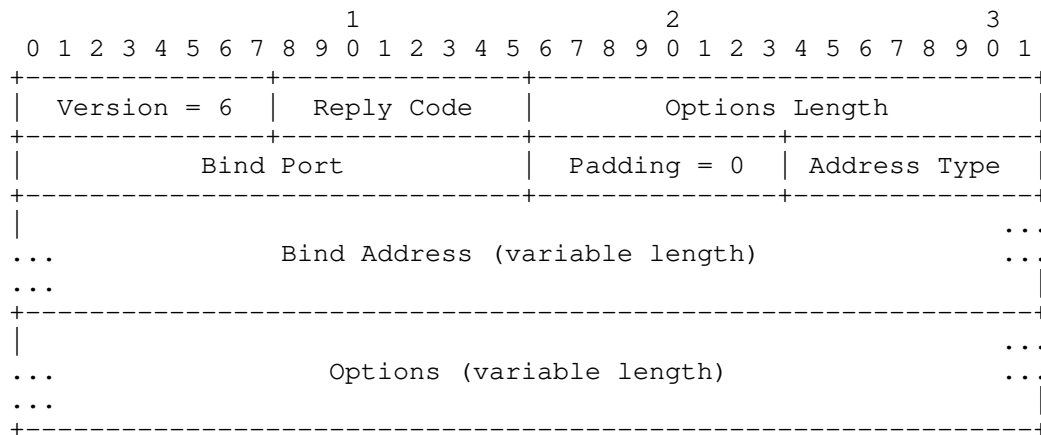


Figure 5: SOCKS 6 Operation Reply

- o Version: 6
- o Reply Code:
  - \* 0x00: Success
  - \* 0x01: General SOCKS server failure
  - \* 0x02: Connection not allowed by ruleset
  - \* 0x03: Network unreachable
  - \* 0x04: Host unreachable
  - \* 0x05: Connection refused
  - \* 0x06: TTL expired
  - \* 0x07: Command not supported
  - \* 0x08: Address type not supported
  - \* 0x09: Connection attempt timed out
- o Bind Port: the proxy bound port in network byte order.
- o Padding: set to 0
- o Address Type:

- \* 0x01: IPv4
- \* 0x03: Domain Name
- \* 0x04: IPv6
- o Bind Address: the proxy bound address in the following format:
  - \* IPv4: a 4-byte IPv4 address
  - \* Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated, but padded by NUL characters, if needed.
  - \* IPv6: a 16-byte IPv6 address
- o Options Length: the total size of the SOCKS options that appear in the Options field. MUST NOT exceed 16KB.
- o Options: see Section 8.

Proxy implementations MAY support any subset of the client commands listed in Section 4.

If the proxy returns a reply code other than "Success", the client MUST close the connection.

If the client issued an NOOP command, the client MUST close the connection after receiving the Operation Reply.

### 7.1. Handling CONNECT

In case the client has issued a CONNECT request, data can now pass.

### 7.2. Handling BIND

In case the client has issued a BIND request, it must wait for a second Operation reply from the proxy, which signifies that a host has connected to the bound port. The Bind Address and Bind Port fields contain the address and port of the connecting host. Afterwards, application data may pass.

### 7.3. Handling UDP ASSOCIATE

Proxies offering UDP functionality may be configured with a UDP port used for relaying UDP datagrams to and from the client, and/or a port used for relaying datagrams over DTLS.

Following a successful Operation Reply, the client and the proxy begin exchanging messages with the following header:

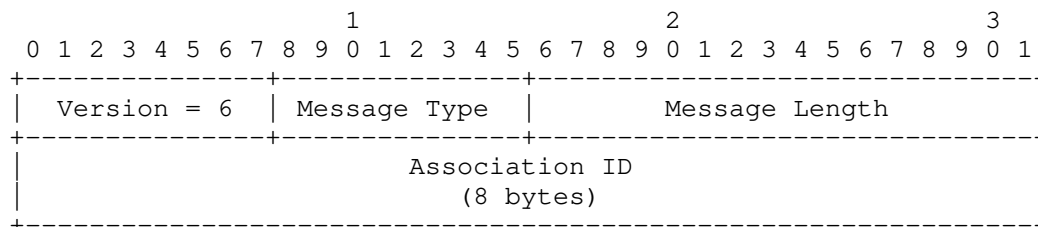


Figure 6: UDP Association Header

o Message Type:

- \* 0x01: Association Initialization
- \* 0x02: Association Confirmation
- \* 0x03: Datagram
- \* 0x04: Error

o Message Length: the total length of the message

o Association ID: the identifier of the UDP association

First, the proxy picks an Association ID sends a an Association Initialization message:

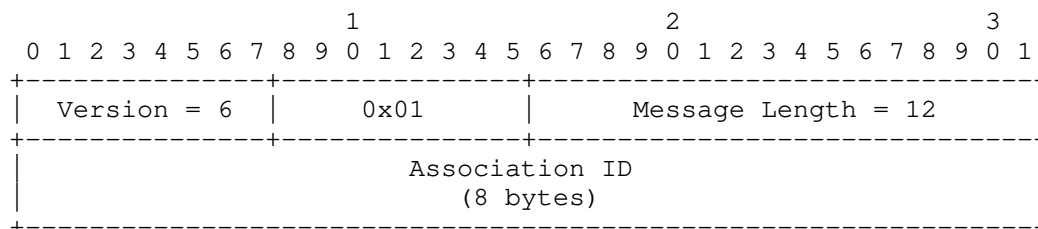


Figure 7: UDP Association Initialization

Proxy implementations SHOULD generate Association IDs randomly or pseudo-randomly.

Clients may start sending datagrams to the proxy either:

- o over the TCP connection,
- o in plaintext, using the proxy's configured UDP port(s), or
- o over an established DTLS session.

A client's datagrams are prefixed by a Datagram Header, indicating the remote host's address and port:

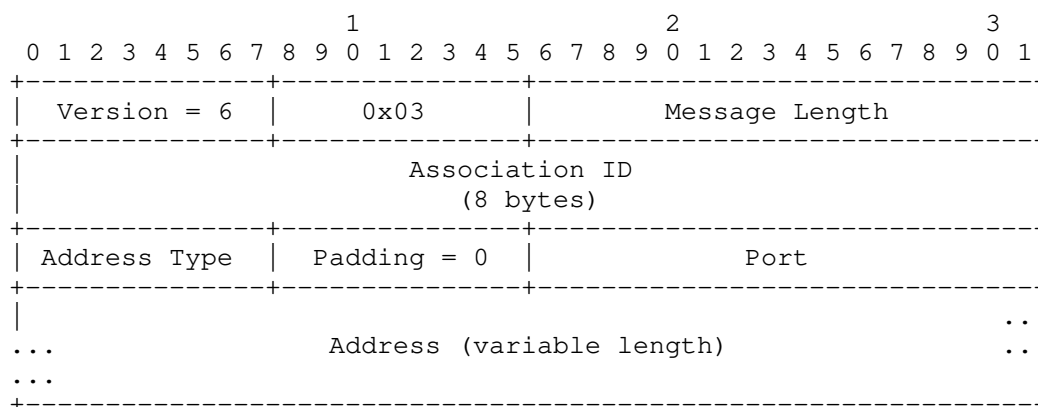


Figure 8: Datagram Header

- o Version: 0x06
- o Association ID: the identifier of the UDP association
- o Address Type:
  - \* 0x01: IPv4
  - \* 0x03: Domain Name
  - \* 0x04: IPv6
- o Address: this field's format depends on the address type:
  - \* IPv4: a 4-byte IPv4 address
  - \* Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated.
  - \* IPv6: a 16-byte IPv6 address

- o Port: the port in network byte order.

Datagrams sent over UDP MAY be padded with arbitrary data (i. e., the Message Length MAY be smaller than the actual UDP/DTLS payload). Client and proxy implementations MUST ignore the padding. If the Message Length is larger than the size of the UDP or DTLS payload, the message MUST be silently ignored.

Following the receipt of the first datagram from the client, the proxy makes a one-way mapping between the Association ID and:

- o the TCP connection, if it was received over TCP, or
- o the 5-tuple of the UDP conversation, if the datagram was received over plain UDP, or
- o the DTLS connection, if the datagram was received over DTLS. The DTLS connection is identified either by its 5-tuple, or some other mechanism, like [I-D.ietf-tls-dtls-connection-id].

The proxy SHOULD close the TCP connection if the initial datagram is not received after a timeout.

Further datagrams carrying the same Association ID, but not matching the established mapping, are silently dropped.

The proxy then sends an UDP Association Confirmation message over the TCP connection with the client:

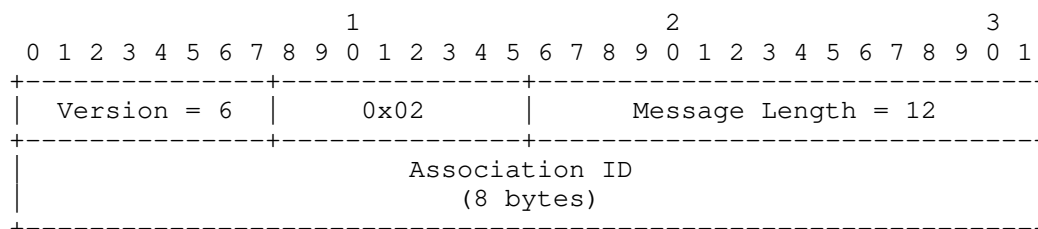


Figure 9: UDP Association Confirmation

Following the confirmation message, UDP packets bound for the proxy's bind address and port are relayed to the client, also prefixed by a Datagram Header.

The UDP association remains active for as long as the TCP connection between the client and the proxy is kept open.

### 7.3.1. Proxying UDP servers

Under some circumstances (e.g. when hosting a server), the SOCKS client expects the remote host to send UDP datagrams first. As such, the SOCKS client must trigger a UDP Association Confirmation without having the proxy relay any datagrams on its behalf.

To that end, it sends an empty datagram prefixed by a Datagram Header with an IP address and port consisting of zeroes. If it is using UDP, the client **SHOULD** resend the empty datagram if an UDP Association Confirmation is not received after a timeout.

### 7.3.2. Proxying multicast traffic

The use of multicast addresses is permitted for UDP traffic only.

### 7.3.3. Reporting ICMP Errors

If a client has opted in (see Section 8.1.8), the proxy **MAY** relay information contained in some ICMP Error packets. The message format is as follows:

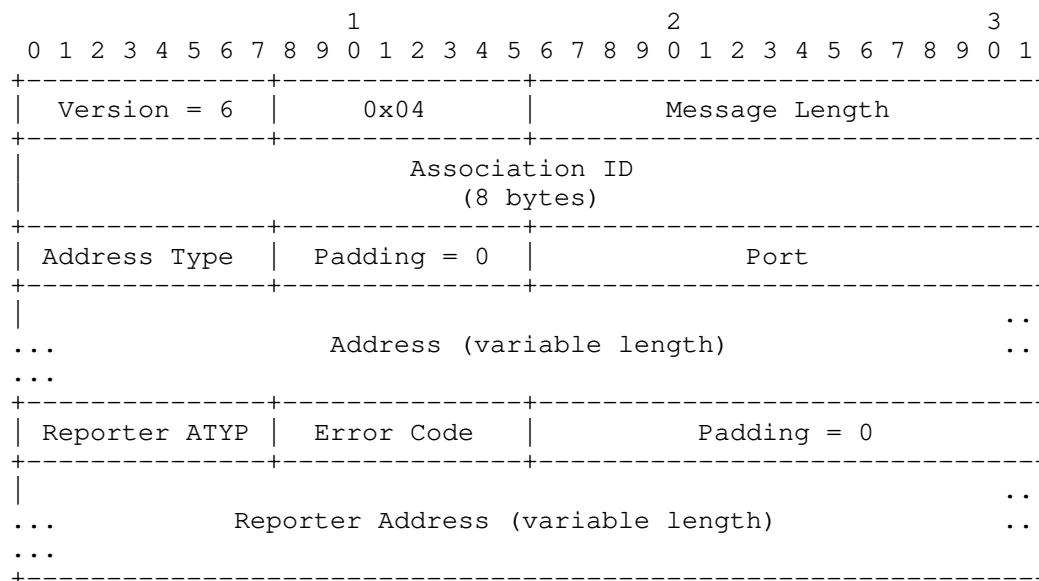


Figure 10: Datagram Error Message

- o Address: The destination address of the IP header contained in the ICMP payload

- o Address Type: Either 0x01 (IPv4) or 0x04 (IPv6)
- o Port: The destination port of the UDP header contained in the ICMP payload
- o Reporter Address: The IP address of the host that issued the ICMP error
- o Reporter Address Type (ATYP): Either 0x01 (IPv4) or 0x04 (IPv6)
- o Error code:
  - \* 0x01: Network unreachable
  - \* 0x02: Host unreachable
  - \* 0x03: TTL expired
  - \* 0x04: Datagram too big (IPv6 only)

It is possible for ICMP Error packets to be spurious, and not be related to any UDP packet that was sent out. The proxy is not required to check the validity of ICMP Error packets before reporting them to the client.

Clients **MUST NOT** send Datagram Error messages to the proxy. Proxies **MUST NOT** send Error messages unless the clients have opted in.

## 8. SOCKS Options

SOCKS options have the following format:

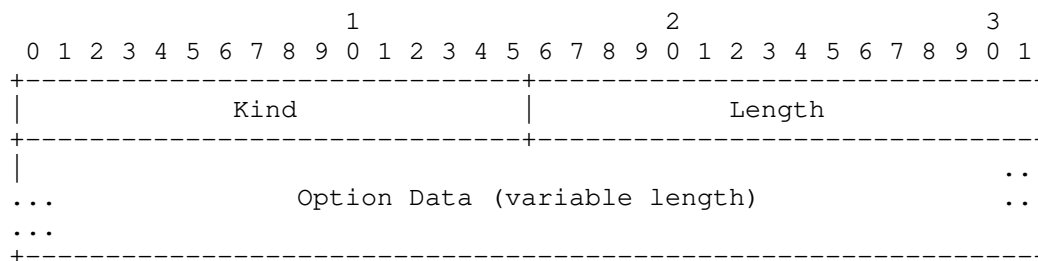


Figure 11: SOCKS 6 Option

- o Kind: Allocated by IANA. (See Section 15.)
- o Length: The total length of the option. **MUST** be a multiple of 4.

- o Option Data: The contents are specific to each option kind.

Unless otherwise noted, client and proxy implementations MAY omit supporting any of the options described in this document. Upon encountering an unsupported option, a SOCKS endpoint MUST silently ignore it.

### 8.1. Stack options

Stack options can be used by clients to alter the behavior of the protocols on top of which SOCKS is running, as well the protocols used by the proxy to communicate with the remote host (i.e. IP, TCP, UDP). A Stack option can affect either the proxy's protocol on the client-proxy leg or on the proxy-remote leg. Clients can only place Stack options inside SOCKS Requests.

Proxies MAY choose not to honor any Stack options sent by the client.

Proxies include Stack options in their Operation Replies to signal their behavior, and MUST do so for every supported Stack option sent by the client. Said options MAY also be unsolicited, i. e. the proxy MAY send them to signal behavior that was not explicitly requested by the client.

If a particular Stack option is unsupported, the proxy MUST silently ignore it.

In case of UDP ASSOCIATE, the stack options refer to the UDP traffic relayed by the proxy.

Stack options that are part of the same message MUST NOT contradict one another or contain duplicate information.

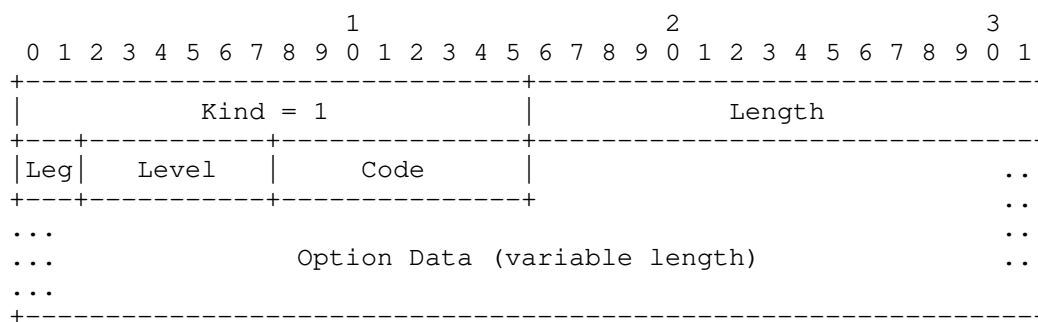


Figure 12: Stack Option



- o Leg:
  - \* 1: Client-Proxy Leg
  - \* 2: Proxy-Remote Leg
  - \* 3: Both Legs
- o Level:
  - \* 1: IP: options that apply to either IPv4 or IPv6
  - \* 2: IPv4
  - \* 3: IPv6
  - \* 4: TCP
  - \* 5: UDP
- o Code: Option code
- o Option Data: Option-specific data

8.1.1.1. IP TOS options

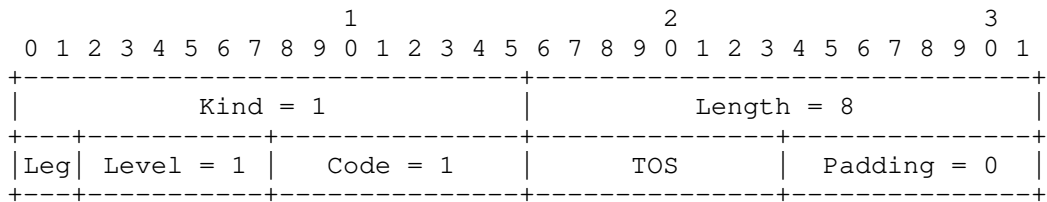


Figure 13: IP TOS Option

- o TOS: The IP TOS code
- The client can use IP TOS options to request that the proxy use a certain value for the IP TOS field. Likewise, the proxy can use IP TOS options to advertise the TOS values being used.
- 8.1.1.2. Happy Eyeballs options

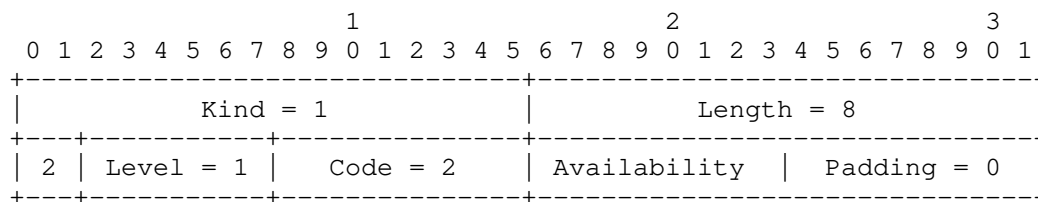


Figure 14: Happy Eyeballs Option

## o Availability:

- \* 0x01: Happy Eyeballs is not desired (client) or was not performed (proxy)
- \* 0x02: Happy Eyeballs is desired (client) or was attempted (proxy)

This memo provides enough features for clients to implement a mechanism analogous to Happy Eyeballs [RFC8305] over SOCKS. However, when the delay between the client and the proxy, or the proxy's vantage point, is high, doing so can become impractical or inefficient.

In such cases, the client can instruct the proxy to employ the Happy Eyeballs technique on its behalf when connecting to a remote host.

The client **MUST** supply a Domain Name as part of its Request. Otherwise, the proxy **MUST** silently ignore the option.

TODO: Figure out which knobs to include.

## 8.1.3. TTL options

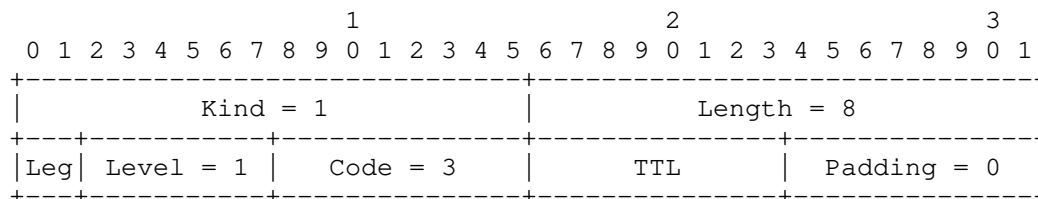


Figure 15: IP TTL Option

## o TTL: The IP TTL or Hop Limit

## 8.1.4. No Fragmentation options

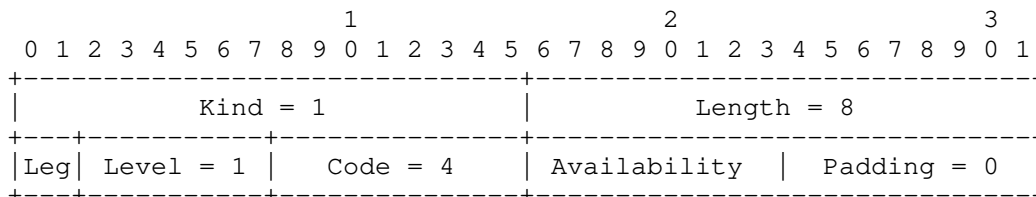


Figure 16: No Fragmentation Option

## o Availability:

- \* 0x01: IP fragmentation is allowed (client) or the lack thereof is not enforced (proxy)
- \* 0x02: IP fragmentation is not desired (client) or avoidance of fragmentation is enforced (proxy)

A No Fragmentation option can be used to instruct the proxy to avoid IP fragmentation. In the case of IPv4, this also entails setting the DF bit on outgoing packets.

## 8.1.5. TFO options

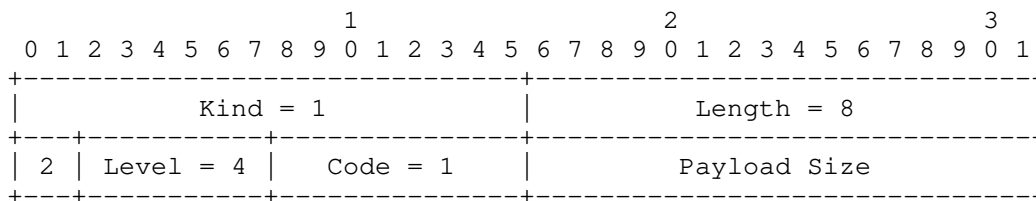


Figure 17: TFO Option

- o Payload Size: The desired payload size of the TFO SYN. Ignored in case of a BIND command.

If a SOCKS Request contains a TFO option, the proxy SHOULD attempt to use TFO in case of a CONNECT command, or accept TFO in case of a BIND command. Otherwise, the proxy MUST NOT attempt to use TFO in case of a CONNECT command, or accept TFO in case of a BIND command.

In case of a CONNECT command, the client can indicate the desired payload size of the SYN. If the field is 0, the proxy can use an

arbitrary payload size. If the field is non-zero, the proxy MUST NOT use a payload size larger than the one indicated. The proxy MAY use a smaller payload size than the one indicated.

#### 8.1.6. Multipath options

In case of a CONNECT or BIND command, the client can inform the proxy whether MPTCP is desired on the proxy-remote leg by sending a Multipath option.

Conversely, the proxy can use a Multipath option to convey the following information:

- o whether or not the connection uses MPTCP or not, when replying to a CONNECT command, or in the second Operation reply to a BIND command, or
- o whether an MPTCP connection will be accepted, when first replying to a BIND command.

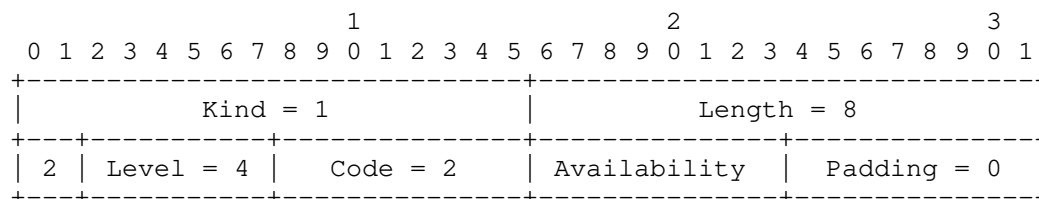


Figure 18: Multipath Option

- o Availability:
  - \* 0x01: MPTCP is not desired (client) or available (proxy)
  - \* 0x02: MPTCP is desired (client) or available (proxy)

In the absence of such an option, the proxy SHOULD NOT enable MPTCP for CONNECT commands.

#### 8.1.7. Listen Backlog options

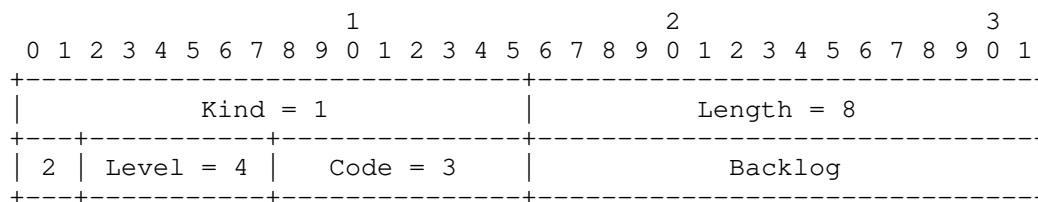


Figure 19: Listen Backlog Option

- o Backlog: The length of the listen backlog.

The default behavior of the BIND does not allow a client to simultaneously handle multiple connections to the same bind address. A client can alter BIND's behavior by adding a TCP Listen Backlog Option to a BIND Request, provided that the Request is part of a Session.

In response, the proxy sends a TCP Listen Backlog Option as part of the Operation Reply, with the Backlog field signaling the actual backlog used. The proxy SHOULD NOT use a backlog longer than requested.

Following the successful negotiation of a backlog, the proxy listens for incoming connections for as long as the initial connection stays open. The initial connection is not used to relay data between the client and a remote host.

To accept connections, the client issues further BIND Requests using the bind address and port supplied by the proxy in the initial Operation Reply. Said BIND requests must belong to the same Session as the original Request.

If no backlog is issued, the proxy signals a backlog length of 0, and BIND's behavior remains unaffected.

#### 8.1.8. UDP Error options

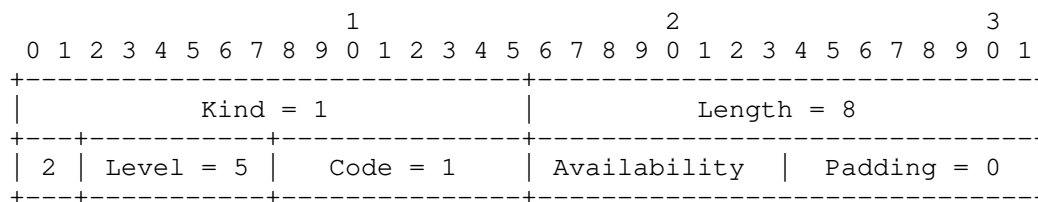


Figure 20: UDP Error Option

o Availability:

- \* 0x01: Error reporting is not desired (client) or will not be performed (proxy)
- \* 0x02: Error reporting is desired (client) or will be performed (proxy)

Clients can use this option to turn on error reporting for a particular UDP association. See Section 7.3.3.

#### 8.1.9. Port Parity options

The RTP specification [RFC3550] recommends running the protocol on consecutive UDP ports, where the even port is the lower of the two.

SOCKS clients can specify the desired port parity when issuing a UDP ASSOCIATE command, and request that the port's counterpart be reserved.

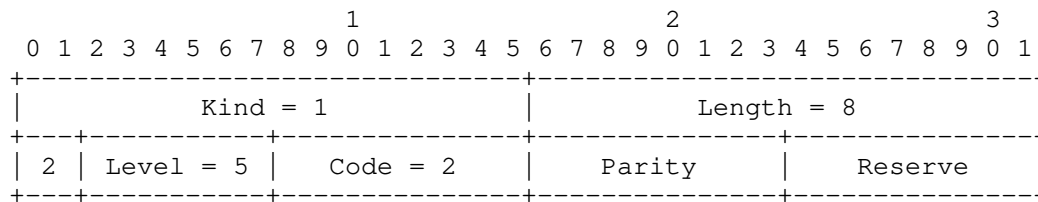


Figure 21: Port Parity Option

o Parity:

- \* 0x00: No particular parity
- \* 0x01: Even

- \* 0x02: Odd
- o Reserve: whether or not to reserve the port's counterpart
- \* 0x00: Don't reserve
- \* 0x01: Reserve

If the UDP ASSOCIATE request does not have the Port field set to 0 (indicating that an arbitrary port can be chosen), the proxy MUST ignore the suggested parity.

A port's counterpart is determined as follows:

- o for even ports, it is the next higher port and
- o for odd ports, it is the next lower port.

If the proxy can not or will not comply with the requested parity, it also does not reserve the allocated port's counterpart.

Port reservations are in place until either:

- o the original association ends, or
- o an association involving the reserved port is made.

An association involving a reserved port can only be made if a client explicitly requests said port. Further, if the original association is part of a session (see Section 8.4), the reserved port can only be claimed from within the same session.

## 8.2. Authentication Method options

A client that is willing to go through the authentication phase MUST include an Authentication Method Advertisement option in its Request. In case of a CONNECT Request, the option is also used to specify the amount of initial data supplied before any method-specific authentication negotiations take place.

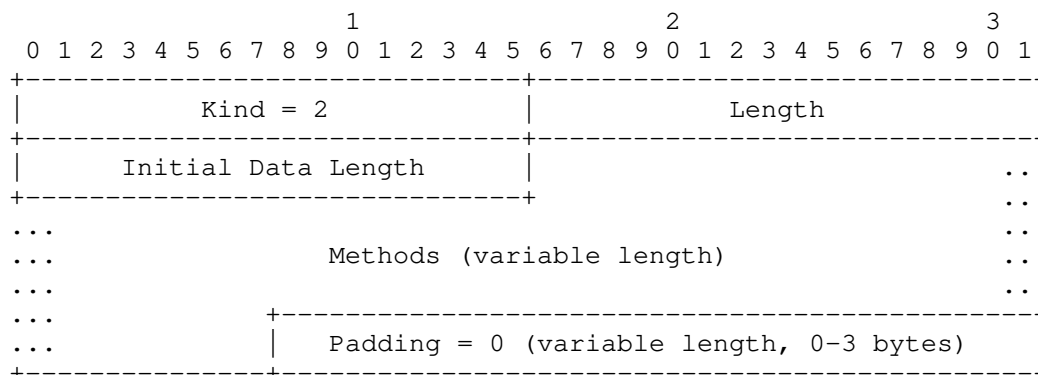


Figure 22: Authentication Method Advertisement Option

- o Initial Data Size: A two-byte number in network byte order. In case of CONNECT, this is the number of bytes of initial data that are supplied by the client immediately following the Request. This number MUST NOT be larger than  $2^{14}$ .
- o Methods: One byte per advertised method. Method numbers are assigned by IANA.
- o Padding: A minimally-sized sequence of zeroes, such that the option length is a multiple of 4. Note that 0 coincides with the value for "No Authentication Required".

Clients MUST support the "No authentication required" method. Clients SHOULD omit advertising the "No authentication required" option.

The proxy indicates which authentication method must proceed by sending an Authentication Method Selection option in the corresponding Authentication Reply:

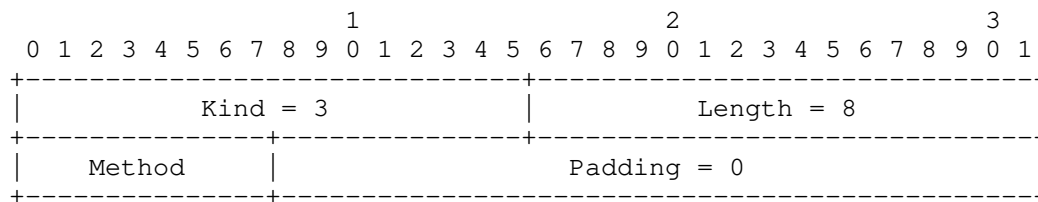


Figure 23: Authentication Method Selection Option



- o Method: The selected method.

If the proxy selects "No Acceptable Methods", the client MUST close the connection.

If authentication is successful via some other means, or not required at all, the proxy silently ignores the Authentication Method Advertisement option.

### 8.3. Authentication Data options

Authentication Data options carry method-specific authentication data. They can be part of SOCKS Requests and Authentication Replies.

Authentication Data options have the following format:

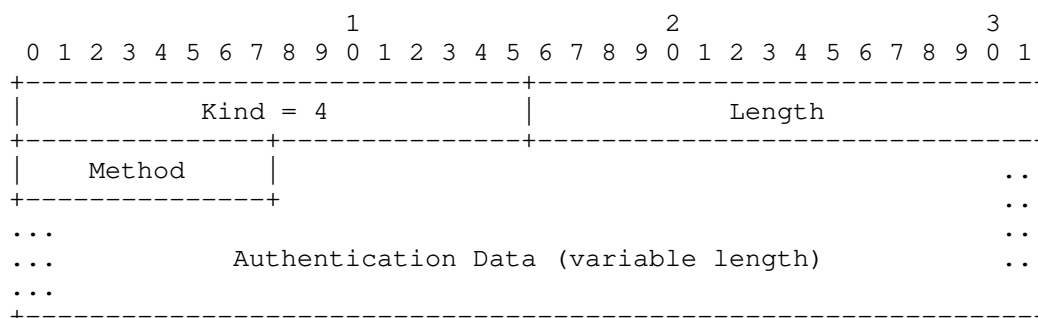


Figure 24: Authentication Data Option

- o Method: The number of the authentication method. These numbers are assigned by IANA.
- o Authentication Data: The contents are specific to each method.

Clients MUST only place one Authentication Data option per authentication method.

### 8.4. Session options

Clients and proxies can establish SOCKS sessions, which span one or more Requests. All session-related negotiations are done via Session Options, which are placed in Requests and Authentication Replies by the client and, respectively, by the proxy.

Client and proxy implementations MUST either support all Session Option Types, or none.

## 8.4.1. Session initiation

A client can initiate a session by sending a Session Request Option:

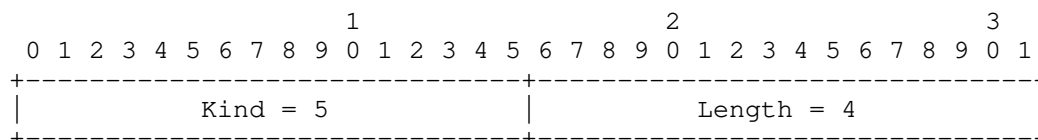


Figure 25: Session Request Option

The proxy then replies with a Session ID Option in the successful Operation Reply:

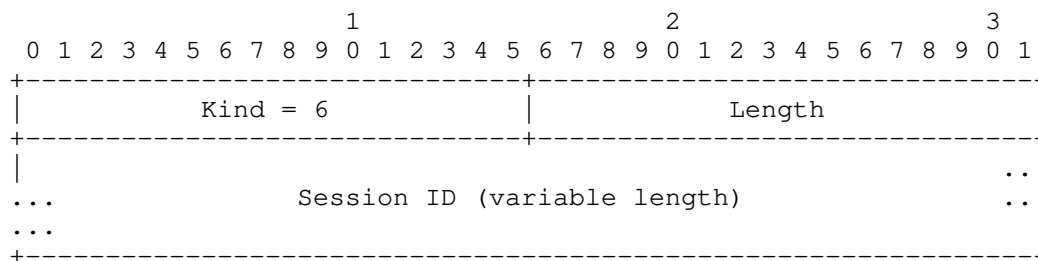


Figure 26: Session ID Option

- o Session ID: An opaque sequence of bytes specific to the session. The size MUST be a multiple of 4. MUST NOT be empty.

The Session ID serves to identify the session and is opaque to the client.

The credentials, or lack thereof, used to initiate the session are tied to the session.

The SOCKS Request that initiated the session is considered part of the session. A client MUST NOT attempt to initiate a session from within a different session.

If the proxy can not or will not honor the Session Request, it does so silently.

#### 8.4.2. Further SOCKS Requests

Any further SOCKS Requests that are part of the session MUST include a Session ID Option (as seen in Figure 26). The proxy MUST silently ignore any authentication attempt in the Request, and MUST NOT require any authentication.

The proxy then replies by placing a Session OK option in the successful Authentication Reply:

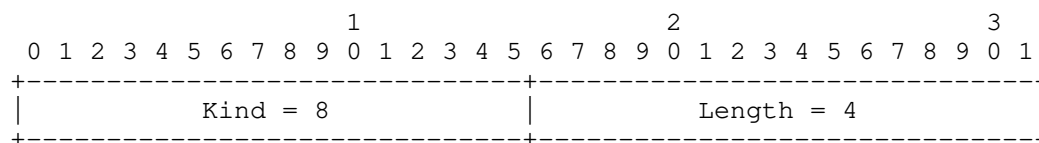


Figure 27: Session OK Option

If the Session ID is invalid, the first Authentication Reply MUST signal that authentication failed and can not continue (by setting the Type field to 0x01). Further, it SHALL contain a Session Invalid option:

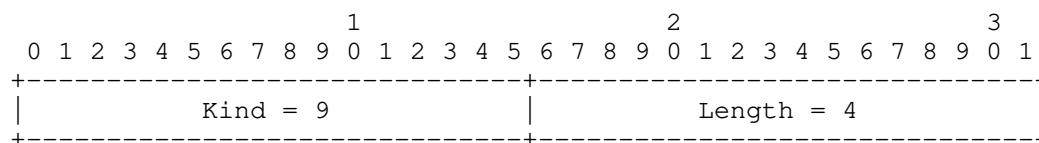


Figure 28: Session Invalid Option

#### 8.4.3. Tearing down the session

Proxies can, at their discretion, tear down a session and free all associated state. Proxy implementations SHOULD feature a timeout mechanism that destroys sessions after a period of inactivity. When a session is terminated, the proxy MAY close all connections associated with said session.

Clients can signal that a session is no longer needed, and can be torn down, by sending a Session Teardown option in addition to the Session ID option:

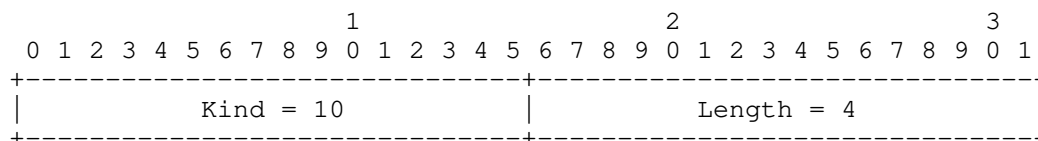


Figure 29: Session Teardown Option

After sending such an option, the client **MUST** assume that the session is no longer valid. The proxy **MUST** treat the session-terminating request as if it were not part of any session.

### 8.5. Idempotence options

To protect against duplicate SOCKS Requests, clients can request, and then spend, idempotence tokens. A token can only be spent on a single SOCKS request.

Tokens are 4-byte unsigned integers in a modular 4-byte space. Therefore, if  $x$  and  $y$  are tokens,  $x$  is less than  $y$  if  $0 < (y - x) < 2^{31}$  in unsigned 32-bit arithmetic.

Proxies grant contiguous ranges of tokens called token windows. Token windows are defined by their base (the first token in the range) and size.

All token-related operations are done via Idempotence options.

Idempotence options are only valid in the context of a SOCKS Session. If a SOCKS Request is not part of a Session (either by supplying a valid Session ID or successfully initiating one via a Session Request), the proxy **MUST** silently ignore any Idempotence options.

Token windows are tracked by the proxy on a per-session basis. There can be at most one token window for every session and its tokens can only be spent from within said session.

Client and proxy implementations **MUST** either support all Idempotence Option Types, or none.

#### 8.5.1. Requesting a token window

A client can obtain a window of tokens by sending an Idempotence Request option as part of a SOCKS Request:

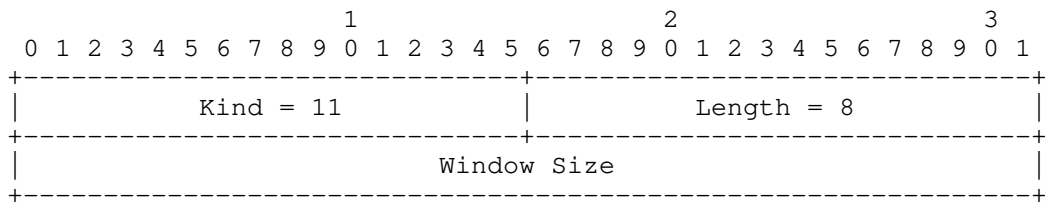


Figure 30: Token Request

- o Window Size: The requested window size.

Once a token window is issued, the proxy MUST include an Idempotence Window option in all subsequent successful Authentication Replies:

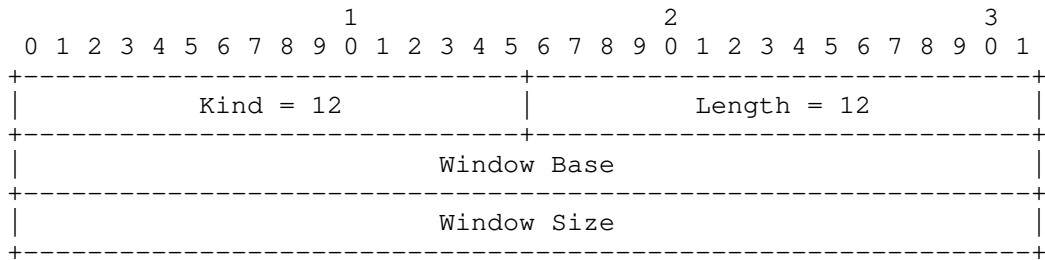


Figure 31: Idempotence Window

- o Window Base: The first token in the window.
- o Window Size: The window size. This value MAY differ from the requested window size. Window sizes MUST be less than 2^31. Window sizes MUST NOT be 0.

If no token window is issued, the proxy MUST silently ignore the Token Request. If there is already a token window associated with the session, the proxy MUST NOT issue a new window.

8.5.2. Spending a token

The client can attempt to spend a token by including a Idempotence Expenditure option in its SOCKS request:

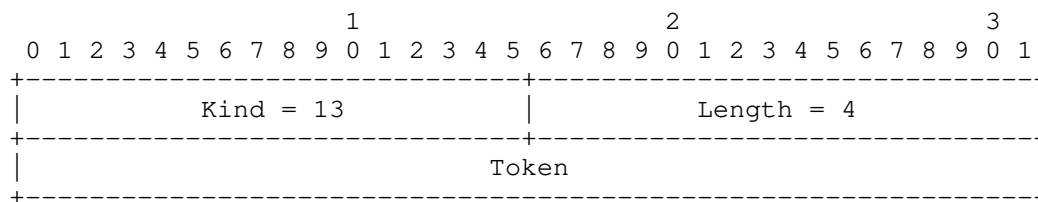


Figure 32: Idempotence Expenditure

- o Kind: 13 (Idempotence Expenditure option)
- o Length: 8
- o Token: The token being spent.

Clients SHOULD prioritize spending the smaller tokens.

The proxy responds by sending either an Idempotence Accepted or Rejected option as part of the Authentication Reply:

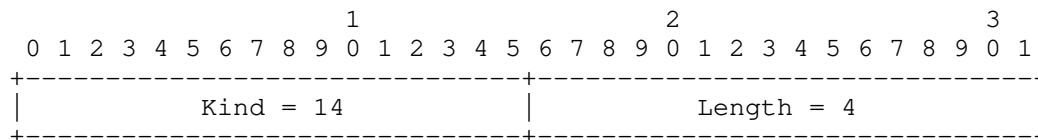


Figure 33: Idempotence Accepted

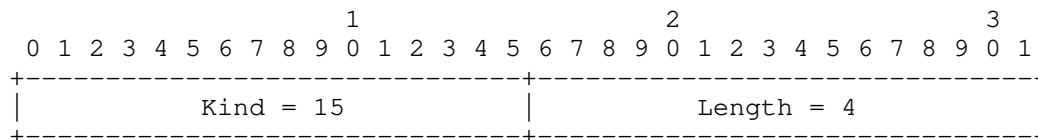


Figure 34: Idempotence Rejected

If eligible, the token is spent before attempting to honor the Request. If the token is not eligible for spending, the Authentication Reply MUST indicate failure.

### 8.5.3. Shifting windows

Windows can be shifted (i. e. have their base increased, while retaining their size) unilaterally by the proxy.

Proxy implementations SHOULD shift the window: \* as soon as the lowest-order token in the window is spent and \* when a sufficiently high-order token is spent.

Proxy implementations SHOULD NOT shift the window's base beyond the highest unspent token.

#### 8.5.4. Out-of-order Window Advertisements

Even though the proxy increases the window's base monotonically, there is no mechanism whereby a SOCKS client can receive the Token Window Advertisements in order. As such, clients SHOULD disregard Token Window Advertisements with a Window Base less than the previously known value.

## 9. Username/Password Authentication

Username/Password authentication is carried out as in [RFC1929].

Clients can also attempt to authenticate by placing the Username/Password request in an Authentication Data Option.

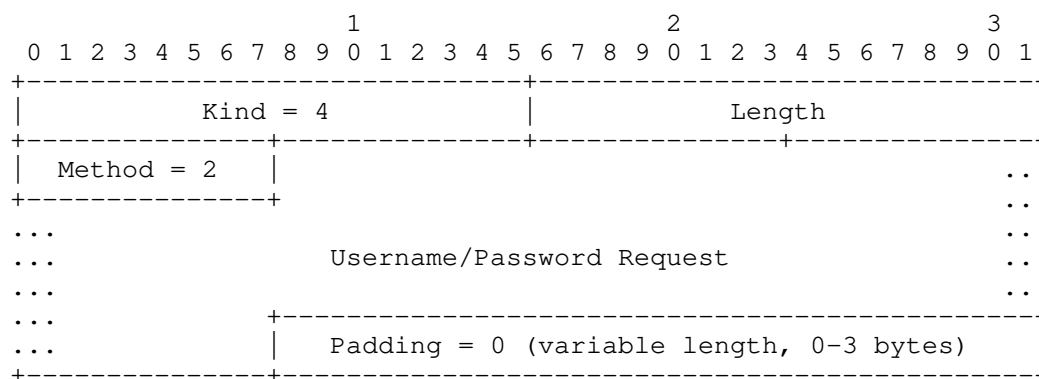


Figure 35: Password authentication via a SOCKS Option

- o Username/Password Request: The Username/Password Request, as described in [RFC1929].

Proxies reply by including a Authentication Data Option in the next Authentication Reply which contains the Username/Password reply:

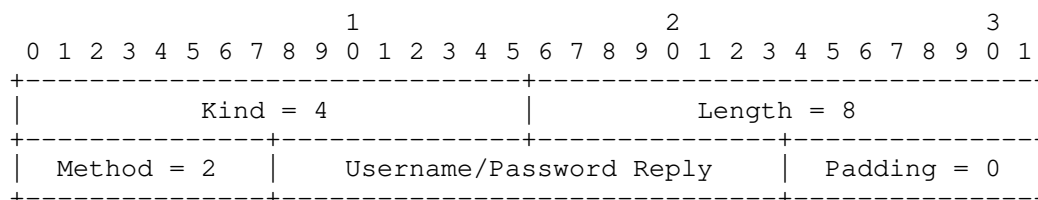


Figure 36: Reply to password authentication via a SOCKS Option

- o Username/Password Reply: The Username/Password Reply, as described in [RFC1929].

#### 10. TCP Fast Open on the Client-Proxy Leg

TFO breaks TCP semantics, causing replays of the data in the SYN's payload under certain rare circumstances [RFC7413]. A replayed SOCKS Request could itself result in a replayed connection on behalf of the client.

As such, client implementations SHOULD NOT use TFO on the client-proxy leg unless:

- o The protocol running on top of SOCKS tolerates the risks of TFO, or
- o The SYN's payload does not contain any application data (so that no data is replayed to the server, even though duplicate connections are still possible), or
- o The client uses Idempotence Options, making replays very unlikely, or
- o SOCKS is running on top of TLS and Early Data is not used.

#### 11. False Starts

In case of CONNECT Requests, the client MAY start sending application data as soon as possible, as long as doing so does not incur the risk of breaking the SOCKS protocol.

Clients must work around the authentication phase by doing any of the following:



- o If the Request does not contain an Authentication Method Advertisement option, the authentication phase is guaranteed not to happen. In this case, application data MAY be sent immediately after the Request.
- o Application data MAY be sent immediately after receiving an Authentication Reply indicating success.
- o When performing a method-specific authentication sequence, application data MAY be sent immediately after the last client message.

## 12. DNS provided by SOCKS

Clients may require information typically obtained from DNS servers, albeit from the proxy's vantage point.

While the CONNECT command can work with domain names, some clients' workflows require that addresses be resolved as a separate step prior to connecting. Moreover, the SOCKS Datagram Header, as described in Section 7.3, can be reduced in size by providing the resolved destination IP address, rather than the FQDN.

Emerging techniques may also make use of DNS to deliver server-specific information to clients. For example, Encrypted SNI [I-D.ietf-tls-esni] relies on DNS to publish encryption keys.

Proxy implementations MAY provide a default plaintext DNS service. A client looking to make use of it issues a CONNECT Request to IP address 0.0.0.0 or 0:0:0:0:0:0:0:0 on port 53. Following successful authentication, the Operation Reply MAY indicate an unspecified bind address (0.0.0.0 or ::) and port (0). The client and proxy then behave as per [RFC7766].

The service itself can be provided directly by the proxy daemon, or by proxying the client's request to a pre-configured DNS server.

If the proxy does not implement such functionality, it MAY return an error code signaling "Connection refused".

## 13. Security Considerations

### 13.1. Large requests

Given the format of the request message, a malicious client could craft a request that is in excess of 16 KB and proxies could be prone to DDoS attacks.

To mitigate such attacks, proxy implementations SHOULD be able to incrementally parse the requests. Proxies MAY close the connection to the client if:

- o the request is not fully received after a certain timeout, or
- o the number of options or their size exceeds an imposed hard cap.

### 13.2. Replay attacks

In TLS 1.3, early data (which is likely to contain a full SOCKS request) is prone to replay attacks.

While Token Expenditure options can be used to mitigate replay attacks, anything prior to the initial Token Request is still vulnerable. As such, client implementations SHOULD NOT make use of TLS early data unless the Request attempts to spend a token.

### 13.3. Resource exhaustion

Malicious clients can issue a large number of Session Requests, forcing the proxy to keep large amounts of state.

To mitigate this, the proxy MAY implement policies restricting the number of concurrent sessions on a per-IP or per-user basis, or barring unauthenticated clients from establishing sessions.

## 14. Privacy Considerations

The timing of Operation Replies can reveal some information about a proxy's recent usage:

- o The DNS resolver used by the proxy may cache the answer to recent queries. As such, subsequent connection attempts to the same hostname are likely to be slightly faster, even if requested by different clients.
- o Likewise, the proxy's OS typically caches TFO cookies. Repeated TFO connection attempts tend to be sped up, regardless of the client.

## 15. IANA Considerations

This document requests that IANA allocate 2-byte option kinds for SOCKS 6 options. Further, this document requests the following option kinds:

- o Unassigned: 0

- o Stack: 1
- o Authentication Method Advertisement: 2
- o Authentication Method Selection: 3
- o Authentication Data: 4
- o Session Request: 5
- o Session ID: 6
- o Session OK: 8
- o Session Invalid: 9
- o Session Teardown: 10
- o Idempotence Request: 11
- o Idempotence Window: 12
- o Idempotence Expenditure: 13
- o Idempotence Accepted: 14
- o Idempotence Rejected: 15
- o Resolution Request: 16
- o IPv4 Resolution: 17
- o IPv6 Resolution: 18
- o Experimental: 64512-0xFFFF

This document also requests that IANA allocate a TCP and UDP port for SOCKS over TLS and DTLS, respectively.

## 16. Acknowledgments

The protocol described in this draft builds upon and is a direct continuation of SOCKS 5 [RFC1928].

## 17. References

### 17.1. Normative References

- [RFC1929] Leech, M., "Username/Password Authentication for SOCKS V5", RFC 1929, DOI 10.17487/RFC1929, March 1996, <<https://www.rfc-editor.org/info/rfc1929>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7766] Dickinson, J., Dickinson, S., Bellis, R., Mankin, A., and D. Wessels, "DNS Transport over TCP - Implementation Requirements", RFC 7766, DOI 10.17487/RFC7766, March 2016, <<https://www.rfc-editor.org/info/rfc7766>>.
- [RFC8305] Schinazi, D. and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency", RFC 8305, DOI 10.17487/RFC8305, December 2017, <<https://www.rfc-editor.org/info/rfc8305>>.

### 17.2. Informative References

- [I-D.ietf-tls-dtls-connection-id] Rescorla, E., Tschofenig, H., and T. Fossati, "Connection Identifiers for DTLS 1.2", draft-ietf-tls-dtls-connection-id-07 (work in progress), October 2019.
- [I-D.ietf-tls-esni] Rescorla, E., Oku, K., Sullivan, N., and C. Wood, "TLS Encrypted Client Hello", draft-ietf-tls-esni-08 (work in progress), October 2020.
- [RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 1928, DOI 10.17487/RFC1928, March 1996, <<https://www.rfc-editor.org/info/rfc1928>>.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <<https://www.rfc-editor.org/info/rfc3550>>.

- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <<https://www.rfc-editor.org/info/rfc6824>>.
- [RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <<https://www.rfc-editor.org/info/rfc7413>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

#### Authors' Addresses

Vladimir Olteanu  
University Politehnica of Bucharest  
313 Splaiul Independentei, Sector 6  
Bucharest  
Romania

Email: [vladimir.olteanu@cs.pub.ro](mailto:vladimir.olteanu@cs.pub.ro)

Dragos Niculescu  
University Politehnica of Bucharest  
313 Splaiul Independentei, Sector 6  
Bucharest  
Romania

Email: [dragos.niculescu@cs.pub.ro](mailto:dragos.niculescu@cs.pub.ro)

Internet Engineering Task Force  
Internet-Draft  
Updates: 4960 (if approved)  
Intended status: Standards Track  
Expires: December 3, 2020

M. Proshin  
Ericsson  
June 01, 2020

Retransmit bit for SCTP DATA, I-DATA and SACK  
draft-proshin-tsvwg-sctp-rtx-bit-03

Abstract

This document defines a method which helps an SCTP sender to understand when a received SACK acknowledges the original transmission of a TSN or its retransmission. It is done by specifying a new bit, called Retransmit bit (R-bit), in the header of DATA, I-DATA and SACK chunks. The bit is used when a TSN is retransmitted and returned back in the acknowledgement. This document updates [RFC4960] if approved.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 3, 2020.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Conventions . . . . .	3
3. Updates in SCTP Chunks Header . . . . .	3
3.1. R-bit in DATA Chunk Header . . . . .	3
3.2. R-bit in I-DATA Chunk Header . . . . .	4
3.3. R-bit in SACK Chunk Header . . . . .	4
4. Procedures . . . . .	5
4.1. Negotiation . . . . .	5
4.2. Sender Side Considerations . . . . .	6
4.3. Receiver Side Considerations . . . . .	7
4.4. Processing of SACK with and without R-bit . . . . .	7
5. R-bit vs Duplicate TSN for Detection of Spurious Retransmission . . . . .	8
6. Interoperability Considerations . . . . .	9
7. Socket API Considerations . . . . .	9
8. Acknowledgements . . . . .	9
9. IANA Considerations . . . . .	9
10. Security Considerations . . . . .	11
11. References . . . . .	11
11.1. Normative References . . . . .	11
11.2. Informative References . . . . .	11
Author's Address . . . . .	12

## 1. Introduction

SCTP which is defined in [RFC4960] is a reliable message-oriented protocol. The SCTP sender splits user messages to DATA chunks and sends them to the receiver. The SCTP receiver uses the SACK chunk to acknowledge incoming data. The reliability in SCTP is achieved by the retransmission of DATA chunks which were not acknowledged.

If a DATA chunk has been retransmitted at least once, at SACK reception SCTP cannot understand if the SACK was sent in response to the originally sent DATA or retransmitted one. Thus, due to that ambiguity, [RFC4960] prohibits making RTT measurements. Some other SCTP mechanisms such as loss recovery and congestion control are not accurate in that case either.

This document describes a simple extension of the DATA and SACK chunks by a new bit, so called Retransmit bit (R-bit). The sender sets the R-bit in the DATA chunk header when it retransmits a DATA and the receiver sets it in the SACK chunk header when a DATA with

R-bit is acknowledged. The sender can now distinguish when a SACK acknowledges the originally sent DATA or retransmitted one. The extension requires support by the sender and the receiver.

The mechanism described in this document is equally relevant for I-DATA chunk which is introduced in [RFC8260].

## 2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [[RFC8174]] when, and only when, they appear in all capitals, as shown here.

## 3. Updates in SCTP Chunks Header

### 3.1. R-bit in DATA Chunk Header

Figure 1 describes the extended DATA chunk header.

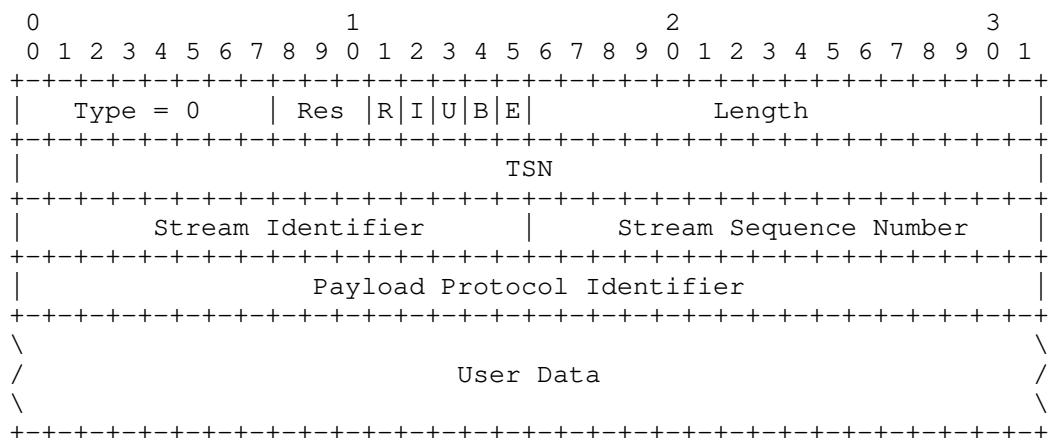


Figure 1: Extended DATA chunk

The only difference between the DATA chunk in Figure 1 and the DATA chunk defined in [RFC4960] is the addition of the R-bit in the flags field of the DATA chunk header. [RFC4960] specified that bit as Reserved and that it should be set to 0 by the sender and ignored by the receiver.



### 3.2. R-bit in I-DATA Chunk Header

Figure 2 describes the extended DATA chunk header.

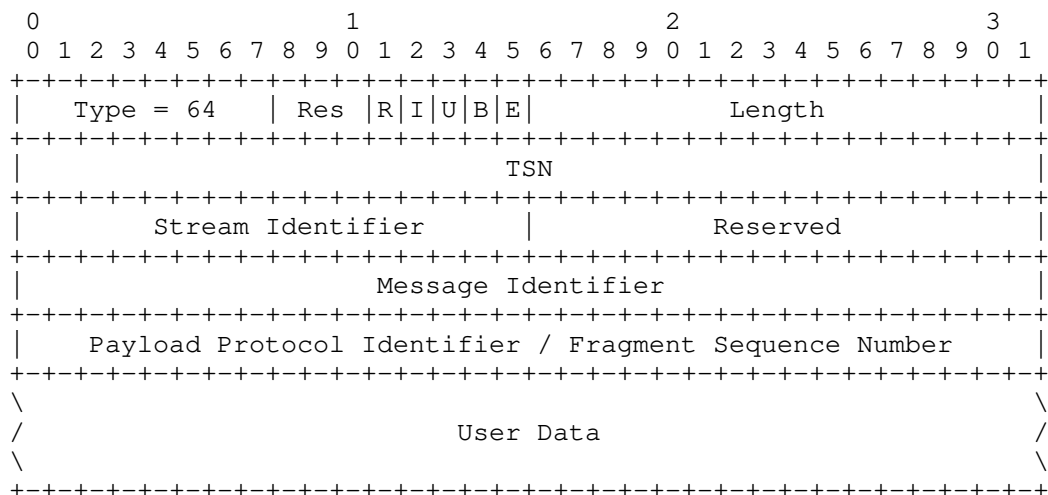


Figure 2: Extended I-DATA chunk

The only difference between the I-DATA chunk in Figure 2 and the I-DATA chunk defined in [RFC8260] is the addition of the R-bit in the flags field of the I-DATA chunk header. [RFC8260] specified that bit as Reserved and that it should be set to 0 by the sender and ignored by the receiver.

### 3.3. R-bit in SACK Chunk Header

Figure 3 describes the extended SACK chunk header.

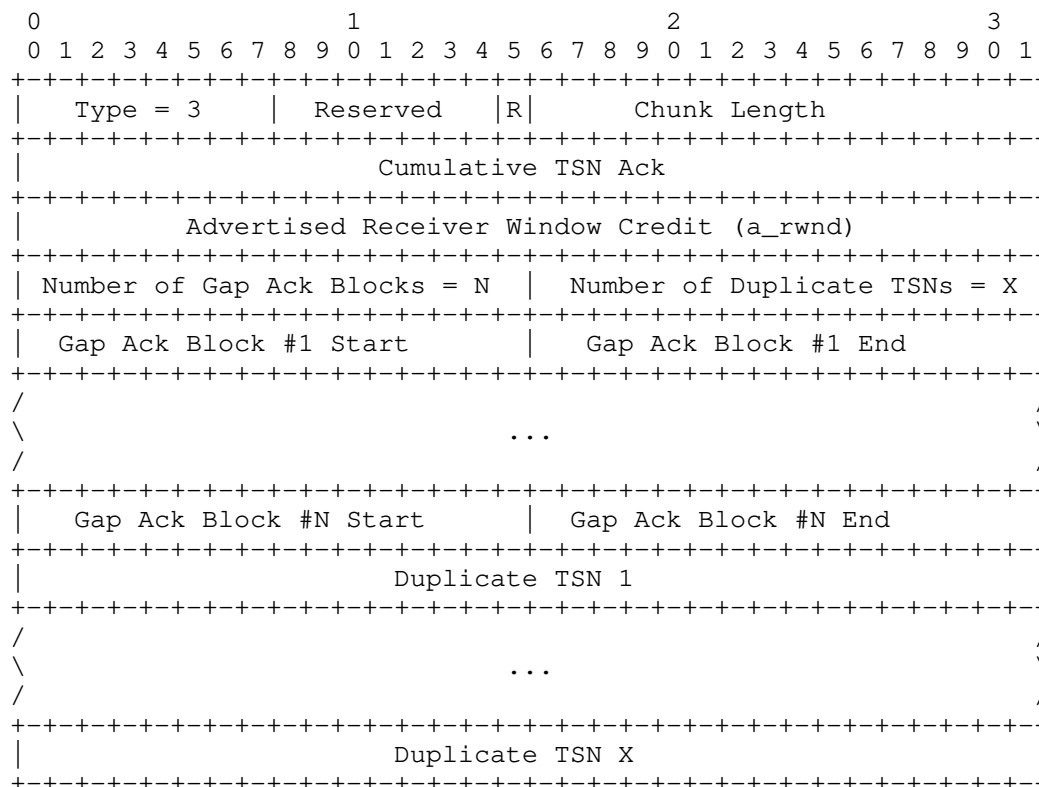


Figure 3: Extended SACK chunk

The only difference between the SACK chunk in Figure 3 and the SACK chunk defined in [RFC4960] is the addition of the R-bit in the flags field of the SACK chunk header. [RFC4960] specified that bit as Reserved and that it should be set to 0 by the sender and ignored by the receiver.

#### 4. Procedures

##### 4.1. Negotiation

R-bit MUST NOT be used unless both SCTP peers negotiated its support.

The following new optional parameter is added to the INIT and INIT ACK chunks to negotiate R-bit support during association setup:

Parameter Type	Parameter Name
0x8100	Retransmit Bit Supported (RBIT-SUPPORTED)

Table 1

The parameter format is the following:

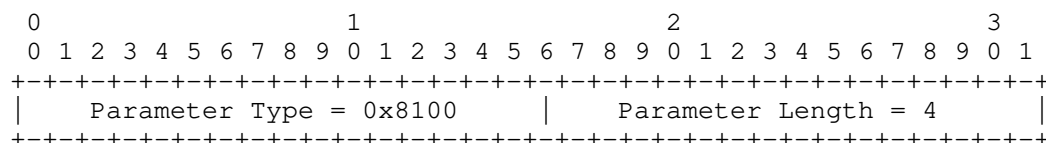


Figure 4: Format of RBIT-SUPPORTED

Parameter Type: 2 bytes (unsigned integer)

This value MUST be set to 0x8100 (33024).

Parameter Length: 2 bytes (unsigned integer)

This value MUST be set to 4.

The RBIT-SUPPORTED parameter MAY be included once in the INIT or INIT ACK chunk if the sender wants to inform its peer that it supports R-bit.

The new parameter type is encoded so that it requires the receiver to skip it and continue processing if the parameter is not recognized according to [RFC4960].

#### 4.2. Sender Side Considerations

SCTP MUST NOT set the R-bit when it sends a DATA or I-DATA chunk first time.

If R-bit support is negotiated as described in Section 4.1, SCTP SHOULD set the R-bit every time it retransmits a DATA or I-DATA chunk. This is regardless of if the chunk is retransmitted on the same path or on an alternative one.

Note that it is possible that the same SCTP packet includes DATA or I-DATA chunks with and without the R-bit set in case when SCTP bundles chunks which are marked for retransmission with chunks which are sent first time. This is aligned with [RFC4960] which allows

bundling of DATA chunks marked for retransmission with new DATA chunks.

IMPLEMENTATION NOTE: According to [RFC4960] new DATA chunks always follow DATA chunks marked for retransmission when bundled in one packet.

#### 4.3. Receiver Side Considerations

SCTP MUST NOT set the R-bit when it sends a SACK which acknowledges a DATA or I-DATA chunk without the R-bit set. The delay for a SACK without the R-bit set is defined according to [RFC4960].

When SCTP receives a packet with DATA or I-DATA chunk(s) with the R-bit set, it MUST immediately respond with a SACK with the R-bit set acknowledging only DATA or I-DATA chunks where the R-bit was set. If the packet also contains DATA or I-DATA chunk(s) without the R-bit set, SCTP MUST NOT acknowledge them in the same SACK chunk.

TBD: SACK with the R-bit bundled with SACK without the R-bit? It may be useful.

#### 4.4. Processing of SACK with and without R-bit

If a DATA or I-DATA was retransmitted and the corresponding SACK is received, SCTP can distinguish if the SACK acknowledges the original transmission or retransmission by checking the R-bit in the SACK. SCTP mechanisms which can be improved by that information include, but are not limited to, the following:

- o RTO Calculation: [RFC4960] refers to Karn's algorithm and prohibits SCTP to make RTT measurements using packets that were retransmitted and for which it is ambiguous whether the reply was for the original transmission or retransmission(s).
- o Path Failure Detection: [RFC4960] specifies that the sender may choose not to clear the path error counter if there is undesirable ambiguity when a DATA is retransmitted on an alternative path.
- o SCTP-PF Operation in [RFC7829]: additionally to the path error counter case described in the previous bullet [RFC7829] also does not recommend to move a destination address in PF state back to the active state in case of ambiguity.
- o Detection of spurious retransmissions: using R-bit SCTP can detect spurious retransmissions. Namely, if a DATA was retransmitted and SACK acknowledging it does not include R-bit, it means that the retransmission was spurious. Note that this is valid even if a

DATA was retransmitted multiple times which makes this method more effective than detecting of spurious retransmissions based on DSACK. When a spurious retransmission is detected, SCTP implementation may:

- \* Choose to revert the congestion control state.
  - \* Choose to adjust RTO settings such as the RTO.Min value to mitigate further spurious retransmissions.
  - \* Indicate the SCTP user.
- o SCTP latency of retransmitted data: If the original DATA is lost, the SCTP receiver will immediately acknowledge the retransmitted DATA.
  - o Calculation of Maximum Ack Delay: SCTP implementations can support a technique for calculating of Maximum Ack Delay in run-time which is impossible to do properly in case of retransmissions. With R-bit SCTP can distinguish if the SACK acknowledges the original transmission or retransmission and can measure the delay even for a retransmitted DATA.
  - o Measurement of packet loss: R-bit can be used for passive loss rate calculation.

TBD: dup TSN but without R-bit: SACK loss or reordering: Can be used somehow?

Note that this document does not solve the problem when the same DATA or I-DATA chunk is retransmitted multiple times. In that case, when SCTP receives a SACK without the R-bit set, it can ensure that the SACK acknowledges the original transmission but when SCTP receives a SACK with the R-bit set, it cannot distinguish which retransmission is actually acknowledged. Such limitation is not considered as severe because multiple retransmissions of the same DATA or I-DATA is a corner case and, if it happens, SCTP transmission is anyway inefficient.

## 5. R-bit vs Duplicate TSN for Detection of Spurious Retransmission

The SACK chunk according to [RFC4960] contains the Duplicate TSN field which is used by the receiver to indicate TSNs received multiple times. This could happen due to spurious retransmissions or if packets were duplicated in the network between endpoints. The Duplicate TSN field in the SACK chunk can also be used by the sender to detect spurious retransmissions in some cases. However, the

mechanism based on the Duplicate TSN field would have serious limitations compared to the mechanism based on R-bit:

- o With R-bit the SCTP sender has an exclusive match between DATA and SACK while in case of the Duplicate TSN it is not guaranteed. Thus, if the original or retransmitted DATA is lost or one of the SACKs is lost or the packets were retransmitted, the SCTP sender cannot rely on the Duplicate TSN field.
- o Even in those cases where the sender could rely on the Duplicate TSN, it would need to wait the second SACK to detect the spurious retransmission, while with R-bit, the sender can detect it as soon as the first SACK is received.
- o In case of the Duplicate TSN the SCTP sender needs to keep information about the retransmitted TSN until the second SACK is received or during some time period which impacts memory usage and SCTP performance and complicates implementation.

## 6. Interoperability Considerations

This document does not introduce any interoperability issues. Section 4.1 requires both ends to negotiate R-bit support before its usage. [RFC4960] requires the receiver of a DATA or SACK chunk with the R-bit set to ignore the bit if it is not recognized. [RFC8260] requires the receiver of an I-DATA chunk with the R-bit set to ignore the bit if it is not recognized.

## 7. Socket API Considerations

This document does not address any changes to the socket API defined in [RFC6458].

## 8. Acknowledgements

TBD

## 9. IANA Considerations

[NOTE to RFC-Editor:

"RFCXXXX" is to be replaced by the RFC number you assign this document.

]

IANA should assign 33024 (0x8100) as a new parameter type to SCTP.

Following the chunk flag registration procedure defined in [RFC6096], IANA should register a new bit, the R-bit, for the DATA chunk. The suggested value is 0x10 and the reference should be RFCXXXX.

This requires an update of the "DATA Chunk Flags" registry for SCTP:

Chunk Flag Value	Chunk Flag Name	Reference
0x01	E bit	[RFC4960]
0x02	B bit	[RFC4960]
0x04	U bit	[RFC4960]
0x08	I bit	[RFC7053]
0x10	R bit	RFCXXXX
0x20	Unassigned	
0x40	Unassigned	
0x80	Unassigned	

Table 2

Following the chunk flag registration procedure defined in [RFC6096], IANA should register a new bit, the R-bit, for the SACK chunk. The suggested value is 0x01 and the reference should be RFCXXXX.

This requires an update of the "SACK Chunk Flags" registry for SCTP:

Chunk Flag Value	Chunk Flag Name	Reference
0x01	R bit	RFCXXXX
0x02	Unassigned	
0x04	Unassigned	
0x08	Unassigned	
0x10	Unassigned	
0x20	Unassigned	
0x40	Unassigned	
0x80	Unassigned	

Table 3

Following the chunk flag registration procedure defined in [RFC6096], IANA should register a new bit, the R-bit, for the I-DATA chunk. The suggested value is 0x10 and the reference should be RFCXXXX.

This requires an update of the "I-DATA Chunk Flags" registry for SCTP:

Chunk Flag Value	Chunk Flag Name	Reference
0x01	E bit	[RFC8260]
0x02	B bit	[RFC8260]
0x04	U bit	[RFC8260]
0x08	I bit	[RFC8260]
0x10	R bit	RFCXXXX
0x20	Unassigned	
0x40	Unassigned	
0x80	Unassigned	

Table 4

## 10. Security Considerations

This document does not introduce any additional security considerations in addition to the ones described in [RFC4960] and [RFC8260].

## 11. References

### 11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.
- [RFC8260] Stewart, R., Tuexen, M., Loreto, S., and R. Seggelmann, "Stream Schedulers and User Message Interleaving for the Stream Control Transmission Protocol", RFC 8260, DOI 10.17487/RFC8260, November 2017, <<https://www.rfc-editor.org/info/rfc8260>>.

### 11.2. Informative References

- [RFC6096] Tuexen, M. and R. Stewart, "Stream Control Transmission Protocol (SCTP) Chunk Flags Registration", RFC 6096, DOI 10.17487/RFC6096, January 2011, <<https://www.rfc-editor.org/info/rfc6096>>.



- [RFC6458] Stewart, R., Tuexen, M., Poon, K., Lei, P., and V. Yasevich, "Sockets API Extensions for the Stream Control Transmission Protocol (SCTP)", RFC 6458, DOI 10.17487/RFC6458, December 2011, <<https://www.rfc-editor.org/info/rfc6458>>.
- [RFC7053] Tuexen, M., Ruengeler, I., and R. Stewart, "SACK-IMMEDIATELY Extension for the Stream Control Transmission Protocol", RFC 7053, DOI 10.17487/RFC7053, November 2013, <<https://www.rfc-editor.org/info/rfc7053>>.
- [RFC7829] Nishida, Y., Natarajan, P., Caro, A., Amer, P., and K. Nielsen, "SCTP-PF: A Quick Failover Algorithm for the Stream Control Transmission Protocol", RFC 7829, DOI 10.17487/RFC7829, April 2016, <<https://www.rfc-editor.org/info/rfc7829>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

## Author's Address

Maksim Proshin  
Ericsson  
Kistavaegen 25  
Stockholm 164 80  
Sweden

Email: [mproshin@tieto.mera.ru](mailto:mproshin@tieto.mera.ru)

TSVWG  
Internet-Draft  
Intended status: Standards Track  
Expires: September 6, 2019

M. Saito  
M. Matsumoto  
Hiroshima University  
V. Roca (Ed.)  
E. Baccelli  
INRIA  
March 5, 2019

TinyMT32 Pseudo Random Number Generator (PRNG)  
draft-roca-tsvwg-tinymt32-01

Abstract

This document describes the TinyMT32 Pseudo Random Number Generator (PRNG) that produces 32-bit pseudo-random unsigned integers and aims at having a simple-to-use and deterministic solution. This PRNG is a small-sized variant of Mersenne Twister (MT) PRNG, also designed by M. Saito and M. Matsumoto. The main advantage of TinyMT32 over MT is the use of a small internal state, compatible with most target platforms including embedded devices, while keeping a reasonably good randomness.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 6, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Definitions . . . . .	3
3. TinyMT32 PRNG Specification . . . . .	3
3.1. TinyMT32 Source Code . . . . .	3
3.2. TinyMT32 Usage . . . . .	7
3.3. Specific Implementation Validation and Deterministic Behavior . . . . .	8
4. Security Considerations . . . . .	9
5. IANA Considerations . . . . .	9
6. Acknowledgments . . . . .	9
7. References . . . . .	9
7.1. Normative References . . . . .	9
7.2. Informative References . . . . .	9
Authors' Addresses . . . . .	10

## 1. Introduction

This document specifies the TinyMT32 PRNG, as a specialization of the reference implementation version 1.1 (2015/04/24) by Mutsuo Saito and Makoto Matsumoto, from Hiroshima University:

- o Official web site: <<http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/TINYMT/>>
- o Official github site and reference implementation: <<https://github.com/MersenneTwister-Lab/TinyMT>>

This specialisation aims at having a simple-to-use and deterministic PRNG, as explained below.

TinyMT is a new small-sized variant of Mersenne Twister (MT) introduced by Mutsuo Saito and Makoto Matsumoto in 2011. This document focusses on the TinyMT32 variant (rather than TinyMT64) of the PRNG, which outputs 32-bit unsigned integers.

The purpose of TinyMT is not to replace Mersenne Twister: TinyMT has a far shorter period than MT. The merit of TinyMT is in its small size of the internal state of 127 bits, far smaller than 19937 bits of MT. According to statistical tests (BigCrush in TestU01 <<http://simul.iro.umontreal.ca/testu01/tu01.html>> and AdaptiveCrush

<<http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/ADAPTIVE/>>) the quality of the outputs of TinyMT seems pretty good, taking the small size of the internal state into consideration. From this point of view, TinyMT32 represents a major improvement with respect to the Park-Miller Linear Congruential PRNG (e.g., as specified in [RFC5170]).

The TinyMT32 PRNG initialization depends, among other things, on a parameter set -- namely (mat1, mat2, tmat) -- that needs to be well chosen (pre-calculated values are available in the official web site). In order to facilitate the use of this PRNG, and unlike the implementation version 1.1 (2015/04/24) by Mutsuo Saito and Makoto Matsumoto, this specification requires the use of a specific parameter set (see Section 3.1). The implementation version 1.1 (2015/04/24) also proposes two initialisation functions that differ on the approach to seed the PRNG. A second difference is the removal of the `tinymt32_init_by_array()` function to keep only the simple initialisation through a single seed value (see Section 3.2).

Finally, the determinism of this PRNG, for a given seed, has been carefully checked (see Section 3.3). Indeed, this determinism can be a key requirement as it the case with [RLC-ID] that normatively depends on this specification.

## 2. Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 3. TinyMT32 PRNG Specification

### 3.1. TinyMT32 Source Code

The TinyMT32 PRNG requires to be initialized with a parameter set that needs to be well chosen. In this specification, for the sake of simplicity, the following parameter set MUST be used:

- o mat1 = 0x8f7011ee = 2406486510
- o mat2 = 0xfc78ff1f = 4235788063
- o tmat = 0x3793fdff = 932445695

This parameter set is the first entry of the precalculated parameter sets in file `tinymt32dc.0.1048576.txt`, by Kenji Rikitake, and available at <<https://github.com/jjlbdx/tinymtdc->

longbatch/blob/master/tinymt32dc/tinymt32dc.0.1048576.txt>. This is also the parameter set used in [KR12].

The TinyMT32 PRNG reference implementation is reproduced in Figure 1, with the following differences with respect to the original source code:

- o the original copyright and licence have been removed, in accordance with BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>);
- o the source code initially spread over the tinymt32.h and tinymt32.c files has been merged;
- o the unused parts of the original source code have been removed. This is the case of the tinymt32\_init\_by\_array() alternative initialisation function;
- o the unused constants TINYMT32\_MEXP and TINYMT32\_MUL have been removed;
- o the appropriate parameter set has been added to the initialization function;
- o the function order has been changed;
- o certain internal variables have been renamed for compactness purposes;
- o the const qualifier has been added to the constant definitions.

<CODE BEGINS>

```
/**
 * Tiny Mersenne Twister only 127 bit internal state.
 * Derived from the reference implementation version 1.1 (2015/04/24)
 * by Mutsuo Saito (Hiroshima University) and Makoto Matsumoto
 * (Hiroshima University).
 */
#include <stdint.h>

/**
 * tinymt32 internal state vector and parameters
 */
typedef struct {
    uint32_t status[4];
    uint32_t mat1;
    uint32_t mat2;
    uint32_t tmat;
} tinymt32_t;

static void tinymt32_next_state (tinymt32_t * s);
static uint32_t tinymt32_temper (tinymt32_t * s);

/**
 * Parameter set to use for this IETF specification. Don't change.
```

```

* This parameter set is the first entry of the precalculated
* parameter sets in file tinymt32dc.0.1048576.txt, by Kenji
* Rikitake, available at:
*   https://github.com/jj1bdx/tinymt32dc-longbatch/blob/master/
*   tinymt32dc/tinymt32dc.0.1048576.txt
* It is also the parameter set used:
*   Rikitake, K., "TinyMT Pseudo Random Number Generator for
*   Erlang", ACM 11th SIGPLAN Erlang Workshop (Erlang'12),
*   September, 2012.
*/
const uint32_t TINYMT32_MAT1_PARAM = UINT32_C(0x8f7011ee);
const uint32_t TINYMT32_MAT2_PARAM = UINT32_C(0xfc78ff1f);
const uint32_t TINYMT32_TMAT_PARAM = UINT32_C(0x3793fdff);

/**
* This function initializes the internal state array with a
* 32-bit unsigned integer seed.
* @param s      pointer to tinymt internal state.
* @param seed   a 32-bit unsigned integer used as a seed.
*/
void tinymt32_init (tinymt32_t * s, uint32_t seed)
{
    const uint32_t MIN_LOOP = 8;
    const uint32_t PRE_LOOP = 8;
    s->status[0] = seed;
    s->status[1] = s->mat1 = TINYMT32_MAT1_PARAM;
    s->status[2] = s->mat2 = TINYMT32_MAT2_PARAM;
    s->status[3] = s->tmat = TINYMT32_TMAT_PARAM;
    for (int i = 1; i < MIN_LOOP; i++) {
        s->status[i & 3] ^= i + UINT32_C(1812433253)
            * (s->status[(i - 1) & 3]
                ^ (s->status[(i - 1) & 3] >> 30));
    }
    /**
    * NB: the parameter set of this specification warrants
    * that none of the possible 2^32 seeds leads to an
    * all-zero 127-bit internal state. Therefore, the
    * period_certification() function of the original
    * TinyMT32 source code has been safely removed. If
    * another parameter set is used, this function will
    * have to be re-introduced here.
    */
    for (int i = 0; i < PRE_LOOP; i++) {
        tinymt32_next_state(s);
    }
}

/**

```

```

* This function outputs a 32-bit unsigned integer from
* the internal state.
* @param s      pointer to tinymt internal state.
* @return       32-bit unsigned integer r ( $0 \leq r < 2^{32}$ ).
*/
uint32_t tinymt32_generate_uint32 (tinymt32_t * s)
{
    tinymt32_next_state(s);
    return tinymt32_temper(s);
}

/**
* Internal tinymt32 constants and functions.
* Users should not call these functions directly.
*/
const uint32_t TINYMT32_SH0 = 1;
const uint32_t TINYMT32_SH1 = 10;
const uint32_t TINYMT32_SH8 = 8;
const uint32_t TINYMT32_MASK = UINT32_C(0x7fffffff);

/**
* This function changes the internal state of tinymt32.
* @param s      pointer to tinymt internal state.
*/
static void tinymt32_next_state (tinymt32_t * s)
{
    uint32_t x;
    uint32_t y;

    y = s->status[3];
    x = (s->status[0] & TINYMT32_MASK)
        ^ s->status[1]
        ^ s->status[2];
    x ^= (x << TINYMT32_SH0);
    y ^= (y >> TINYMT32_SH0) ^ x;
    s->status[0] = s->status[1];
    s->status[1] = s->status[2];
    s->status[2] = x ^ (y << TINYMT32_SH1);
    s->status[3] = y;
    /*
    * The if (y & 1) {...} block below replaces:
    *     s->status[1] ^= -((int32_t)(y & 1)) & s->mat1;
    *     s->status[2] ^= -((int32_t)(y & 1)) & s->mat2;
    * The adopted code is equivalent to the original code
    * but does not depend on the representation of negative
    * integers by 2's complements. It is therefore more
    * portable, but includes an if-branch which may slow
    * down the generation speed.
    */
}

```

```

        */
        if (y & 1) {
            s->status[1] ^= s->mat1;
            s->status[2] ^= s->mat2;
        }
    }

/**
 * This function outputs a 32-bit unsigned integer from
 * the internal state.
 * @param s      pointer to tinymt internal state.
 * @return       32-bit unsigned pseudo-random number.
 */
static uint32_t tinymt32_temper (tinymt32_t * s)
{
    uint32_t t0, t1;
    t0 = s->status[3];
    t1 = s->status[0] + (s->status[2] >> TINYMT32_SH8);
    t0 ^= t1;
    t0 ^= -(int32_t)(t1 & 1) & s->tmat;
    return t0;
}
<CODE ENDS>

```

Figure 1: TinyMT32 Reference Implementation

### 3.2. TinyMT32 Usage

This PRNG MUST first be initialized with the following function:

```
void tinymt32_init (tinymt32_t * s, uint32_t seed);
```

It takes as input a 32-bit unsigned integer used as a seed (note that value 0 is authorized by TinyMT32). This function also takes as input a pointer to an instance of a `tinymt32_t` structure that needs to be allocated by the caller but left uninitialized. This structure will then be updated by the various TinyMT32 functions in order to keep the internal state of the PRNG. The use of this structure authorizes several instances of this PRNG to be used in parallel, each of them having its own instance of the structure.

Then, each time a new 32-bit pseudo-random unsigned integer between 0 and  $2^{32} - 1$  inclusive is needed, the following function is used:

```
uint32_t tinymt32_generate_uint32 (tinymt32_t * s);
```

Of course, the `tinymt32_t` structure must be left unchanged by the caller between successive calls to this function.



### 3.3. Specific Implementation Validation and Deterministic Behavior

PRNG determinism, for a given seed, can be a requirement (e.g., with [RLC-ID]). Consequently, any implementation of the TinyMT32 PRNG in line with this specification MUST comply with the following criteria. Using a seed value of 1, the first 50 values returned by `tinymt32_generate_uint32(s)` as 32-bit unsigned integers MUST be equal to values provided in Figure 2. Note that these values come from the `tinymt/check32.out.txt` file provided by the PRNG authors to validate implementations of TinyMT32, as part of the MersenneTwister-Lab/TinyMT Github repository.

```
2545341989 981918433 3715302833 2387538352 3591001365
3820442102 2114400566 2196103051 2783359912 764534509
 643179475 1822416315 881558334 4207026366 3690273640
3240535687 2921447122 3984931427 4092394160 44209675
2188315343 2908663843 1834519336 3774670961 3019990707
4065554902 1239765502 4035716197 3412127188 552822483
 161364450 353727785 140085994 149132008 2547770827
4064042525 4078297538 2057335507 622384752 2041665899
2193913817 1080849512 33160901 662956935 642999063
3384709977 1723175122 3866752252 521822317 2292524454
```

Figure 2: First 50 decimal values returned by `tinymt32_generate_uint32(s)` as 32-bit unsigned integers, with a seed value of 1.

In particular, the deterministic behavior of the Figure 1 source code has been checked across several platforms: high-end laptops running 64-bits Mac OSX and Linux/Ubuntu; a board featuring a 32-bits ARM Cortex-A15 and running 32-bit Linux/Ubuntu; several embedded cards featuring either an ARM Cortex-M0+, a Cortex-M3 or a Cortex-M4 32-bit microcontroller, all of them running RIOT [Baccelli18]; two low-end embedded cards featuring either a 16-bit microcontroller (TI MSP430) or a 8-bit microcontroller (Arduino ATMEGA2560), both of them running RIOT.

This specification only outputs 32-bit unsigned pseudo-random numbers and does not try to map this output to a smaller integer range (e.g., between 10 and 49 inclusive). If a specific use-case needs such a mapping, it will have to provide its own function. In that case, if PRNG determinism is also required, the use of floating point (single or double precision) to perform this mapping should probably be avoided, these calculations leading potentially to different rounding errors across different target platforms. Great care should also be put on not introducing biases in the randomness of the mapped output (it may be the case with some mapping algorithms) incompatible with

the use-case requirements. The details of how to perform such a mapping are out-of-scope of this document.

#### 4. Security Considerations

The authors do not believe the present specification generates specific security risks per se.

#### 5. IANA Considerations

This document does not require any IANA action.

#### 6. Acknowledgments

The authors would like to thank Belkacem Teibi with whom we explored TinyMT32 specificities when looking to an alternative to the Park-Miller Linear Congruential PRNG. The authors would like to thank the three TSVWG chairs, Wesley Eddy, our shepherd, David Black and Gorry Fairhurst, as well as Spencer Dawkins and Mirja Kuhlewind. Last but not least, the authors are really grateful to the IESG members, in particular Benjamin Kaduk, Eric Rescorla, and Adam Roach for their highly valuable feedbacks that greatly contributed to improve this specification.

#### 7. References

##### 7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

##### 7.2. Informative References

- [Baccelli18] Baccelli, E., Gundogan, C., Hahm, O., Kietzmann, P., Lenders, M., Petersen, H., Schleiser, K., Schmidt, T., and M. Wahlisch, "RIOT: An Open Source Operating System for Low-End Embedded Devices in the IoT", IEEE Internet of Things Journal (Volume 5, Issue 6), DOI: 10.1109/JIOT.2018.2815038, December 2018.

- [KR12] Rikitake, K., "TinyMT Pseudo Random Number Generator for Erlang", ACM 11th SIGPLAN Erlang Workshop (Erlang'12), September 14, 2012, Copenhagen, Denmark, DOI: <http://dx.doi.org/10.1145/2364489.2364504>, September 2012.
- [RFC5170] Roca, V., Neumann, C., and D. Furodet, "Low Density Parity Check (LDPC) Staircase and Triangle Forward Error Correction (FEC) Schemes", RFC 5170, DOI 10.17487/RFC5170, June 2008, <<https://www.rfc-editor.org/info/rfc5170>>.
- [RLC-ID] Roca, V. and B. Teibi, "Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC) Scheme for FECFRAME", Work in Progress, Transport Area Working Group (TSVWG) draft-ietf-tsvwg-rlc-fec-scheme (Work in Progress), February 2019, <<https://tools.ietf.org/html/draft-ietf-tsvwg-rlc-fec-scheme>>.

## Authors' Addresses

Mutsuo Saito  
Hiroshima University  
Japan

EEmail: [saito@math.sci.hiroshima-u.ac.jp](mailto:saito@math.sci.hiroshima-u.ac.jp)

Makoto Matsumoto  
Hiroshima University  
Japan

EEmail: [m-mat@math.sci.hiroshima-u.ac.jp](mailto:m-mat@math.sci.hiroshima-u.ac.jp)

Vincent Roca  
INRIA  
Univ. Grenoble Alpes  
France

EEmail: [vincent.roca@inria.fr](mailto:vincent.roca@inria.fr)

Emmanuel Baccelli  
INRIA  
France

EEmail: [emmanuel.baccelli@inria.fr](mailto:emmanuel.baccelli@inria.fr)

Transport Area Working Group  
Internet-Draft  
Intended status: Informational  
Expires: September 12, 2019

G. White  
K. Sundaresan  
B. Briscoe  
CableLabs  
March 11, 2019

Low Latency DOCSIS - Technology Overview  
draft-white-tsvwg-lld-00

Abstract

NOTE: This document is a reformatted version of [LLD-white-paper].

The evolution of the bandwidth capabilities - from kilobits per second to gigabits - across generations of DOCSIS cable broadband technology has paved the way for the applications that today form our digital lives. Along with increased bandwidth, or "speed", the latency performance of DOCSIS technology has also improved in recent years. Although it often gets less attention, latency performance contributes as much or more to the broadband experience and the feasibility of future applications as does speed.

Low Latency DOCSIS technology (LLD) is a specification developed by CableLabs in collaboration with DOCSIS vendors and cable operators that tackles the two main causes of latency in the network: queuing delay and media acquisition delay. LLD introduces an approach wherein data traffic from applications that aren't causing latency can take a different logical path through the DOCSIS network without getting hung up behind data from applications that are causing latency, as is the case in today's Internet architectures. This mechanism doesn't interfere with the way applications share the total bandwidth of the connection, and it doesn't reduce one application's latency at the expense of others. In addition, LLD improves the DOCSIS upstream media acquisition delay with a faster request-grant loop and a new proactive scheduling mechanism. LLD makes the internet experience better for latency sensitive applications without any negative impact on other applications.

The latest generation of DOCSIS equipment that has been deployed in the field - DOCSIS 3.1 - experiences typical latency performance of around 10 milliseconds (ms) on the Access Network link. However, under heavy load, the link can experience delay spikes of 100 ms or more. LLD systems can deliver a consistent 1 ms delay on the DOCSIS network for traffic that isn't causing latency, imperceptible for nearly all applications. The experience will be more consistent with much smaller delay variation.

LLD can be deployed by field-upgrading DOCSIS 3.1 cable modem and cable modem termination system devices with new software. The technology includes tools that enable automatic provisioning of these new services, and it also introduces new tools to report statistics of latency performance to the operator.

Cable operators, DOCSIS equipment manufacturers, and application providers will all have to act in order to take advantage of LLD. This white paper explains the technology and describes the role that each of these parties plays in making LLD a reality.

#### Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 12, 2019.

#### Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Latency in DOCSIS Networks . . . . .	4
3. New Dual-Queue Approach . . . . .	7
3.1. Low-Latency Aggregate Service Flows . . . . .	8
3.2. Identifying NQB Packets – Default Classifiers . . . . .	9
3.3. Coupled AQM . . . . .	10
3.4. Queue Protection . . . . .	11
4. Upstream Scheduling Improvements . . . . .	12
4.1. Faster Request Grant Loop . . . . .	12
4.2. Proactive Grant Service . . . . .	13
5. Low Latency DOCSIS Performance . . . . .	13
6. Deployment Considerations . . . . .	16
6.1. Device Support . . . . .	16
6.2. Packet Marking . . . . .	17
6.3. Provisioning Mechanisms . . . . .	18
6.3.1. Aggregate QoS Profiles . . . . .	18
6.3.2. Migration Using Existing Configuration File and Service Class Name . . . . .	18
6.3.3. Explicit Definition of ASF in the Configuration File . . . . .	19
6.4. Latency Histogram Reporting . . . . .	19
7. Conclusion . . . . .	19
8. Acknowledgements . . . . .	20
9. IANA Considerations . . . . .	20
10. Security Considerations . . . . .	20
11. Informative References . . . . .	20
Appendix A. Low Latency and High Bandwidth: L4S . . . . .	22
Appendix B. Simulation Details . . . . .	24
Authors' Addresses . . . . .	24

## 1. Introduction

Let's begin with bandwidth (or "speed"): the amount of data that can be delivered across a network connection over a period of time. Sometimes bandwidth is very important to the broadband experience, particularly when an application is trying to send or receive large amounts of data, such as watching videos on Netflix, downloading videos/music, syncing file-shares or email clients, uploading a video to YouTube or Instagram, or downloading a new application or system update. Other times, bandwidth (or bandwidth alone) isn't enough, and latency has a big effect on the user experience.

Latency is the time that it takes for a short message (a packet, in networking terminology) to make it across the network from the sender to the receiver and for a response to come back. Network latency is commonly measured as round-trip-time and is sometimes referred to as "ping time." Applications that are more interactive or real-time,

like web browsing, online gaming, and video conferencing/chatting, perform the best when latency is kept low, and adding more bandwidth without addressing latency doesn't make things better.

When multiple applications share the broadband connection of one household (e.g., several users doing different activities at the same time), each of those applications can have an impact on the performance of the others. They all share the total bandwidth of the connection (so more active applications mean less bandwidth for each one), and they can all cause the latency of the connection to increase.

It turns out that applications today that want to send a lot of data all at once do a reasonably good job of sharing the bandwidth in a fair manner, but they actually cause a pretty big latency problem when they do it because they send data too quickly and expect the network to queue it up. We call these applications "queue-building" applications, e.g., video streaming (Netflix). There are also plenty of other applications that don't send data too quickly, so they don't cause latency. We call these "non-queue-building" applications, e.g., video chatting (FaceTime).

LLD separates these two types of traffic into two logical queues, which greatly improves the latency experienced by the non-queue-building applications (many of which may be latency-sensitive) without having any downside for the queue-building applications. In addition, two queues allow LLD to support a next-generation application protocol that can scale up to sending data at 10 Gbps and beyond while maintaining ultra-low queuing delay, which means that in the future, there may not be queue-building applications at all.

As of the writing of this document, the Low Latency DOCSIS specifications have just been published ([DOCSIS-MULPIv3.1], [DOCSIS-CCAP-OSSIV3.1], [DOCSIS-CM-OSSIV3.1]), and DOCSIS equipment manufacturers are working on building support for the functionality. In addition, work is underway in the Internet Engineering Task Force to standardize low-latency architectures across the broader Internet ecosystem.

## 2. Latency in DOCSIS Networks

Low Latency DOCSIS technology is the next step in a progression of latency improvements that have been made to the DOCSIS specifications by CableLabs in recent years. Table 1 provides a snapshot of the milestones in round-trip latency performance with DOCSIS technology from the first DOCSIS 3.0 equipment to DOCSIS 3.1 equipment that supports [RFC8034] Active Queue Management, and finally the new Low Latency DOCSIS, which achieves ~1 ms of round-trip latency. The

table references three metrics that describe the range of latencies added by the DOCSIS network link that would be experienced by a broadband user. The first, "When Idle," refers to a broadband connection that is not being actively used by the customer. The second, "Under Load," represents average latency while the user is actively using the service (e.g., streaming video). Finally, the third, "99th Percentile," gives an indication of the maximum latency that a customer would commonly experience in real usage scenarios. The table uses order-of-magnitude numbers because the actual performance will vary because of a number of factors including DOCSIS channel configuration and actual application usage pattern.

For latency-sensitive applications, the 99th percentile value has the most impact on user experience.

TABLE 1. EVOLUTION OF LATENCY PERFORMANCE IN DOCSIS NETWORKS (ROUND-TRIP TIME IN MILLISECONDS BETWEEN THE CM AND CMTS)

	When Idle	Under Load	99th Percentile
DOCSIS 3.0 Early Equipment	~10 ms	~1000 ms	~1000 ms
DOCSIS 3.0 w/ Buffer Control	~10 ms	~100 ms	~100 ms
DOCSIS 3.1 Active Queue Management	~10 ms	~10 ms	~100 ms
Low Latency DOCSIS 3.1	~1 ms	~1 ms	~1 ms

Table 1

The latency described in Table 1 is caused by a series of factors in the DOCSIS cable modem (CM) and cable modem termination system (CMTS). Figure 1 in [LLD-white-paper] illustrates the range of latencies caused by those factors in DOCSIS 3.1 networks.

The lowest two latency sources in Figure 1 in [LLD-white-paper] have minor impacts on overall latency.

The "Switching/Forwarding" delay represents the amount of time it takes for the CM and CMTS to make the decision to forward a packet. This has a very minor impact on overall latency.

The "Propagation" delay (the amount of time it takes for a signal to travel on the HFC plant) is set by the speed of light and the distance from CM to CMTS. Not much can be done to affect latency from this source.



Of the sources in Figure 1 in [LLD-white-paper], the top three significantly drive latency performance.

The range of the "Serialization/Encoding" delay comes from the upstream and downstream channel configuration options available to the operator. Some of these configurations provide significant robustness benefits at the expense of latency, whereas others may be less robust to noise but provide very low latency. The LLD specification does not modify the set of options available to the operator. Rather, operators should be encouraged to use the lowest latency channel configurations that they can, given the plant conditions.

The "Media Acquisition" delay is a result of the shared-medium scheduling currently provided by DOCSIS technology, in which the CMTS arbitrates access to the upstream channel via a request-grant mechanism.

The "Queuing" delay is mainly caused by the current TCP protocol and its variants. Applications today that need to seek out as much bandwidth as possible use a transport protocol like TCP (or the TCP-replacement known as QUIC), which uses a "congestion control" algorithm (such as Reno, Cubic, or BBR) to adjust to the available bandwidth at the bottleneck link through the network. Typically, this will be the last mile link – the DOCSIS link for cable customers – where the bandwidth available for each application often varies rapidly as the activity of all the devices in the household varies.

With today's congestion control algorithms, the sender ramps up the sending rate until it's sending data faster than the bottleneck link can support. Packets then start queuing in a buffer at the entrance to the link, i.e. the CM or CMTS. This queue of packets grows quickly until the device decides to discard some newly arriving packets, which triggers the sender to pause for a bit in order to allow the buffer to drain somewhat before resuming sending. This process is an inherent feature of the TCP family of Internet transport protocols, and it repeats over and over again until the file transfer completes. In doing so, it causes latency and packet loss for all of the traffic that shares the broadband link.

LLD tackles the two main causes of latency in the network: queuing delay and media acquisition delay.

- o LLD addresses Queueing Delay by allowing non-queue-building applications to avoid waiting behind the delays caused by the current TCP or its variants. At a high level, the low-latency architecture consists of a dual-queue approach that treats both queues as a single pool of bandwidth.

- o LLD cuts Media Acquisition Delay by using a faster request-grant loop and by adding support for a new proactive scheduler that can provide extremely low latency service.

In addition, LLD introduces detailed statistics on queueing delay via histogram calculations performed by the CM (for upstream) and CMTS (for downstream). Furthermore, CableLabs is working with a broad cross-section of stakeholders in the IETF to standardize an end-to-end service architecture that can leverage LLD to enable even high bandwidth TCP flows to achieve ultra-low queueing delay. This technology will be important for future, interactive high-data-rate applications like holographic light field experiences, as well as for enabling higher performance versions of today's applications like web and video conferencing.

The sections below describe these features in more detail.

### 3. New Dual-Queue Approach

Of all the features of LLD, the dual-queue mechanism has by far the greatest impact on round-trip latency and latency variation. The concept of the dual-queue approach is that the majority of the applications that use the internet can be divided into two categories:

- o Queue-Building Applications: These application traffic flows frequently send data faster than the path between sender and receiver can support. The most common instance of queue-building flows are flows that use the current TCP or QUIC protocols. As discussed above, these capacity-seeking protocols use a legacy congestion control algorithm that probes for available capacity on the path by sending data faster than the path can support and expecting the network to queue the excess data in internal buffers. The majority of traffic (by volume) today is queue-building. Some examples of queue-building applications are video streaming (e.g., Netflix, YouTube) and application downloads.
- o Non-Queue-Building Applications: These application traffic flows very rarely send data faster than the path can support. They come in two subcategories:
  - \* Today's self-limited, non-capacity-seeking apps, such as multiplayer online games and IP communication apps (such as Skype or FaceTime). These applications send data at a relatively low data rate and generally space their packets out in a manner that does not cause a queue to form in the network.

- \* Future capacity-seeking TCP/QUIC applications that adopt the new L4S congestion control algorithm (see Appendix A) and so can immediately respond to fast congestion signals sent by the network. These applications are still in development, as networks must first support L4S before applications are able to take advantage, but some prime candidates are web browsing, cloud VR, and interactive light field experiences.

Queue-building (QB) application flows are the source of queuing delay, and today's non-queue-building (NQB) apps typically suffer from the latency caused by the QB flows.

The purpose of the dual-queue mechanism is to segment queue-building traffic from non-queue-building traffic in a manner that can be readily implemented in DOCSIS 3.1 equipment and that doesn't alter the overall bandwidth of the broadband service.

By segmenting these two types of applications into separate queues, each can get optimal performance. The QB traffic can build a queue and achieve the necessary and expected throughput performance, and the NQB traffic can take advantage of the available lower latencies by avoiding the delay caused by the QB flows. It is important to note that this segmentation of traffic isn't for purposes of giving one class of traffic benefits at the expense of the other - it isn't a high-priority queue and a low-priority queue. Instead, each queue is optimized for the distinct features and requirements of the two classes of traffic, enabling increased functionality and adding value for the broadband user. This is smart network management at work.

### 3.1. Low-Latency Aggregate Service Flows

DOCSIS 3.1 equipment, like equipment built against earlier versions of the specification, supports a number of upstream and downstream Service Flows (SFs). These Service Flows are logical pipes that are defined by their configured Quality of Service (QoS) parameters (most commonly, the rate shaping parameters [MULPIv3.1] that specify the speed of user connections) and that carry a subset of the traffic to/from a particular CM, as specified by a set of packet classifiers configured by the operator. Traditionally, each Service Flow provides near-complete isolation of its traffic from the traffic transiting other Service Flows (those on the same CM as well as those on other CMs) - each Service Flow has its own buffer and queue and is scheduled independently by the CMTS.

Typically, the operator defines a service offering via the configuration of a single upstream Service Flow and a single downstream Service Flow with rate shaping enabled, and all of the user's traffic transits these two Service Flows.

The DOCSIS 3.1 specification already includes optional support in the CMTS for a mechanism to group any number of the Service Flows serving a particular CM. LLD leverages and extends this "Aggregate Service Flow" (ASF) feature to establish (and group) a pair of Service Flows in each direction specifically to enable low-latency services. One of the Service Flows in the pair (the "Low Latency Service Flow") will carry NQB traffic, and the other Service Flow (the "Classic Service Flow") will carry QB traffic. The Aggregate Service Flow is configured for the service's rate shaping setting, and the two constituent Service Flows inside the Aggregate have rate shaping disabled. The result is that the operator can configure the total aggregate rate of the service offering in each direction and does not have to configure (or even consider) how much of the user's traffic is likely to be NQB vs QB.

Figure 2 in [LLD-white-paper] illustrates an example configuration of broadband service as it might look in a current DOCSIS deployment, as well as how it would look with Low Latency DOCSIS. In the traditional configuration, there is a single downstream Service Flow with a rate of 100 Mbps and a single upstream Service Flow with a rate of 20 Mbps. In the LLD configuration, there is a single downstream Aggregate Service Flow with a rate of 100 Mbps, containing two individual Service Flows, one for Low Latency traffic and one for Classic traffic. Similarly, there is single upstream Aggregate Service Flow with a rate of 20 Mbps, containing two individual Service Flows for Low Latency and Classic traffic.

The CMTS will enforce the Aggregate "Max Sustained Traffic Rate" (AMSR), and the end-user's applications determine how much of the aggregate bandwidth they consume irrespective of which SF they use - just as they do today with a single DOCSIS SF.

As described later, Inter-Service-Flow scheduling is arranged to make the ASF function as a single pool of bandwidth.

### 3.2. Identifying NQB Packets - Default Classifiers

By default, the traffic within an Aggregate Service Flow is segmented into the two constituent Service Flows by a set of packet classifiers (see Figure 3 in [LLD-white-paper]) that examine the Differentiated Services (DiffServ) Field and the Explicit Congestion Notification (ECN) Field, which are standard elements of the IPv4/IPv6 header [RFC3168]. Specifically, packets with an NQB DiffServ value or an ECN field indicating either ECN Capable Transport 1 (ECT(1)) or Congestion Experienced (CE) will get mapped to the Low Latency Service Flow, and the rest of the traffic will get mapped to the Classic Service Flow.

As of the writing of this draft, it is proposed that the DiffServ value 0x2A be standardized in IETF/IANA to indicate NQB [I-D.white-tsvwg-nqb]. Certain existing DiffServ values may also be classified as NQB by default, such as Expedited Forwarding (EF).

The expectation is that non-queue-building traffic sources (applications) will either mark their packets with an NQB DiffServ value or support ECN.

Although the DiffServ Field is being used to indicate NQB behavior, that does not imply adoption of the Differentiated Services architecture as it is typically understood. In the traditional DiffServ architecture, applications indicate a desire for a particular treatment of their packets - often implemented as a priority level - which in essence conveys a value judgement as to the importance of that traffic relative to the traffic of other applications. Such an architecture can work just fine in a managed environment where all applications conform to a common view of their relative priority levels and so can be trusted to mark their packets appropriately. It fails, however, when applications need to send packets across trust boundaries between networks, where there would be no common view on their relative importance. As a result, the DiffServ architecture is often used within managed networks (corporate networks, campus networks, etc.) but is not used on the Internet.

LLD's usage of the DiffServ Field to indicate NQB sidesteps this fundamental problem by eliminating the subjective value judgement on the relative importance of applications. Instead, this usage of the DiffServ Field describes objectively verifiable behavior on the part of the application - that it will not build a queue. Therefore, networks can verify that the marking has been applied properly before a packet is allowed into the Low Latency Service Flow queue (see Section 3.4).

The ECN classifiers enable LLD's support of the IETF's Low Latency Low Loss Scalable throughput (L4S) service [I-D.ietf-tsvwg-ecn-l4s-id], which is an evolution of the original ECN facility to support applications needing both high bandwidth and low latency (see Appendix A).

### 3.3. Coupled AQM

To manage queuing delay, both the Low Latency Service Flow queue and the Classic Service Flow queue support Active Queue Management (AQM) (see Figure 4 in [LLD-white-paper]).

In the case of the Classic Service Flow, the queue implements the same state-of-the-art Active Queue Management techniques used in today's DOCSIS 3.1 networks. For upstream Classic Service Flows, the DOCSIS 3.1 specification mandates that the CM implement the DOCSIS-PIE (Proportional-Integral-Enhanced AQM Algorithm), which introduces packet drops at an appropriate rate to drive the queue delay to the default target value of 10 ms. For downstream Classic Service Flows, the AQM in the CMTS is still vendor specific.

In the case of the Low Latency Service Flow, the queue supports L4S congestion controllers by implementing an Immediate Active Queue Management algorithm that utilizes ECN marking instead of packet drops. By default, the algorithm does not mark the packet if the queuing delay is less than 0.475 milliseconds and always marks the packet if the delay is greater than 1 ms. Between those configurable values, the algorithm marks at a rate that ramps up from 0% to 100% over the range. In addition, per [I-D.ietf-tsvwg-aqm-dualq-coupled], the Immediate AQM in the Low Latency Queue is coupled to the Classic Queue AQM so that congestion in the Classic Queue will induce ECN marking in the Low Latency Queue that will act to balance the per-flow throughput across all of the flows in both queues. L4S congestion control and the role of the dual-queue-coupled-aqm in providing flow balance is described further in Appendix A.

To enable the Low Latency Queue to rapidly dequeue an arrived burst of traffic, the Inter-Service-Flow scheduler gives a higher weight to the Low Latency Queue than it does to the Classic Queue. The coupling to the Low Latency AQM counterbalances the weighted scheduler by making low-latency applications leave space for Classic traffic. This ensures that the weighted scheduler does not give priority over bandwidth, as a traditional weighted scheduler would.

### 3.4. Queue Protection

Because of the small buffer size of the Low Latency Queue, classic TCP flows or other queue-building flows would see poor performance (due to high packet loss) if they were to end up in the Low Latency Queue. In addition, they would destroy the latency performance for the non-queue-building flows, negating the primary benefits of LLD.

To prevent this situation, the packets that are classified to the Low Latency queue pass through a "Queue Protection" function (see Figure 5 in [LLD-white-paper]), which scores each flow's contribution to the growth of the queue. If the queue delay exceeds a threshold, the Queue Protection function identifies the flow or flows that have contributed most to the growth of the queue delay, and it redirects future packets from those flows to the Classic Service Flow. This

mechanism is performed objectively and statistically, without examining the identifiers or contents of the data being transmitted.

#### 4. Upstream Scheduling Improvements

The DOCSIS upstream Media Access Control (MAC) Layer uses a request-grant mechanism. When data to be transmitted arrive at the CM, a request message is sent from the CM to the CMTS. The CMTS schedules the individual transmission bursts for all the CMs and communicates this via a bandwidth allocation map (MAP) message. Each MAP message describes the upstream transmission opportunities (grants) for a time interval and is sent shortly before the interval to which it applies.

When a CM has data to send, it waits for a "contention request" transmission opportunity. During that opportunity, it sends a short request message indicating the amount of data it has to send. It then waits for a subsequent MAP message granting it a transmission opportunity in which to send its data. This time interval between the arrival of the packet at the CM and the time at which the data arrives at the CMTS on the upstream channel is known as the Request-Grant Delay (see Figure 6 in [LLD-white-paper]). In the absence of queuing delay, this delay is generally 2-8 ms.

##### 4.1. Faster Request Grant Loop

LLD lowers the request-grant delay by requiring support for a shorter MAP Interval and a shorter MAP Processing Time (see Figure 7 in [LLD-white-paper]).

The MAP interval is the amount of time that each MAP message describes. The MAP interval is also the time interval between consecutive MAP messages. Reducing the MAP interval means that the CMTS processes incoming requests more frequently, thus shortening the amount of time that a request might wait at the CMTS before being processed. A shorter MAP interval also means that grants are not scheduled as far into the future within each MAP message.

The MAP Processing Time is the amount of time the CMTS uses to perform its scheduling calculations. With a shorter MAP Processing Time, there is less delay between a request being received at the CMTS and the resulting grant being scheduled.

The LLD specification requires support for a nominal MAP interval of 1 ms or less for OFDMA upstream channels, in place of the 2-4 ms used previously. In certain configurations, a 1 ms MAP interval may introduce tradeoffs such as upstream and/or downstream inefficiency that will need to be weighed against the latency improvement.

#### 4.2. Proactive Grant Service

DOCSIS scheduling services are designed to customize the behavior of the request-grant process for particular traffic types. LLD introduces a new scheduling service called Proactive Grant Service (PGS), which can eliminate the request-grant loop entirely (see Figure 8 in [LLD-white-paper]).

In PGS, a CMTS proactively schedules a stream of grants to a Service Flow at a rate that is intended to match or exceed the instantaneous demand. In doing so, the vast majority of packets carried by the Service Flow can be transmitted without being delayed by the Request-Grant process. During periods when the CMTS estimates no demand for bandwidth for a particular PGS Service Flow, it can conserve bandwidth by providing periodic unicast request opportunities rather than a stream of grants.

The service parameters that are specific to PGS are Guaranteed Grant Interval (GGI), Guaranteed Grant Rate (GGR), and Guaranteed Request Interval (GRI). In addition, the traditional rate-shaping parameters, such as Maximum Sustained Traffic Rate and Peak Rate, serve as an upper bound on the grants that can be provided to a PGS Service Flow.

PGS can eliminate the delay caused by the Request-Grant loop, but it comes at the price of efficiency. Inevitably, the CMTS will not be able to exactly predict the instantaneous demand for the Service Flow, so it may overestimate the capacity needed. When the shared channel is fully utilized, this could reduce the capacity available to other Service Flows.

The PGS scheduling type may appear at first to be similar to an existing DOCSIS upstream scheduling type "UGS/AD." The main differences with PGS are that it sets a minimum floor on the level of granting (minimum grant spacing and minimum granted bandwidth) rather than setting a fixed grant pattern (fixed grant size and precise grant spacing), it supports the "Continuous Concatenation and Fragmentation" method of filling grants (where a contiguous sequence of bytes are dequeued to fill the grant, regardless of packet boundaries) rather than only carrying a single packet in each grant, and the CM is expected to continue to send Requests to the CMTS to inform it of packets that might be waiting in the queue.

#### 5. Low Latency DOCSIS Performance

CableLabs has developed a simulator using the NS3 platform (<<https://www.nsnam.org>>) in order to evaluate the performance of different aspects of LLD. The simulator models a DOCSIS 3.1 link



(OFDM/A channel types) between the CM and the CMTS and can be configured to enable or disable various components of the technology.

Because the latency performance of the service depends on the mix of applications in use by the customer, we have developed a set of 10 traffic mix scenarios that represent what we believe to be common busy-hour behaviors for a cable customer. All traffic mixes include two bidirectional UDP sessions that are modeled after online games, but they could also represent VoIP or video conferencing/chatting applications. One of the sessions has its packets marked as NQB and the other does not, allowing us to see the benefit that the low-latency queue provides.

In addition, each traffic mix has a set of other applications that create background load, as summarized in Table 2 (see Appendix B for details on the traffic types). All of this background load traffic utilizes the classic queue.

Some of these traffic mixes represent behaviors that may be very common for broadband users during busy hour, whereas others represent more extreme behaviors that users may occasionally engage in. When generating an overall view of the performance across all of the traffic mixes, we model the fact that they may not all be equally likely to occur by giving the more common mixes (1, 2, and 8) ten times the weight that we give to each of the other less common mixes.

TABLE2. BACKGROUND TRAFFIC MIXES

Traffic Mix 1	1 web user
Traffic Mix 2	1 web user, 1 video streaming user
Traffic Mix 3	1 web user, 1 FTP upstream
Traffic Mix 4	1 web user, 1 FTP downstream
Traffic Mix 5	1 web user, 1 FTP upstream and 1 FTP downstream
Traffic Mix 6	1 web user, 5 FTP upstream and 5 FTP downstream
Traffic Mix 7	1 web user, 5 FTP up, 5 FTP down, and 2 video streaming users
Traffic Mix 8	5 web users
Traffic Mix 9	16 TCP down (speedtest)
Traffic Mix 10	8 TCP up (speedtest)

Table 2

Table 3 summarizes the 99th percentile per-packet latency for the NQB-marked game traffic across all ten traffic mixes, as well as the weighted overall performance, for four different systems:

1. a legacy DOCSIS 3.1 system with AQM disabled, 2 ms MAP interval;
2. a legacy DOCSIS 3.1 system with AQM enabled, 2 ms MAP interval;
3. a Low Latency DOCSIS 3.1 system without PGS, 1 ms MAP interval;  
and
4. a Low Latency DOCSIS 3.1 system with PGS configured for 5 Mbps  
GGR, 1 ms MAP interval.

We include LLD with and without PGS because some network operators may wish to deploy LLD without the overhead that comes with PGS scheduling.

TABLE 3. 99TH PERCENTILE ROUND-TRIP LATENCY FOR NQB-MARKED TRAFFIC BETWEEN THE CM AND CMTS

	Legacy DOCSIS 3.1 with no AQM	Legacy DOCSIS 3.1 with AQM	Low Latency DOCSIS with no PGS	Low Latency DOCSIS with PGS
Traffic Mix 1	7.7 ms	7.7 ms	4.7 ms	0.9 ms
Traffic Mix 2	7.7 ms	7.7 ms	4.8 ms	0.9 ms
Traffic Mix 3	159.5 ms	36.6 ms	4.7 ms	0.9 ms
Traffic Mix 4	7.8 ms	7.9 ms	4.7 ms	0.9 ms
Traffic Mix 5	159.6 ms	57.4 ms	4.7 ms	0.9 ms
Traffic Mix 6	253.7 ms	96.7 ms	4.7 ms	0.9 ms
Traffic Mix 7	253.9 ms	74.7 ms	4.7 ms	0.9 ms
Traffic Mix 8	7.7 ms	7.7 ms	4.7 ms	0.9 ms
Traffic Mix 9	259.3 ms	52.1 ms	4.8 ms	0.9 ms
Traffic Mix 10	254.0 ms	34.1 ms	4.8 ms	0.9 ms
Weighted Overall P99	250.5 ms	32.4 ms	4.7 ms	0.9 ms

Table 3

As can be seen in this table, there are several traffic mixes (notably 1, 2, 4, and 8) for which the relatively light traffic load doesn't create the conditions for TCP to cause significant queuing delay, so even the "Legacy DOCSIS 3.1 with no AQM" system results in fairly low latency. However, in the heavier traffic mixes, the benefit of AQM can be seen and the benefit of the dual-queue mechanism in LLD becomes very apparent. By separating the NQB-marked traffic from the queue-building traffic, the NQB-marked traffic is isolated from the delay created by the TCP flows entirely, and very reliable low latency is achieved. The right-most system, which additionally implements PGS, can eliminate the request-grant delay for the NQB traffic and thereby drive the round-trip latency below 1 ms at 99th percentile.

Figure 9 in [LLD-white-paper] illustrates the weighted overall latency performance across all ten traffic mixes. The plot is a log-log complementary cumulative distribution function, with the y-axis labeled with the equivalent quantile values.

Focusing, for instance, on the horizontal through the 99th percentile (P99), it can be seen that LLD with PGS holds delay below 0.9 ms for 99% of packets. In contrast, a DOCSIS 3.1 network without AQM can only hold delay below 250 ms for 99% of packets. So, P99 delay is more than 250 times better with LLD. We therefore see that LLD will bring a consistent, low-latency, responsive quality to cable broadband performance and user experiences for NBQ traffic.

## 6. Deployment Considerations

### 6.1. Device Support

Deploying LLD in the MSO network can be accomplished via software-only upgrades to the existing DOCSIS 3.1 CMs and CMTSs. Table 4 shows which LLD features need implementation on the CM side, the CMTS side, or both. The Dual Queue feature in the upstream requires an upgrade to the CM as well as to the CMTS. The other features (Dual Queue in Downstream, Upstream Scheduling improvements) only require upgrades on the CMTS, so they can be deployed to CMs that don't support LLD (including DOCSIS 3.0 modems).

TABLE 4. DEVICE DEPENDENCIES FOR LLD FEATURES

LLD Feature	Downstream Latency Improvements - CMTS upgrade?	Downstream Latency Improvements - CM upgrade?	Upstream Latency Improvements - CMTS upgrade?	Upstream Latency Improvements - CM upgrade?
Dual Queue (ASF, Coupled AQM, QP)	Required	Not required	Required	Required
Upstream Scheduling (Faster Req-Grant Loop, PGS)	Not applicable	Not applicable	Required	Not required

Table 4

## 6.2. Packet Marking

The design of LLD takes the approach that applications are in the best position to determine which flows or which packets are non-queue-building. Thus, applications such as online games will be able to tag their packets with the NQB DiffServ value to indicate that they behave in a non-queue-building way, so that LLD will be able to classify them into the Low Latency Service Flow.

For these packet markings to be useful for the LLD classifiers, they will need to survive the journey from the application source to the CM or CMTS. In some cases, operators today clear the DiffServ Field in packets entering their network from an interconnecting network, which would prevent the markings making their way to the CMTS. This practice is presumably driven by the view that DiffServ Field usage is defined by each operator for use within its network, in which case preserving another network's markings has no value. As was described in Section 3.2, it is proposed that a single globally standard value be chosen to indicate NQB so that operators that intend to support LLD can ensure that this specific value traverses their inbound interconnects and their network and then arrives at the CMTS intact.

Although application marking is preferable, some network operators might want to provide immediate benefits to applications that behave in a non-queue-building way, in advance of application developers introducing support for NQB tagging. It might be possible to

repurpose the queue protection function to identify NQB behavior even if the packets are not tagged as NQB, e.g., by assuming that all non-TCP traffic is likely to be NQB and relying on queue protection to redirect the QB flows. This is currently an area of active research.

Further, it is possible that intermediary software or devices (either installed by the user or provided by the operator) could identify flows that are expected to be NQB and mark the packets on behalf of the application.

### 6.3. Provisioning Mechanisms

The LLD specifications include provisioning mechanisms to allow an MSO to deploy low-latency features with minimal operational impact. Figure 10 in [LLD-white-paper] shows all the pieces needed to build a low-latency service in the upstream and downstream direction. Although it is possible to define a Low Latency ASF, its constituent Classic and Low Latency SFs, and the associated classifiers explicitly in the CM's configuration file, a new feature known as the Aggregate QoS Profile can make this configuration automatic in many cases. Default classifiers will be created and default parameters for AQM and queue protection will be used, or any of these can be overridden by the operator as needed.

#### 6.3.1. Aggregate QoS Profiles

Similar to Service Class Names that are expanded by the CMTS into a set of QoS parameters for a Service Flow during the registration process, an operator can create an Aggregate QoS Profile (AQP) on the CMTS to describe the parameters of an Aggregate Service Flow, its constituent Service Flows, and the classifiers used to identify NQB traffic.

Just like with Service Class Names, the operator can also provide explicit values in the configuration file for any ASF or SF parameters that they wish to "override".

#### 6.3.2. Migration Using Existing Configuration File and Service Class Name

One very straightforward way to migrate to LLD configurations may not involve any changes to the CM configuration file. This method involves the automatic expansion of a Service Flow definition to a Low Latency ASF via the use of a Service Class Name and matching AQP definition.

When the CMTS sees a Service Class Name in a Service Flow definition from the CM's config file, if the CM indicates support for LLD, then

the CMTS will first use the Service Class Name as an AQP Name and look for a matching entry in the AQP Table. If it finds a matching entry, it will automatically expand the Service Flow into an ASF and two Service Flows.

This mechanism allows the operator to deploy LLD by simply updating the CMTS to support the feature and configuring AQP entries that match the Service Class Names in use in CM config files. Then, as CMs are updated over time to include support for LLD, they will automatically start being configured with a Low Latency ASF.

#### 6.3.3. Explicit Definition of ASF in the Configuration File

An operator can also encode a Low Latency ASF in a CM configuration file directly using an Aggregate Service Flow TLV (70 or 71). The ASF TLV could have an AQP Name that is used by the CMTS to look up a definition of the ASF in its AQP Table. It could also have ASF parameters that would explicitly define the ASF or would override the AQP parameters. A configuration could also have explicit individual Service Flow TLVs (24 or 25) that are linked to the ASF via the Aggregate Service Flow Reference TLV.

#### 6.4. Latency Histogram Reporting

As part of the AQM operation, CMs and CMTSs generate estimates of the queuing latency for the upstream and downstream Service Flows, respectively. The latency histogram reporting function exposes these estimates to the operator to provide information that can be utilized to characterize network performance, optimize configurations, or troubleshoot problems in the field.

This latency histogram reporting can be enabled via a configuration file setting or can be initiated by setting a MIB object on the device. The operator configures the bins of the histogram, and the CM or the CMTS logs the number of packets with recorded latencies into each of the bins. The CM implements histograms for upstream Service Flows, and the CMTS implements histograms for downstream Service Flows. (This function can be enabled even for Service Flows for which AQM is disabled.) The latency estimates from the AQM are represented in the form of a histogram as well as a maximum latency value. See Figure 11 in [LLD-white-paper].

#### 7. Conclusion

LLD enables a huge leap in latency performance and will improve the Internet experience overall. With LLD, online gaming will become more responsive and video chats will cease to be "choppy." This technology will enable a range of new applications that require real-

time interface between the cyber and physical worlds, such as vehicular communications and remote health care services.

To realize the benefits of LLD, a number of parties need to take action. DOCSIS equipment manufacturers will need to develop and integrate the LLD features into software updates for CMTSSs and CMs. Cable operators need to plan the roll-out of software updates and configurations to DOCSIS equipment and set up the network to support those services (e.g., carrying DiffServ/ECN markings through the network). Application and operating system vendors will need to adopt packet marking for NQB traffic and/or adopt the L4S congestion controller. Each element of the Internet ecosystem will make these decisions independently; the faster that all take the necessary steps, the more quickly the user experience will improve.

The cable industry has provisioned its network with substantial bandwidth and is poised to take another leap forward with its 10G networks. But more bandwidth is only part of the broadband performance story. Latency is becoming crucial to the evolution of broadband. That is why LLD is a cornerstone of cable's 10G future.

## 8. Acknowledgements

CableLabs would like to thank the participants of the Low Latency DOCSIS Working Group, representing ARRIS, Broadcom, Casa, Charter, Cisco, Comcast, Cox Communications, Huawei, Intel, Liberty Global, Nokia, Rogers, Shaw, Videotron

## 9. IANA Considerations

None

## 10. Security Considerations

TBD

## 11. Informative References

[DOCSIS-CCAP-OSSIV3.1]

Cable Television Laboratories, Inc., "DOCSIS 3.1 CCAP Operations Support System Interface Specification, CM-SP-CCAP-OSSIV3.1-I14-190121", January 21, 2019, <<https://specification-search.cablelabs.com/CM-SP-CCAP-OSSIV3.1>>.

- [DOCSIS-CM-OSSIV3.1]  
Cable Television Laboratories, Inc., "DOCSIS 3.1 Cable Modem Operations Support System Interface Specification, CM-SP-CM-OSSIV3.1-I14-190121", January 21, 2019, <<https://specification-search.cablelabs.com/CM-SP-CM-OSSIV3.1>>.
- [DOCSIS-MULPIV3.1]  
Cable Television Laboratories, Inc., "MAC and Upper Layer Protocols Interface Specification, CM-SP-MULPIV3.1-I17-190121", January 21, 2019, <<https://specification-search.cablelabs.com/CM-SP-MULPIV3.1>>.
- [I-D.ietf-tsvwg-aqm-dualq-coupled]  
Schepper, K., Briscoe, B., Bondarenko, O., and I. Tsang, "DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)", draft-ietf-tsvwg-aqm-dualq-coupled-08 (work in progress), November 2018.
- [I-D.ietf-tsvwg-ecn-l4s-id]  
Schepper, K. and B. Briscoe, "Identifying Modified Explicit Congestion Notification (ECN) Semantics for Ultra-Low Queuing Delay (L4S)", draft-ietf-tsvwg-ecn-l4s-id-05 (work in progress), November 2018.
- [I-D.ietf-tsvwg-l4s-arch]  
Briscoe, B., Schepper, K., and M. Bagnulo, "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture", draft-ietf-tsvwg-l4s-arch-03 (work in progress), October 2018.
- [I-D.white-tsvwg-nqb]  
White, G., "Identifying and Handling Non Queue Building Flows in a Bottleneck Link", draft-white-tsvwg-nqb-00 (work in progress), October 2018.
- [LLD-white-paper]  
White, G., Sundaresan, K., and B. Briscoe, "Low Latency DOCSIS: Technology Overview", February 2019, <<https://cablela.bs/low-latency-docsis-technology-overview-february-2019>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.



- [RFC8034] White, G. and R. Pan, "Active Queue Management (AQM) Based on Proportional Integral Controller Enhanced PIE) for Data-Over-Cable Service Interface Specifications (DOCSIS) Cable Modems", RFC 8034, DOI 10.17487/RFC8034, February 2017, <<https://www.rfc-editor.org/info/rfc8034>>.
- [RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <<https://www.rfc-editor.org/info/rfc8311>>.
- [web-user-model]  
3GPP, "3GPP2-TSGC5, HTTP, FTP and TCP models for 1xEV-DV simulations", 2001.

#### Appendix A. Low Latency and High Bandwidth: L4S

How can LLD support applications that want maximum speed, and low latency too? CableLabs is working with the Internet Engineering Task Force to make this a reality through a new technology called L4S: Low Latency Low Loss Scalable throughput [I-D.ietf-tsvwg-l4s-arch].

L4S improves many of today's applications (e.g., video chat, everything on the web), but it will also enable future applications that will need both high bandwidth and low delay, such as HD video conferencing, cloud-rendered interactive video, cloud-rendered virtual reality, augmented reality, remote presence with remote control, interactive light field experiences, and others yet to be invented.

L4S involves incremental changes to the congestion controller on the sender and to the AQM at the bottleneck. The key is to indicate congestion by marking packets using Explicit Congestion Notification (ECN) rather than discarding packets. L4S uses the 2-bit ECN field in the IP header (v4 or v6) and defines each marked packet to represent a lower strength of congestion signal [RFC8311] than the original ECN standard. All the benefits of L4S follow from that.

- o Low Latency: The sender's L4S congestion controller makes small but frequent rate adjustments dependent on the proportion of ECN marked packets, and the L4S AQM starts applying ECN-marks to packets at a very shallow buffer threshold. This means an L4S queue can ripple at the very bottom of the buffer with sub-millisecond queuing delay but still fully utilize the link. Small, frequent adjustments could not even be considered if packet discards were used instead of ECN - they would induce a prohibitively high loss level. Further, AQMs could not consider a

very shallow threshold if small adjustments were not used, as severe link under-utilization would result.

- o Low Loss: By definition, using ECN eliminates packet discard. In turn, that eliminates retransmission delays, which particularly impact the responsiveness of short web-like exchanges of data. Using ECN eliminates both the round-trip delay repairing a loss and the delay while detecting a loss. In addition, an L4S AQM can immediately signal queue growth using ECN, catching queue growth early. In contrast, classic AQMs hold back from discarding a packet for 100-200 ms because if a burst subsides of its own accord, a loss in itself could cause more harm than the good it would do as a signal to slow down. Furthermore, eliminating packet discard eliminates the collateral damage caused to flows that were not significantly contributing to congestion.
- o Scalable Throughput: Existing congestion control algorithms don't scale, so applications need to open many simultaneous connections to fully utilize today's broadband connections. An L4S congestion controller can rapidly ramp up its sending rate to match any link capacity. This is because L4S uses a "scalable congestion controller" that maintains the same frequency of control signals (2 ECN marks per round trip on average) regardless of flow rate. With classic congestion controllers, the faster they try to go, the longer they run blind without any control signals.

The technology behind L4S isn't new; it is based on a scalable congestion control called Data Center TCP (DCTCP) that is currently used in data centers to get very high throughputs with ultra-low delay and loss. What is new is the development of a way that scalable traffic can coexist with the existing TCP and QUIC traffic on the Internet - the key that unlocks a transition to L4S. Until now, DCTCP has been confined to data centers because it would starve any classic flows sharing a link.

Separation into two queues serves two purposes: (1) it isolates L4S flows from the queuing of classic TCP and QUIC and (2) it sends each type of traffic appropriately scaled congestion signals. This results in any number of application flows (of either type) all getting roughly equal bandwidth each, as if there were just one aggregate pool of bandwidth, with no division between the Service Flows.

The approach couples the levels of ECN and drop signaling, as shown in Figure 12 in [LLD-white-paper]. The packet rate of today's classic congestion controls conforms to the well-known square-root rule (on the left of the figure). So, the classic AQM applies a drop level to Classic traffic that is coupled to the square of the ECN

marking level being applied to Low Latency traffic. The squaring in the network counterbalances the square root at the sender, so the packet rates of the two types of flow turn out roughly the same.

Supporting L4S in LLD is relatively straightforward. All that is needed is to classify L4S flows into the Low Latency SF and support the logic in the Low Latency SF to perform immediate ECN marking of packets (see Section 3.2).

## Appendix B. Simulation Details

For the results reported in this paper, we set up the following network with 5 types of client devices behind the CM and a set of servers north of the CMTS. See Figure 13 in [LLD-white-paper]. The link delays shown are 1-way values. The DOCSIS link is configured in the most latency-efficient manner (short interleavers, small OFDMA frame sizes) and models a plant distance of 8 km. The service is configured with a Maximum Sustained Traffic Rate (rate limit) of 50 Mbps in the upstream direction and 200 Mbps in the downstream direction.

The upstream game traffic model involves normally distributed packet interarrival times ( $\mu=33$  ms,  $\sigma=3$  ms) and normally distributed packet sizes ( $\mu=110$  bytes,  $\sigma=20$  bytes) constrained to discard draws of packet size  $<32$  bytes or  $>188$  bytes. The downstream game traffic model involves normally distributed packet interarrival times ( $\mu=33$  ms,  $\sigma=5$  ms) and normally distributed packet sizes ( $\mu=432$  bytes,  $\sigma=20$  bytes) constrained to discard draws of packet size  $<32$  bytes or  $>832$  bytes.

The background load traffic is configured as follows. The web user is based on the 3GPP standardized web user model [web-user-model]. The video streaming model is an abstracted model of a Dynamic Adaptive Streaming over HTTP (DASH) streaming video user where the video stream is 6 Mbps and is implemented as a 3.75 MB file download every 5 seconds. Each FTP session involves the sender selecting a file size using a log-normal random variable ( $\mu=14.8$ ,  $\sigma=2.0$ , leading to a median file size of 2.7 MB), opening a TCP connection, sending the file, closing the TCP connection, then pausing for 100 ms before repeating the process. Although we refer to this model as an FTP model, the intention is that it models TCP usage across all applications other than web browsing and video streaming.

## Authors' Addresses

Greg White  
CableLabs  
858 Coal Creek Circle  
Louisville, CO 80027  
US

Email: g.white@cablelabs.com

Karthik Sundaresan  
CableLabs  
858 Coal Creek Circle  
Louisville, CO 80027  
US

Email: k.sundaresan@cablelabs.com

Bob Briscoe  
CableLabs  
UK

Email: b.briscoe-contractor@cablelabs.com

Transport Area Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: September 12, 2019

G. White  
CableLabs  
March 11, 2019

Identifying and Handling Non Queue Building Flows in a Bottleneck Link  
draft-white-tsvwg-nqb-01

Abstract

This draft proposes the definition of a standardized DSCP to identify Non-Queue-Building flows, along with a Per-Hop-Behavior (PHB) that provides a separate queue for such flows.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 12, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Requirements Language . . . . .	3
3. Non-Queue Building Flows . . . . .	3
4. Endpoint Marking and Queue Protection . . . . .	3
5. Non Queue Building PHB and DSCP . . . . .	4
6. End-to-end Support . . . . .	5
7. Relationship to L4S . . . . .	6
8. Comparison to Existing Approaches . . . . .	6
9. Acknowledgements . . . . .	7
10. IANA Considerations . . . . .	7
11. Security Considerations . . . . .	7
12. Informative References . . . . .	8
Author's Address . . . . .	9

## 1. Introduction

Residential broadband internet services are commonly configured with a single bottleneck link (the access network link) upon which the service definition is applied. The service definition, typically an upstream/downstream data rate tuple, is implemented as a configured pair of rate shapers that are applied to the user's traffic. In such networks, the quality of service that each application receives, and as a result, the quality of experience that it generates for the user is influenced by the characteristics of the access network link.

The vast majority of packets that are carried by residential broadband access networks are managed by an end-to-end congestion control algorithm, such as Reno, Cubic or BBR. These congestion control algorithms attempt to seek the available capacity of the end-to-end path (which in the case of residential broadband networks, can frequently be the access network link capacity), and in doing so generally overshoot the available capacity, causing a queue to build-up at the bottleneck link. This queue build up results in queuing delay that the application experiences as variable latency, and commonly results in packet loss as well.

In contrast to congestion-controlled applications, there are a variety of relatively low data rate applications that do not materially contribute to queueing delay, but are nonetheless subjected to it by sharing the same bottleneck link in the access network. Many of these applications may be sensitive to latency or latency variation, as well as packet loss, and thus produce a poor quality of experience in such conditions.

Active Queue Management (AQM) mechanisms (such as PIE [RFC8033], DOCSIS-PIE [RFC8034], or CoDel [RFC8289]) can improve the quality of

experience for latency sensitive applications, but there are practical limits to the amount of improvement that can be achieved without impacting the throughput of capacity-seeking applications.

This document considers differentiating between these two classes of traffic in bottleneck links in order that both classes can deliver exceptional quality of experience for their applications.

## 2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

## 3. Non-Queue Building Flows

There are many applications that send traffic at relatively low data rates and/or in a fairly smooth and consistent manner such that they are highly unlikely to exceed the available capacity of the network path between source and sink. Such applications are ideal candidates to be queued separately from the capacity-seeking applications that cause queue buildups and latency.

These Non-queue-building (NQB) flows are typically UDP flows, which send traffic at a lower data rate and don't seek the capacity of the link (examples: online games, voice chat, dns lookups). Here the data rate is essentially limited by the Application itself. In contrast, Queue-building (QB) flows include traffic which uses the Traditional TCP, QUIC, BBR or other TCP variants.

There are a lot of great examples of applications that fall very neatly into these two categories, but there are also application flows that may be in a gray area in between (e.g. they are NQB on high-speed links, but QB on low-speed links).

## 4. Endpoint Marking and Queue Protection

This memo proposes that application endpoints apply a marking, utilizing the Diffserv field of the IP header, to packets of NQB flows that could then be used by the network to differentiate between QB and NQB flows. It is important for such a marking to be universally agreed upon, rather than being locally defined by the network operator, such that applications could be written to apply the marking without regard to local network policies.

Some questions that arise when considering endpoint marking are: How can an application determine whether it is queue building or not, given that the sending application is generally not aware of the

available capacity of the path to the receiving endpoint? Even in cases where an application is aware of the capacity of the path, how can it be sure that the available capacity (considering other flows that may be sharing the path) would be sufficient to result in the application's traffic not causing a queue to form? In an unmanaged environment, how can networks trust endpoint marking, why wouldn't all applications mark their packets as NQB?

As an answer the last question, it would be worthwhile to note that the NQB designation and marking would be intended to convey verifiable traffic behavior, not needs or wants. Also, it would be important that incentives are aligned correctly, i.e. that there is a benefit to the application in marking its packets correctly, and no benefit for an application in intentionally mismarking its traffic. Thus, a useful property of nodes that support separate queues for NQB and QB flows would be that for NQB flows, the NQB queue provides better performance (considering latency, loss and throughput) than the QB queue; and for QB flows, the QB queue provides better performance (considering latency, loss and throughput) than the NQB queue.

Even so, it is possible that due to an implementation error or misconfiguration, a QB flow would end up getting mismarked as NQB, or vice versa. In the case of an NQB flow that isn't marked as NQB and ends up in the QB queue, it would only impact its own quality of service, and so it seems to be of lesser concern. However, a QB flow that is mismarked as NQB would cause queuing delays for all of the other flows that are sharing the NQB queue.

To prevent this situation from harming the performance of the real NQB flows, it would likely be valuable to support a "queue protection" function that could identify QB flows that are mismarked as NQB, and reclassify those flows/packets to the QB queue. This would benefit the reclassified flow by giving it access to a large buffer (and thus lower packet loss rate), and would benefit the actual NQB flows by preventing harm (increased latency variability) to them. Such a function should be implemented in an objective and verifiable manner, basing its decisions upon the behavior of the flow rather than on application-level constructs.

## 5. Non Queue Building PHB and DSCP

This section uses the DiffServ nomenclature of per-hop-behavior (PHB) to describe how a network node could provide better quality of service for NQB flows without reducing performance of QB flows.

A node supporting the NQB PHB MUST provide a queue for non-queue-building traffic separate from the queue used for queue-building



traffic. This queue SHOULD support a latency-based queue protection mechanism that is able to identify queue-building behavior in flows that are classified into the queue, and to redirect flows causing queue build-up to a different queue. One example algorithm can be found in Annex P of [DOCSIS-MULPIv3.1].

While there may be some similarities between the characteristics of NQB flows and flows marked with the Expedited Forwarding DSCP, the NQB PHB would differ from the Expedited Forwarding PHB in several important ways.

- o NQB traffic is not rate limited or rate policed. Rather, the NQB queue would be expected to support a latency-based queue protection mechanism that identifies NQB marked flows that are beginning to cause latency, and redirects packets from those flows to the queue for QB flows.
- o The node supporting the NQB PHB makes no guarantees on latency or data rate for NQB marked flows, but instead aims to provide sub-millisecond queuing delays for as many such marked flows as it can, and shed load when needed.
- o EF is commonly used exclusively for voice traffic, for which additional functions are applied, such as admission control, accounting, prioritized delivery, etc.

In networks that support the NQB PHB, it may be preferred to also include traffic marked EF (101110b) in the NQB queue. The choice of the 0x2A codepoint (101010b) for NQB would conveniently allow a node to select these two codepoints using a single mask pattern of 101x10b.

Additionally, WiFi routers and APs that support WiFi MultiMedia commonly use the upper three bits of the DiffServ Field to select the WiFi User Priority. In the case of the 0x2A codepoint, this would map to UP\_5 which is in the "Video" Access Category (one level above "Best Effort").

## 6. End-to-end Support

In contrast to the existing standard DSCPs, which are typically only enforced within a DiffServ Domain (e.g. an AS), this DSCP would be intended for end-to-end usage across the Internet. Some access network service providers bleach the Diffserv field on ingress into their network, and in some cases apply their own DSCP for internal usage. Access networks that support the NQB PHB would need to permit the NQB PHB to pass through this bleaching operation such that the PHB can be provided at the access network link.

## 7. Relationship to L4S

The dual-queue mechanism described in this draft is similar to, and is intended to be compatible with [I-D.ietf-tsvwg-l4s-arch].

## 8. Comparison to Existing Approaches

Traditional QoS mechanisms focus on prioritization in an attempt to achieve two goals, reduced latency for "latency-sensitive" traffic, and increased bandwidth availability for "important" applications. Applications are generally given priority in proportion to some combination of latency-sensitivity and importance.

Downsides to this approach include the difficulties in sorting out what priority level each application should get (making the value judgement as to latency-sensitivity and importance), associating packets to priority levels (lots of classifier state, or trusting endpoint markings and the value judgements that they convey), ensuring that high priority traffic doesn't starve lower priority traffic (admission control, weighted scheduling, etc. are possible solutions). This solution can work in a managed network, where the network operator can control the usage of the QoS mechanisms, but has not been adopted end-to-end across the internet.

Flow queueing approaches (such as fq\_codel [RFC8290]), on the other hand, achieve latency improvements by associating packets into "flow" queues and then prioritizing "sparse flows", i.e. packets that arrive to an empty flow queue. Flow queueing does not attempt to differentiate between flows on the basis of value (importance or latency-sensitivity), it simply gives preference to sparse flows, and tries to guarantee that the non-sparse flows all get an equal share of the remaining channel capacity and are interleaved with one another. As a result, fq mechanisms could be considered more appropriate for unmanaged environments and general internet traffic.

Downsides to this approach include loss of low latency performance due to the possibility of hash collisions (where a sparse flow shares a queue with a bulk data flow), complexity in managing a large number of queues in certain implementations, and the DRR scheduling, which enforces that each non-sparse flow gets an equal fraction of link bandwidth, causes problems with VPNs and other tunnels, exhibits poor behavior with less-aggressive CA algos, e.g. LEDBAT, and exhibits poor behavior with RMCAT CA algos. In effect the network element is making a decision as to what constitutes a flow, and then forcing all such flows to take equal bandwidth at every instant.

The Dual-queue approach achieves the main benefit of fq\_codel: latency improvement without value judgements, without the downsides.

The distinction between NQB flows and QB flows is similar to the distinction made between "sparse flow queues" and "non-sparse flow queues" in fq\_codel. In fq\_codel, a flow queue is considered sparse if it is drained completely by each packet transmission, and remains empty for at least one cycle of the round robin over the active flows (this is approximately equivalent to saying that it utilizes less than its fair share of capacity). While this definition is convenient to implement in fq\_codel, it isn't the only useful definition of sparse flows.

The Linux Heavy-Hitter Filter ([HHF]) qdisc and the Cisco Dynamic Packet Prioritization ([DPP]) feature both categorize application flows into "mice" and "elephants", and provide a separate queue that gives high priority to the "mice" flows. In both of these implementations, the definition of a mouse flow is one that falls below a defined number of bytes or packets (respectively). In essence, the first N bytes or packets of every new flow are queued separately, and given priority over other traffic. The HHF implementation defaults to using 128KB for N, whereas the DPP documentation discusses using 120 packets.

This approach is relatively simple to implement, but it is making the wrong distinction between flows. To illustrate, an hour-long 60 kbps multiplayer online gaming flow sending 60 packets per second would be classified as an elephant after the first 17 seconds using HFF or 2 seconds using DPP, whereas it should be considered as NQB for the entire duration.

## 9. Acknowledgements

TBD

## 10. IANA Considerations

This draft proposes the registration of a standardized DSCP = 0x2A to denote Non-Queue-Building behavior.

## 11. Security Considerations

There is no incentive for an application to mismark its packets as NQB (or vice versa). If a queue-building flow were to mark its packets as NQB, it could experience excessive packet loss (in the case that queue-protection is not supported by a node) or it could receive no benefit (in the case that queue-protection is supported). If a non-queue-building flow were to fail to mark its packets as NQB, it could suffer the latency and loss typical of sharing a queue with capacity seeking traffic.

The NQB signal is not integrity protected and could be flipped by an on-path attacker. This might negatively affect the QoS of the tampered flow.

## 12. Informative References

- [DOCSIS-MULPIv3.1] Cable Television Laboratories, Inc., "MAC and Upper Layer Protocols Interface Specification, CM-SP-MULPIv3.1-I17-190121", January 21, 2019, <<https://specification-search.cablelabs.com/CM-SP-MULPIv3.1>>.
- [DPP] Cisco, "Intelligent Buffer Management on Cisco Nexus 9000 Series Switches White Paper", June 2017, <<https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-738488.html>>.
- [HHF] Lam, T., "net-qdisc-hhf: Heavy-Hitter Filter (HHF) qdisc", December 2013, <<https://lwn.net/Articles/577208/>>.
- [I-D.ietf-tsvwg-l4s-arch] Briscoe, B., Schepper, K., and M. Bagnulo, "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture", draft-ietf-tsvwg-l4s-arch-03 (work in progress), October 2018.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC8033] Pan, R., Natarajan, P., Baker, F., and G. White, "Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem", RFC 8033, DOI 10.17487/RFC8033, February 2017, <<https://www.rfc-editor.org/info/rfc8033>>.
- [RFC8034] White, G. and R. Pan, "Active Queue Management (AQM) Based on Proportional Integral Controller Enhanced PIE) for Data-Over-Cable Service Interface Specifications (DOCSIS) Cable Modems", RFC 8034, DOI 10.17487/RFC8034, February 2017, <<https://www.rfc-editor.org/info/rfc8034>>.

- [RFC8289] Nichols, K., Jacobson, V., McGregor, A., Ed., and J. Iyengar, Ed., "Controlled Delay Active Queue Management", RFC 8289, DOI 10.17487/RFC8289, January 2018, <<https://www.rfc-editor.org/info/rfc8289>>.
- [RFC8290] Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm", RFC 8290, DOI 10.17487/RFC8290, January 2018, <<https://www.rfc-editor.org/info/rfc8290>>.

## Author's Address

Greg White  
CableLabs  
858 Coal Creek Circle  
Louisville, CO 80027  
US

Email: [g.white@cablelabs.com](mailto:g.white@cablelabs.com)