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 middleBoxes 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 middleBoxes 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

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."
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.


The basic format of a U-DCCP packet is:

```
+-----------------------------------+     Variable length
|     IP Header (IPv4 or IPv6)     |
+-----------------------------------+
|UDP like arranged DCCP ext. Header| 8 bytes \  
+-----------------------------------+  ) U-DCCP header
|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.
All generic header fields have the meaning specified in [RFC4340], updated by [RFC5596].

### 3.3. UDP header

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |           Dest Port           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data Offset  | CCVal | CsCov |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     |       |X|               |                               |
| Res | Type  |=|   Reserved    |  Sequence Number (high bits)  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Sequence Number (low bits)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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.

```
 0                   1                   2                   3
+---+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| U |          Source Port          |           Dest Port           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| D |          Length               |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   | Type  | CCVal |  Data Offset  |  Sequence Number (high bits)  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Sequence Number (low bits)                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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


Internet-Draft        DCCP - UDP header conversion             July 2019

(DCCP) Simultaneous-Open Technique to Facilitate NAT/
Middlebox Traversal", RFC 5596, DOI 10.17487/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,

Datagram Congestion Control Protocol UDP Encapsulation for
NAT Traversal", RFC 6773, DOI 10.17487/RFC6773, November

[ 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,

[ 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,

Authors’ Addresses

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

Email: Markus.Amend@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
DCCP Extensions for Multipath Operation with Multiple Addresses

draft-amend-tsvwg-multipath-dccp-02

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 failure.

Multipath DCCP provides the ability to simultaneously use multiple paths between peers. This document presents a set of extensions to traditional DCCP to support multipath operation. 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 January 9, 2020.
1. Introduction

Multipath DCCP (MP-DCCP) is a set of extensions to regular DCCP [RFC4340], which enables a transport connection to operate across multiple paths simultaneously. DCCP multipath operations 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.lhwzx-hybrid-access-network-architecture][I-D.mu-ley-network-based-bonding-hybrid-access]. It can be applied for load-balancing, seamless session handover and aggregation purposes (referred to as steering, switching and splitting in 3GPP terminology [TR23.793]).

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, reassembly of data, and termination of sessions.

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.

```
+-------------------------------+
|           Application         |
+---------------+            +-------------------------------+
|  Application  |            |            MP-DCCP            |
+---------------+            + - - - - - - - + - - - - - - +
|      DCCP     |            |Subflow (DCCP) |Subflow (DCCP) |
+---------------+            +-------------------------------+
|      IP       |            |       IP      |      IP       |
+---------------+            +-------------------------------+
Figure 1: Comparison of Standard DCCP and MP-DCCP Protocol Stacks
```

1.2. Terminology

[Tbd], could be similar to [RFC6824]

1.3. MP-DCCP Concept

```
Host A                  Host B
+-----------------------+-----------------------+
| Address A1            | Address B1            |
| (DCCP flow setup)     | (DCCP flow setup)     |
<------------------------>                        <------------------------>
| merge individual DCCP flows to one multipath connection |
Figure 2: Example MP-DCCP Usage Scenario
```
1.4. Differences from Multipath TCP

Multipath DCCP is similar to Multipath TCP [RFC6824], 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, mainly dominated by the basic protocols TCP and DCCP, the transport characteristics 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 contrary and does not guarantee to transmit any payload nor the order of delivery. Since this is mainly affecting the receiving endpoint of a TCP or DCCP communication, many similarities on sender side can be stated.

Both transport protocols share the 3-way initiation of a communication and both exploit a congestion control to adapt to path characteristics.
Table 1: TCP and DCCP protocol comparison

Consequently, the multipath features, shown in Table 2, are the same, supporting volatile paths, session handover and path aggregation capabilities. All of them profit by the existence of congestion control.
### 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. MP-TCP 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. In practice, it is assumed that some sort of re-assembly has to be applied, even if DCCP and the order of delivery is unreliable by nature. Such re-assembly mechanisms have to account for the fact that packet loss may occur for any of the DCCP subflows. Another issue is the packet reordering introduced when a DCCP communication is split across paths with disjoint latencies. In theory, applications using DCCP certainly have to deal with packet reordering, since DCCP has no mechanisms to prevent it. However, in practice, without any multipath extension, packet reordering can be assumed to be very limited. Therefore most services on top of DCCP are not expecting massive packet reordering and degrades their performance if it happens anyway.

The receiving process for MP-TCP is on the other hand a simple "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

[Tbd], could be similar to [RFC6824]

3. MP-DCCP Protocol

[Tbd], could be similar to [RFC6824]

[Tbd] On top it requires particular considerations for:

- The minimum PMTU of the individual paths must be selected to announce to the application. Changes of individual path PMTUs must be re-announced to the application if they are lower than the current announced PMTU.

- Overall sequencing for optional reassembly procedure

- Congestion control

- Robust MP-DCCP session establishment (no dependency on an initial path setup)

4. Security Considerations

[Tbd]

5. Interactions with Middleboxes

[Tbd], should mention standardized technologies like [RFC5597] or [RFC6773] and U-DCCP [I-D.amend-tsvwg-dccp-udp-header-conversion]

6. Acknowledgments

1. Notes

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

[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

8. Informative References

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

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

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


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

Amend, et al.  
Expires January 9, 2020
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.lhwxz-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.lhwxz-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:
- IP compatibility: A multiaccess framework shall be able to transport IP packets and not make any assumptions on which transport protocol is encapsulated.

- 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.

- Support for flexible re-ordering: A multiaccess framework should support flexible re-ordering of user traffic, including no re-ordering at all. This requirements is important to support low latency traffic, where the re-creation of packet order may negatively impact delivery latency.

- 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.

- 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.

- 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 QUICs 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.

<table>
<thead>
<tr>
<th>Device</th>
<th>Middlebox 1</th>
<th>Middlebox 2</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender</td>
<td>+-------------+</td>
<td>+-------------+</td>
<td>+-------------+</td>
</tr>
<tr>
<td></td>
<td>MP-DCCP entry</td>
<td>MP-DCCP exit</td>
<td>Receiver</td>
</tr>
</tbody>
</table>

Figure 2: Sender and receiver independent MP-DCCP
5. Conclusion

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

- Pure routing
- Inherent path estimation and measurement
- Imposes no constraints on reliability or in-order delivery of application PDUs
- Modular re-ordering
- Modular scheduling
- IP compatible
- 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.lhwxz-hybrid-access-network-architecture]
Leymann, N., Heidemann, C., Wasserman, M., Xue, L., and M.
Zhang, "Hybrid Access Network Architecture", draft-lhwxz-
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., See, 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.

Ed., "Diameter Base Protocol", RFC 6733,
DOI 10.17487/RFC6733, October 2012,

[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,

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
Abstract

This document describes a YANG model for Quality of Service (QoS) configuration and operational parameters.

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 6, 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
1. Introduction

This document defines a base YANG [RFC6020] [RFC7950] data module for Quality of Service (QoS) configuration parameters. Differentiated Services (DiffServ) module is an augmentation of the base QoS model. Remote Procedure Calls (RPC) or notification definition is not part of this document. QoS base modules define a basic building blocks to define a classifier, policy, action and target. The base modules have been augmented to include packet match fields and action parameters to define the DiffServ module. Queues and schedulers are stitched as part of diffserv policy itself or separate modules are defined for creating Queue policy and Scheduling policy. The DiffServ model is based on DiffServ architecture, and various references have been made to available standard architecture documents.
DiffServ is a preferred approach for network service providers to offer services to different customers based on their network Quality-of-Service (QoS) objectives. The traffic streams are differentiated based on DiffServ Code Points (DSCP) carried in the IP header of each packet. The DSCP markings are applied by upstream node or by the edge router on entry to the DiffServ network.

Editorial Note: (To be removed by RFC Editor)

This draft contains several placeholder values that need to be replaced with finalized values at the time of publication. Please apply the following replacements:

- "XXXX" --> the assigned RFC value for this draft both in this draft and in the YANG models under the revision statement.
- The "revision" date in model, in the format XXXX-XX-XX, needs to be updated with the date the draft gets approved.

The YANG modules in this document conform to the Network Management Datastore Architecture (NMDA) [RFC8342 [RFC8342]].

1.1. Tree Diagrams

Tree diagrams used in this document follow the notation defined in [RFC8340 [RFC8340]]

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.

3. QoS Model Design

A classifier consists of packets which may be grouped when a logical set of rules are applied on different packet header fields. The grouping may be based on different values or range of values of same packet header field, presence or absence of some values or range of values of a packet field or a combination thereof. The QoS classifier is defined in the ietf-qos-classifier module.

A classifier entry contains one or more packet conditioning functions. A packet conditioning function is typically based on direction of traffic and may drop, mark or delay network packets. A set of classifier entries with corresponding conditioning functions when arranged in order of priority represents a QoS policy. A QoS...
policy may contain one or more classifier entries. These are defined in ietf-qos-policy module.

Actions are configured in line with respect to the policy module. These include marking, dropping or shaping. Actions are defined in the ietf-qos-action module.

A meter qualifies if the traffic arrival rate is based on agreed upon rate and variability. A meter is modeled based on commonly used algorithms in industry, Single Rate Tri Color Marking (srTCM) [RFC2697] meter, Two Rate Tri Color Marking (trTCM) [RFC2698] meter, and Single Rate Two Color Marking meter. Different vendors can extend it with other types of meters as well.

4. DiffServ Model Design

DiffServ architecture [RFC3289] and [RFC2475] describe the architecture as a simple model where traffic entering a network is classified and possibly conditioned at the boundary of the network and assigned a different Behavior Aggregate (BA). Each BA is identified by a specific value of DSCP, and is used to select a Per Hop Behavior (PHB).

The packet classification policy identifies the subset of traffic which may receive a DiffServ by being conditioned or mapped. Packet classifiers select packets within a stream based on the content of some portion of the packet header. There are two types of classifiers, the BA classifier, and the Multi-Field (MF) classifier which selects packets based on a value which is combination of one or more header fields. In the ietf-diffserv module, this is realized by augmenting the QoS classification module.

Traffic conditioning includes metering, shaping and/or marking. A meter is used to measure the traffic against a given traffic profile. The traffic profile specifies the temporal property of the traffic. A packet that arrives is first determined to be in or out of the profile, which will result in the action of marked, dropped or shaped. This is realized in vendor specific modules based on the parameters defined in action module. The metering parameters are augmented to the QoS policy module when metering is defined inline, and to the metering template when metering profile is referred in policy module.

5. Modules Tree Structure

This document defines seven YANG modules – four QoS base modules, a scheduler policy module, a queuing policy module and one DiffServ module.
ietf-qos-classifier consists of classifier entries identified by a classifier entry name. Each entry MAY contain a list of filter entries. When no filter entry is present in a classifier entry, it matches all traffic.

module: ietf-qos-classifier
    +--rw classifiers
        +--rw classifier-entry* [classifier-entry-name]
            +--rw classifier-entry-name                string
            +--rw classifier-entry-descr?              string
            +--rw classifier-entry-filter-operation?   identityref
        +--rw filter-entry* [filter-type filter-logical-not]
            +--rw filter-type           identityref
            +--rw filter-logical-not    boolean

An ietf-qos-policy module contains list of policy objects identified by a policy name and policy type which MUST be provided. With different values of policy types, each vendor MAY define their own construct of policy for different QoS functionalities. Each vendor MAY augment classifier entry in a policy definition with a set of actions.

module: ietf-qos-policy
    +--rw policies
        +--rw policy-entry* [policy-name policy-type]
            +--rw policy-name         string
            +--rw policy-type         identityref
            +--rw policy-descr?       string
        +--rw classifier-entry* [classifier-entry-name]
            +--rw classifier-entry-name           string
            +--rw classifier-entry-inline?        boolean
            +--rw classifier-entry-filter-oper?   identityref
        +--rw filter-entry* [filter-type filter-logical-not]
            +--rw filter-type           identityref
            +--rw filter-logical-not    boolean
        +--rw classifier-action-entry-cfg* [action-type]
            +--rw action-type    identityref
            +--rw (action-cfg-params)?
module: ietf-qos-action
  +--rw meter-template
  |   +--rw meter-entry* [meter-name] [meter-template-support]?
  |   +--rw meter-name                  string
  |   +--rw (meter-type)?
  |   |  +--:(one-rate-two-color-meter-type)
  |   |   +--rw one-rate-two-color-meter
  |   |       +--rw committed-rate-value?    uint64
  |   |       +--rw committed-rate-unit?     identityref
  |   |       +--rw committed-burst-value?   uint64
  |   |       +--rw committed-burst-unit?    identityref
  |   |       +--rw conform-action
  |   |           +--rw conform-2color-meter-action-params* [conform-2color-meter-action-type]
  |   |           |   +--rw conform-2color-meter-action-type identityref
  |   |           |   +--rw (conform-2color-meter-action-val)?
  |   |           +--rw exceed-action
  |   |               +--rw exceed-2color-meter-action-params* [exceed-2color-meter-action-type]
  |   |               |   +--rw exceed-2color-meter-action-type identityref
  |   |               |   +--rw (exceed-2color-meter-action-val)?
  |   +--:(one-rate-tri-color-meter-type)
  |   +--rw one-rate-tri-color-meter
  |       +--rw committed-rate-value?    uint64
  |       +--rw committed-rate-unit?     identityref
  |       +--rw committed-burst-value?   uint64
  |       +--rw committed-burst-unit?    identityref
  |       +--rw excess-burst-value?      uint64
  |       +--rw excess-burst-unit?       identityref
  |       +--rw conform-action
  |           +--rw conform-3color-meter-action-params* [conform-3color-meter-action-type]
  |           |   +--rw conform-3color-meter-action-type identityref
  |           |   +--rw (conform-3color-meter-action-val)?
  |           +--rw exceed-action
  |               +--rw exceed-3color-meter-action-params* [exceed-3color-meter-action-type]
  |               |   +--rw exceed-3color-meter-action-type identityref
  |               |   +--rw (exceed-3color-meter-action-val)?
  |               +--rw violate-action
  |                   +--rw violate-3color-meter-action-params* [violate-3color-meter-action-type]
  |                   |   +--rw violate-3color-meter-action-type identityref
  |                   +--rw (violate-3color-meter-action-val)?
Classifier, metering and queuing counters are associated with a target.

module: ietf-qos-target
augment /if:interfaces/if:interface:
  +--rw qos-target-entry* [direction policy-type]
    +--rw direction identityref
    +--rw policy-type identityref
    +--rw policy-name string
DiffServ module augments QoS classifier module. Many of the YANG types defined in [RFC6991] are represented as leafs in the classifier module.

Metering and marking actions are realized by augmenting the QoS policy-module. Any queuing, AQM and scheduling actions are part of vendor specific augmentation. Statistics are realized by augmenting the QoS target module.

```yangle
module: ietf-diffserv
  augment /classifier:classifiers/classifier:classifier-entry +
    /classifier:filter-entry:
      +--rw (filter-param)?
        +--:(dscp)
          |  +--rw dscp-cfg* [dscp-min dscp-max]
          |     +--rw dscp-min inet:dscp
          |     +--rw dscp-max inet:dscp
          +--:(source-ipv4-address)
          |  +--rw source-ipv4-address-cfg* [source-ipv4-addr]
          |     +--rw source-ipv4-addr inet:ipv4-prefix
          +--:(destination-ipv4-address)
          |  +--rw destination-ipv4-address-cfg* [destination-ipv4-addr]
          |     +--rw destination-ipv4-addr inet:ipv4-prefix
          +--:(source-ipv6-address)
          |  +--rw source-ipv6-address-cfg* [source-ipv6-addr]
          |     +--rw source-ipv6-addr inet:ipv6-prefix
          +--:(destination-ipv6-address)
          |  +--rw destination-ipv6-address-cfg* [destination-ipv6-addr]
          |     +--rw destination-ipv6-addr inet:ipv6-prefix
          +--:(source-port)
          |  +--rw source-port-cfg* [source-port-min source-port-max]
          |     +--rw source-port-min inet:port-number
          |     +--rw source-port-max inet:port-number
          +--:(destination-port)
          |  +--rw destination-port-cfg*
          |     [destination-port-min destination-port-max]
          |     +--rw destination-port-min inet:port-number
          |     +--rw destination-port-max inet:port-number
          +--:(protocol)
          |  +--rw protocol-cfg* [protocol-min protocol-max]
          |     +--rw protocol-min uint8
          |     +--rw protocol-max uint8
          +--:(traffic-group)
          |  +--rw traffic-group-cfg
          |     +--rw traffic-group-name? string
```

augment /policy:policies/policy:policy-entry +
  /policy:classifier-entry/policy:filter-entry:
    +--rw (filter-params)?
++--:(dscp)
 |   ++--rw dscp-cfg* [dscp-min dscp-max]
 |     |     +--rw dscp-min    inet:dscp
 |     |     +--rw dscp-max    inet:dscp
++--:(source-ipv4-address)
 |   ++--rw source-ipv4-address-cfg* [source-ipv4-addr]
 |     |     +--rw source-ipv4-addr    inet:ipv4-prefix
++--:(destination-ipv4-address)
 |   ++--rw destination-ipv4-address-cfg* [destination-ipv4-addr]
 |     |     +--rw destination-ipv4-addr    inet:ipv4-prefix
++--:(source-ipv6-address)
 |   ++--rw source-ipv6-address-cfg* [source-ipv6-addr]
 |     |     +--rw source-ipv6-addr    inet:ipv6-prefix
++--:(destination-ipv6-address)
 |   ++--rw destination-ipv6-address-cfg* [destination-ipv6-addr]
 |     |     +--rw destination-ipv6-addr    inet:ipv6-prefix
++--:(source-port)
 |   ++--rw source-port-cfg* [source-port-min source-port-max]
 |     |     +--rw source-port-min    inet:port-number
 |     |     +--rw source-port-max    inet:port-number
++--:(destination-port)
 |   ++--rw destination-port-cfg* [destination-port-min destination-port-max]
 |     |     +--rw destination-port-min    inet:port-number
 |     |     +--rw destination-port-max    inet:port-number
++--:(protocol)
 |   ++--rw protocol-cfg* [protocol-min protocol-max]
 |     |     +--rw protocol-min    uint8
 |     |     +--rw protocol-max    uint8
++--:(traffic-group)
 |   ++--rw traffic-group-cfg
 |     |     ++--rw traffic-group-name?    string
augment /policy:policies/policy:policy-entry +
|   |   /policy:classifier-entry +
|   |   /policy:classifier-action-entry-cfg +
|   |   /policy:action-cfg-params:
++--:(dscp-marking)
 |   ++--rw dscp-cfg
 |     |     ++--rw dscp?    inet:dscp
++--:(meter-inline) {action:meter-inline-feature}?
 |   ++--rw (meter-type)?
 |     |     ++--:(one-rate-two-color-meter-type)
 |     |     |     ++--rw committed-rate-value?    uint64
 |     |     |     ++--rw committed-rate-unit?    identityref
 |     |     |     ++--rw committed-burst-value?    uint64
 |     |     |     ++--rw committed-burst-unit?    identityref
 |     |     |     ++--rw conform-action
Internet-Draft             YANG Model For QoS                  July 2019

|     |     |  +--rw conform-2color-meter-action-params*  [conform-2color-meter-action-type]
|     |     |    +--rw conform-2color-meter-action-type identityref
|     |     |    +--rw (conform-2color-meter-action-val)?
|     |     |    +--rw exceed-action  
|     |     |    |   +--rw exceed-2color-meter-action-params*  [exceed-2color-meter-action-type]
|     |     |    |   +--rw exceed-2color-meter-action-type identityref
|     |     |    |   +--rw (exceed-2color-meter-action-val)?
|     |     |    +--:(one-rate-tri-color-meter-type)
|     |     |    |   +--rw one-rate-tri-color-meter  
|     |     |    |   +--rw committed-rate-value? uint64
|     |     |    |   +--rw committed-rate-unit? identityref
|     |     |    |   +--rw committed-burst-value? uint64
|     |     |    |   +--rw committed-burst-unit? identityref
|     |     |    |   +--rw excess-burst-value? uint64
|     |     |    |   +--rw excess-burst-unit? identityref
|     |     |    |   +--rw conform-action  
|     |     |    |    +--rw conform-3color-meter-action-params*  [conform-3color-meter-action-type]
|     |     |    |    +--rw conform-3color-meter-action-type identityref
|     |     |    |    +--rw (conform-3color-meter-action-val)?
|     |     |    |    +--rw exceed-action  
|     |     |    |    |   +--rw exceed-3color-meter-action-params*  [exceed-3color-meter-action-type]
|     |     |    |    |   +--rw exceed-3color-meter-action-type identityref
|     |     |    |    |   +--rw (exceed-3color-meter-action-val)?
|     |     |    |    +--rw violate-action  
|     |     |    |    |   +--rw violate-3color-meter-action-params*  [violate-3color-meter-action-type]
|     |     |    |    |   +--rw violate-3color-meter-action-type identityref
|     |     |    |    |   +--rw (violate-3color-meter-action-val)?
|     |     |    +--:(two-rate-tri-color-meter-type)
|     |     |    |   +--rw two-rate-tri-color-meter  
|     |     |    |   +--rw committed-rate-value? uint64
|     |     |    |   +--rw committed-rate-unit? identityref
|     |     |    |   +--rw committed-burst-value? uint64
|     |     |    |   +--rw committed-burst-unit? identityref
|     |     |    |   +--rw peak-rate-value? uint64
|     |     |    |   +--rw peak-rate-unit? identityref
|     |     |    |   +--rw peak-burst-value? uint64
|     |     |    |   +--rw peak-burst-unit? identityref
|     |     |    |   +--rw conform-action

---:rw conform-3color-meter-action-params*
   [conform-3color-meter-action-type]
   | ---:rw conform-3color-meter-action-type
     identityref
   | ---:rw (conform-3color-meter-action-val)?
     exceed-action
     | ---:rw exceed-3color-meter-action-params*
       [exceed-3color-meter-action-type]
     | ---:rw exceed-3color-meter-action-type
       identityref
     | ---:rw (exceed-3color-meter-action-val)?
     violate-action
     | ---:rw violate-3color-meter-action-params*
       [violate-3color-meter-action-type]
     | ---:rw violate-3color-meter-action-type
       identityref
     | ---:rw (violate-3color-meter-action-val)?
   +--:(meter-reference) (action:meter-reference-feature)?
     ---:rw meter-reference-cfg
     | ---:rw meter-reference-name string
     | ---:rw meter-type identityref
   +--:(traffic-group-marking) (action:traffic-group-feature)?
     ---:rw traffic-group-cfg
     | ---:rw traffic-group? string
   +--:(child-policy) (action:child-policy-feature)?
     ---:rw child-policy-cfg (child-policy-feature)?
     | ---:rw policy-name? string
   +--:(count) (action:count-feature)?
     ---:rw count-cfg (count-feature)?
     | ---:rw count-action? empty
   +--:(named-count) (action:named-counter-feature)?
     ---:rw named-counter-cfg (named-counter-feature)?
     | ---:rw count-name-action? string
   +--:(queue-inline) (diffserv-queue-inline-support)?
     ---:rw queue-cfg
     | ---:rw priority-cfg
       | ---:rw priority-level? uint8
     ---:rw min-rate-cfg
     | ---:rw rate-value? uint64
     | ---:rw rate-unit? identityref
     ---:rw max-rate-cfg
     | ---:rw rate-value? uint64
     | ---:rw rate-unit? identityref
     | ---:rw burst-value? uint64
     | ---:rw burst-unit? identityref
     ---:rw algorithmic-drop-cfg
     | ---:rw (drop-algorithm)?
     +--:(tail-drop)
module: ietf-queue-policy

++-rw queue-template {queue-policy-support}?
  +--rw name?      string
  +--rw queue-cfg
    +--rw priority-cfg
      +--rw priority-level?   uint8
      +--rw min-rate-cfg
        +--rw rate-value?   uint64
        +--rw rate-unit?    identityref
      +--rw max-rate-cfg
        +--rw rate-value?   uint64
        +--rw rate-unit?    identityref
        +--rw burst-value?   uint64
        +--rw burst-unit?    identityref
    +--rw algorithmic-drop-cfg
      +--rw (drop-algorithm)?
        +--:(tail-drop)
          +--rw tail-drop-cfg
            +--rw tail-drop-alg?   empty

augment /policy:policies/policy:policy-entry +
  /policy:classifier-entry/policy:classifier-action-entry-cfg +
  /policy:action-cfg-params:

  +--:(queue-template-name)
    {queue-template-support,queue-policy-support}?
    +--rw queue-template-reference-cfg
    +--rw queue-template-name    string

++-:(queue-inline)
  {queue-inline-support,queue-policy-support}?
module: ietf-scheduler-policy
    augment /policy:policies/policy:policy-entry + 
    /policy:classifier-entry/policy:classifier-action-entry-cfg + 
    /policy:action-cfg-params:
        +--:(scheduler)
            +--rw scheduler-cfg
                +--rw min-rate-cfg
                    |   +--rw rate-value?   uint64
                    |   +--rw rate-unit?    identityref
                +--rw max-rate-cfg
                    |   +--rw rate-value?   uint64
                    |   +--rw rate-unit?    identityref
                    |   +--rw burst-value?   uint64
                    |   +--rw burst-unit?    identityref
                +--:(queue-policy-name)
                    +--rw queue-policy-name
                        +--rw queue-policy    string

6. Modules
6.1. IETF-QOS-CLASSIFIER

```yang
<CODE BEGINS>file "ietf-qos-classifier@2019-03-13.yang"
module ietf-qos-classifier {
    yang-version 1.1;
    prefix classifier;

    organization
        "IETF RTG (Routing Area) Working Group";
    contact
        "WG Web:   <http://tools.ietf.org/wg/rtgwg/>
         WG List:  <mailto:rtgwg@ietf.org>
         WG Chair: Chris Bowers
            <mailto:cbowers@juniper.net>
         WG Chair: Jeff Tantsura
            <mailto:jefftant.ietf@gmail.com>
         Editor:   Aseem Choudhary
            <mailto:asechoud@cisco.com>
         Editor:   Mahesh Jethanandani
            <mailto:mjethanandani@gmail.com>
         Editor:   Norm Strahle
            <mailto:nstrahle@juniper.net>";
    description
        "This module contains a collection of YANG definitions for
         configuring qos specification implementations.
         Copyright (c) 2019 IETF Trust and the persons identified as
         the document authors.  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 2019-03-13 {
        description
            "Latest revision of qos base classifier module";
        reference "RFC XXXX: YANG Model for QoS";
    }

    feature policy-inline-classifier-config {
        description
            "This feature allows classifier configuration
             directly under policy.";
    }
```
feature classifier-template-feature {
    description
        "This feature allows classifier as template configuration in a policy.";
}

feature match-any-filter-type-support {
    description
        "This feature allows classifier configuration directly under policy.";
}

identity filter-type {
    description
        "This is identity of base filter-type";
}

identity classifier-entry-filter-operation-type {
    description
        "Classifier entry filter logical operation";
}

identity match-all-filter {
    base classifier-entry-filter-operation-type;
    description
        "Classifier entry filter logical AND operation";
}

identity match-any-filter {
    base classifier-entry-filter-operation-type;
    if-feature "match-any-filter-type-support";
    description
        "Classifier entry filter logical OR operation";
}

grouping filters {
    description
        "Filters types in a Classifier entry";
    leaf filter-type {
        type identityref {
            base filter-type;
        }
        description
            "This leaf defines type of the filter";
    }
    leaf filter-logical-not {
        type boolean;
        description
            "This leaf defines logical not of the filter";
    }
}
This is logical-not operator for a filter. When true, it indicates filter looks for absence of a pattern defined by the filter.

```
""
```

grouping classifier-entry-generic-attr {
  description
    "Classifier generic attributes like name, description, operation type"
  ;
  leaf classifier-entry-name {
    type string;
    description
      "classifier entry name";
  }
  leaf classifier-entry-descr {
    type string;
    description
      "classifier entry description statement";
  }
  leaf classifier-entry-filter-operation {
    type identityref {
      base classifier-entry-filter-operation-type;
    }
    default "match-all-filter";
    description
      "Filters are applicable as match-any or match-all filters";
  }
}

grouping classifier-entry-inline-attr {
  description
    "attributes of inline classifier in a policy";
  leaf classifier-entry-inline {
    type boolean;
    default "false";
    description
      "Indication of inline classifier entry";
  }
  leaf classifier-entry-filter-oper {
    type identityref {
      base classifier-entry-filter-operation-type;
    }
    default "match-all-filter";
  }
}
description
   "Filters are applicable as match-any or match-all filters";
}
list filter-entry {
    if-feature "policy-inline-classifier-config";
    must " ../classifier-entry-inline = 'true' " {
        description
           "For inline filter configuration, inline attribute must
            be true";
    }
    key "filter-type filter-logical-not";
    uses filters;
    description
       "Filters configured inline in a policy";
}

container classifiers {
    if-feature "classifier-template-feature";
    description
       "list of classifier entry";
    list classifier-entry {
        key "classifier-entry-name";
        description
           "each classifier entry contains a list of filters";
        uses classifier-entry-generic-attr;
        list filter-entry {
            key "filter-type filter-logical-not";
            uses filters;
            description
               "Filter entry configuration";
        }
    }
}

<CODE ENDS>

6.2. IETF-QOS-POLICY

<CODE BEGINS>file "ietf-qos-policy@2019-03-13.yang"
module ietf-qos-policy {
    yang-version 1.1;
    prefix policy;
    import ietf-qos-classifier {
        prefix classifier;
        reference "RFC XXXX: YANG Model for QoS";
    }
}

This module contains a collection of YANG definitions for configuring qos specification implementations. Copyright (c) 2019 IETF Trust and the persons identified as the document authors. 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 2019-03-13 {

description
"Latest revision of qos policy";
reference "RFC XXXX: YANG Model for QoS";
}

identity policy-type {

description
"This base identity type defines policy-types";
}

grouping policy-generic-attr {

description
"Policy Attributes";
leaf policy-name {

type string;

description
"policy name";
}
leaf policy-type {

type identityref {

    base policy-type;
}
}
description
   "policy type";
}
leaf policy-descr {
    type string;
    description
       "policy description";
}
}
identity action-type {
    description
       "This base identity type defines action-types";
}

grouping classifier-action-entry-cfg {
    description
       "List of Configuration of classifier & associated actions";
    list classifier-action-entry-cfg {
        key "action-type";
        ordered-by user;
        description
           "Configuration of classifier & associated actions";
        leaf action-type {
            type identityref {
                base action-type;
            }
            description
               "This defines action type ";
        }
        choice action-cfg-params {
            description
               "Choice of action types";
        }
    }
}

container policies {
    description
       "list of policy templates";
    list policy-entry {
        key "policy-name policy-type";
        description
           "policy template";
        uses policy-generic-attr;
        list classifier-entry {
            key "classifier-entry-name";
            ordered-by user;
            description
               "Classifier entry configuration in a policy";
            leaf classifier-entry-name {
type string;
description
  "classifier entry name";
}
uses classifier:classifier-entry-inline-attr;
uses classifier-action-entry-cfg;
}
}
</CODE ENDS>

6.3. IETF-QOS-ACTION

<CODE BEGINS>file "ietf-qos-action@2019-03-13.yang"
module ietf-qos-action {
  yang-version 1.1;
  prefix action;
import ietf-inet-types {
  prefix inet;
  reference "RFC 6991: Common YANG Data Types";
}
import ietf-qos-policy {
  prefix policy;
  reference "RFC XXXX: YANG Model for QoS";
}organization "IETF RTG (Routing Area) Working Group";
contact
  "WG Web: <http://tools.ietf.org/wg/rtgwg/>
  WG List: <mailto:rtgwg@ietf.org>
  WG Chair: Chris Bowers
    <mailto:cbowers@juniper.net>
  WG Chair: Jeff Tantsura
    <mailto:jefftant.ietf@gmail.com>
  Editor: Aseem Choudhary
    <mailto:asechoud@cisco.com>
  Editor: Mahesh Jethanandani
    <mailto:mjethanandani@gmail.com>
  Editor: Norm Strahle
    <mailto:nstrahle@juniper.net>";
description
  "This module contains a collection of YANG definitions for
configuring qos specification implementations.
Copyright (c) 2019 IETF Trust and the persons identified as
the document authors. 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 2019-03-13 {
  description
    "Latest revision for qos actions";
  reference "RFC XXXX: YANG Model for QoS";
}

feature meter-template-support {
  description
    "This feature allows support of meter-template.";
}

feature meter-inline-feature {
  description
    "This feature allows support of meter-inline configuration.";
}

feature meter-reference-feature {
  description
    "This feature allows support of meter by reference
     configuration.";
}

feature queue-action-support {
  description
    "This feature allows support of queue action configuration
     in policy.";
}

feature scheduler-action-support {
  description
    "This feature allows support of scheduler configuration
     in policy.";
}

feature child-policy-feature {
  description
    "This feature allows configuration of hierarchical policy.";
}

feature count-feature {
  description
    "This feature allows action configuration to enable
     counter in a classifier";
}

feature named-counter-feature {
  description
    "This feature allows action configuration to enable
     named counter in a classifier";
}
feature traffic-group-feature {
    description
        "traffic-group action support";
}

feature burst-time-unit-support {
    description
        "This feature allows burst unit to be configured as time duration.";
}

identity rate-unit-type {
    description
        "base rate-unit type";
}

identity bits-per-second {
    base rate-unit-type;
    description
        "bits per second identity";
}

identity kilo-bits-per-second {
    base rate-unit-type;
    description
        "kilo bits per second identity";
}

identity mega-bits-per-second {
    base rate-unit-type;
    description
        "mega bits per second identity";
}

identity giga-bits-per-second {
    base rate-unit-type;
    description
        "mega bits per second identity";
}

identity percent {
    base rate-unit-type;
    description
        "percentage";
}

identity burst-unit-type {
    description
        "base burst-unit type";
}

identity bytes {
    base burst-unit-type;
    description
        "bytes";
}
identity kilo-bytes {
  base burst-unit-type;
  description
    "kilo bytes";
}

identity mega-bytes {
  base burst-unit-type;
  description
    "mega bytes";
}

identity millisecond {
  base burst-unit-type;
  if-feature burst-time-unit-support;
  description
    "milli seconds";
}

identity microsecond {
  base burst-unit-type;
  if-feature burst-time-unit-support;
  description
    "micro seconds";
}

identity dscp-marking {
  base policy:action-type;
  description
    "dscp marking action type";
}

identity meter-inline {
  base policy:action-type;
  if-feature meter-inline-feature;
  description
    "meter-inline action type";
}

identity meter-reference {
  base policy:action-type;
  if-feature meter-reference-feature;
  description
    "meter reference action type";
}

identity queue {
  base policy:action-type;
  if-feature queue-action-support;
  description
    "queue action type";
}

identity scheduler {
  base policy:action-type;
  if-feature scheduler-action-support;
description  
"scheduler action type";
}
identity discard {
    base policy:action-type;
    description  
    "discard action type";
}
identity child-policy {
    base policy:action-type;
    if-feature child-policy-feature;
    description  
    "child-policy action type";
}
identity count {
    base policy:action-type;
    if-feature count-feature;
    description  
    "count action type";
}
identity named-counter {
    base policy:action-type;
    if-feature named-counter-feature;
    description  
    "name counter action type";
}

identity meter-type {
    description  
    "This base identity type defines meter types";
}
identity one-rate-two-color-meter-type {
    base meter-type;
    description  
    "one rate two color meter type";
}
identity one-rate-tri-color-meter-type {
    base meter-type;
    description  
    "one rate three color meter type";
    reference  
    "RFC2697: A Single Rate Three Color Marker";
}
identity two-rate-tri-color-meter-type {
    base meter-type;
    description  
    "two rate three color meter action type";
    reference  
    "RFC2697: A Single Rate Three Color Marker";
}
"RFC2698: A Two Rate Three Color Marker";
)

identity drop-type {
    description
        "drop algorithm";
}

identity tail-drop {
    base drop-type;
    description
        "tail drop algorithm";
}

drop-type {
    identity conform-2color-meter-action-type {
        description
            "action type in a meter";
    }
    identity exceed-2color-meter-action-type {
        description
            "action type in a meter";
    }
    identity conform-3color-meter-action-type {
        description
            "action type in a meter";
    }
    identity exceed-3color-meter-action-type {
        description
            "action type in a meter";
    }
    identity violate-3color-meter-action-type {
        description
            "action type in a meter";
    }
}

grouping rate-value-unit {
    leaf rate-value {
        type uint64;
        description
            "rate value";
    }
    leaf rate-unit {
        type identityref {
            base rate-unit-type;
        }
        description
            "rate unit";
    }
    description
            "rate unit";
grouping burst {
  description
  "burst value and unit configuration";
  leaf burst-value {
    type uint64;
    description
    "burst value";
  }
  leaf burst-unit {
    type identityref {
      base burst-unit-type;
    }
    description
    "burst unit";
  }
}

grouping threshold {
  description
  "Threshold Parameters";
  container threshold {
    description
    "threshold";
    choice threshold-type {
      case size {
        leaf threshold-size {
          type uint64;
          units "bytes";
          description
          "Threshold size";
        }
      }
      case interval {
        leaf threshold-interval {
          type uint64;
          units "microsecond";
          description
          "Threshold interval";
        }
      }
    description
    "Choice of threshold type";
  }
}

grouping drop {
    container drop-cfg {
        leaf drop-action {
            type empty;
            description
            "always drop algorithm";
        }
        description
        "the drop action";
        description
        "always drop grouping";
    }
}

grouping queuelimit {
    container qlimit-thresh {
        uses threshold;
        description
        "the queue limit";
    }
    description
    "the queue limit beyond which queue will not hold any packet";
}

grouping conform-2color-meter-action-params {
    description
    "meter action parameters";
    list conform-2color-meter-action-params {
        key "conform-2color-meter-action-type";
        ordered-by user;
        description
        "Configuration of basic-meter & associated actions";
        leaf conform-2color-meter-action-type {
            type identityref {
                base conform-2color-meter-action-type;
            }
            description
            "meter action type";
        }
        choice conform-2color-meter-action-val {
            description
            "meter action based on choice of meter action type";
        }
    }
}

grouping exceed-2color-meter-action-params {
    description
}
"meter action parameters";
list exceed-2color-meter-action-params {
  key "exceed-2color-meter-action-type";
  ordered-by user;
  description "Configuration of basic-meter & associated actions";
  leaf exceed-2color-meter-action-type {
    type identityref {
      base exceed-2color-meter-action-type;
    } 
    description "meter action type";
  } 
  choice exceed-2color-meter-action-val {
    description " meter action based on choice of meter action type";
  } 
}

grouping conform-3color-meter-action-params {
  description "meter action parameters";
  list conform-3color-meter-action-params {
    key "conform-3color-meter-action-type";
    ordered-by user;
    description "Configuration of basic-meter & associated actions";
    leaf conform-3color-meter-action-type {
      type identityref {
        base conform-3color-meter-action-type;
      } 
      description "meter action type";
    } 
    choice conform-3color-meter-action-val {
      description " meter action based on choice of meter action type";
    } 
  } 
}

grouping exceed-3color-meter-action-params {
  description "meter action parameters";
  list exceed-3color-meter-action-params {
    key "exceed-3color-meter-action-type";
  } 
}
ordered-by user;
description
"Configuration of basic-meter & associated actions";
leaf exceed-3color-meter-action-type {
    type identityref {
        base exceed-3color-meter-action-type;
    }
description
"meter action type";
} choice exceed-3color-meter-action-val {
description
" meter action based on choice of meter action type";
}
}
grouping violate-3color-meter-action-params {
description
"meter action parameters";
list violate-3color-meter-action-params {
    key "violate-3color-meter-action-type";
    ordered-by user;
description
"Configuration of basic-meter & associated actions";
leaf violate-3color-meter-action-type {
    type identityref {
        base violate-3color-meter-action-type;
    }
description
"meter action type";
} choice violate-3color-meter-action-val {
description
" meter action based on choice of meter action type";
}
}
grouping one-rate-two-color-meter {
    container one-rate-two-color-meter {
        description
"single rate two color marker meter";
    leaf committed-rate-value {
        type uint64;
description
"committed rate value";
    }
    }
}
leaf committed-rate-unit {
    type identityref {
        base rate-unit-type;
    }
    description
    "committed rate unit";
}

leaf committed-burst-value {
    type uint64;
    description
    "burst value";
}

leaf committed-burst-unit {
    type identityref {
        base burst-unit-type;
    }
    description
    "committed burst unit";
}

container conform-action {
    uses conform-2color-meter-action-params;
    description
    "conform action";
}

container exceed-action {
    uses exceed-2color-meter-action-params;
    description
    "exceed action";
}

description
"single rate two color marker meter attributes";
}

grouping one-rate-tri-color-meter {
    container one-rate-tri-color-meter {
        description
        "single rate three color meter";
        reference
        "RFC2697: A Single Rate Three Color Marker";
        leaf committed-rate-value {
            type uint64;
            description
            "meter rate";
        }
        leaf committed-rate-unit {
            type identityref {
                base rate-unit-type;
            }
        }
    }
}
} description "committed rate unit";
}
leaf committed-burst-value {
    type uint64;
    description "committed burst size";
}
leaf committed-burst-unit {
    type identityref {
        base burst-unit-type;
    }
    description "committed burst unit";
}
leaf excess-burst-value {
    type uint64;
    description "excess burst size";
}
leaf excess-burst-unit {
    type identityref {
        base burst-unit-type;
    }
    description "excess burst unit";
}
container conform-action {
    uses conform-3color-meter-action-params;
    description "conform, or green action";
}
container exceed-action {
    uses exceed-3color-meter-action-params;
    description "exceed, or yellow action";
}
container violate-action {
    uses violate-3color-meter-action-params;
    description "violate, or red action";
}
}

description "one-rate-tri-color-meter attributes";
grouping two-rate-tri-color-meter {
  container two-rate-tri-color-meter {
    description
      "two rate three color meter";
    reference
      "RFC2698: A Two Rate Three Color Marker";
    leaf committed-rate-value {
      type uint64;
      units "bits-per-second";
      description
        "committed rate";
    }
    leaf committed-rate-unit {
      type identityref {
        base rate-unit-type;
      }
      description
        "committed rate unit";
    }
    leaf committed-burst-value {
      type uint64;
      description
        "committed burst size";
    }
    leaf committed-burst-unit {
      type identityref {
        base burst-unit-type;
      }
      description
        "committed burst unit";
    }
    leaf peak-rate-value {
      type uint64;
      description
        "peak rate";
    }
    leaf peak-rate-unit {
      type identityref {
        base rate-unit-type;
      }
      description
        "committed rate unit";
    }
    leaf peak-burst-value {
      type uint64;
      description
        "committed burst size";
    }
  }
}
leaf peak-burst-unit {
  type identityref {
    base burst-unit-type;
  }
  description
    "peak burst unit";
}
container conform-action {
  uses conform-3color-meter-action-params;
  description
    "conform, or green action";
}
container exceed-action {
  uses exceed-3color-meter-action-params;
  description
    "exceed, or yellow action";
}
container violate-action {
  uses violate-3color-meter-action-params;
  description
    "exceed, or red action";
}
}
description
  "two-rate-tri-color-meter attributes";
}

grouping meter {
  choice meter-type {
    case one-rate-two-color-meter-type {
      uses one-rate-two-color-meter;
      description
        "basic meter";
    }
    case one-rate-tri-color-meter-type {
      uses one-rate-tri-color-meter;
      description
        "one rate tri-color meter";
    }
    case two-rate-tri-color-meter-type {
      uses two-rate-tri-color-meter;
      description
        "two rate tri-color meter";
    }
    description
      "meter action based on choice of meter action type";
  }
  description
    "two-rate-tri-color-meter attributes";
}
"meter attributes";
}

container meter-template {
    description "list of meter templates";
    list meter-entry {
        if-feature meter-template-support;
        key "meter-name";
        description "meter entry template";
        leaf meter-name {
            type string;
            description "meter identifier";
            uses meter;
        }
    }
}

grouping meter-reference {
    container meter-reference-cfg {
        leaf meter-reference-name {
            type string;
            mandatory true;
            description "This leaf defines name of the meter referenced";
        }
        leaf meter-type {
            type identityref {
                base meter-type;
            }
            mandatory true;
            description "This leaf defines type of the meter";
        }
        description "meter reference name";
    }
    description "meter reference";
}

grouping count {
    container count-cfg {
        if-feature count-feature;
        leaf count-action {
            type empty;
        }
    }
}
description "count action";
}
}

description "the count action grouping";
}

grouping named-counter {
  container named-counter-cfg {
    if-feature named-counter-feature;
    leaf count-name-action {
      type string;
      description "count action";
    }
    description "the count action";
  }
  description "the count action grouping";
}

grouping discard {
  container discard-cfg {
    leaf discard {
      type empty;
      description "discard action";
    }
    description "discard action";
  }
  description "discard grouping";
}

grouping priority {
  container priority-cfg {
    leaf priority-level {
      type uint8;
      description "priority level";
    }
    description "priority attributes";
  }
  description "priority attributes";
}
grouping min-rate {
  container min-rate-cfg {
    uses rate-value-unit;
    description "min guaranteed bandwidth";
    reference "RFC3289, section 3.5.3";
  }
  description "minimum rate grouping";
}

grouping dscp-marking {
  container dscp-cfg {
    leaf dscp {
      type inet:dscp;
      description "dscp marking";
    }
    description "dscp marking container";
  }
  description "dscp marking grouping";
}

grouping traffic-group-marking {
  container traffic-group-cfg {
    leaf traffic-group {
      type string;
      description "traffic group marking";
    }
    description "traffic group marking container";
  }
  description "traffic group marking grouping";
}

grouping child-policy {
  container child-policy-cfg {
    if-feature child-policy-feature;
    leaf policy-name {
      type string;
      description "Hierarchical Policy";
    }
  }
  description "child policy grouping";
}


grouping max-rate {
  container max-rate-cfg {
    uses rate-value-unit;
    uses burst;
    description
      "maximum rate attributes container";
    reference
      "RFC3289, section 3.5.4";
  } 
  description
    "maximum rate attributes";
}
grouping queue {
  container queue-cfg {
    uses priority;
    uses min-rate;
    uses max-rate;
    container algorithmic-drop-cfg {
      choice drop-algorithm {
        case tail-drop {
          container tail-drop-cfg {
            leaf tail-drop-alg {
              type empty;
              description
                "tail drop algorithm";
            } 
            description
              "Tail Drop configuration container";
          } 
          description
            "Tail Drop choice";
        } 
        description
          "Choice of Drop Algorithm";
      } 
      description
        "Algorithmic Drop configuration container";
    } 
    description
      "Queue configuration container";
  }
description
  "Queue grouping";
}

module ietf-qos-target {
  yang-version 1.1;
  prefix target;

  import ietf-interfaces {
    prefix if;
    reference "RFC8343: A YANG Data Model for Interface Management";
  }

  import ietf-qos-policy {
    prefix policy;
    reference "RFC XXXX: YANG Model for QoS";
  }

  organization "IETF RTG (Routing Area) Working Group";
  contact
    "WG Web:  <http://tools.ietf.org/wg/rtgwg/>
    WG List:  <mailto:rtgwg@ietf.org>
    WG Chair: Chris Bowers
              <mailto:cbowers@juniper.net>
    WG Chair: Jeff Tantsura
              <mailto:jefftant.ietf@gmail.com>
    Editor:   Aseem Choudhary
              <mailto:asechoud@cisco.com>
    Editor:   Mahesh Jethanandani
              <mailto:mjethanandani@gmail.com>
    Editor:   Norm Strahle
              <mailto:nstrahle@juniper.net>
  description

  description
    "Scheduler configuration container";
}

description
  "Scheduler configuration grouping";
}
"This module contains a collection of YANG definitions for configuring qos specification implementations.
Copyright (c) 2019 IETF Trust and the persons identified as the document authors. 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 2019-03-13 {
  description
    "Latest revision qos based policy applied to a target";
  reference "RFC XXXX: YANG Model for QoS";
}

identity direction {
  description
    "This is identity of traffic direction";
}

identity inbound {
  base direction;
  description
    "Direction of traffic coming into the network entry";
}

identity outbound {
  base direction;
  description
    "Direction of traffic going out of the network entry";
}

augment "/if:interfaces/if:interface" {
  description
    "Augments Diffserv Target Entry to Interface module";
  list qos-target-entry {
    key "direction policy-type";
    description
      "policy target for inbound or outbound direction";
    leaf direction {
      type identityref {
        base direction;
      }
      description

"Direction of the traffic flow either inbound or outbound";
}
leaf policy-type {
  type identityref {
    base policy:policy-type;
  }
  description
    "Policy entry type";
}
leaf policy-name {
  type string;
  mandatory true;
  description
    "Policy entry name";
}
This module contains a collection of YANG definitions for configuring diffserv specification implementations.

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. 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 2019-03-13 {
    description
        "Latest revision of diffserv based classifier";
        reference "RFC XXXX: YANG Model for QoS";
}

feature diffserv-queue-inline-support {
    description
        "Queue inline support in diffserv policy";
}

feature diffserv-scheduler-inline-support {
    description
        "scheduler inline support in diffserv policy";
}

identity diffserv-policy-type {
    base policy:policy-type;
    description
        "This defines ip policy-type";
}

identity ipv4-diffserv-policy-type {
    base policy:policy-type;
    description
        "This defines ipv4 policy-type";
}
identity ipv6-diffserv-policy-type {
    base policy:policy-type;
    description
        "This defines ipv6 policy-type";
}

identity dscp {
    base classifier:filter-type;
    description
        "Differentiated services code point filter-type";
}

identity source-ipv4-address {
    base classifier:filter-type;
    description
        "source ipv4 address filter-type";
}

identity destination-ipv4-address {
    base classifier:filter-type;
    description
        "destination ipv4 address filter-type";
}

identity source-ipv6-address {
    base classifier:filter-type;
    description
        "source ipv6 address filter-type";
}

identity destination-ipv6-address {
    base classifier:filter-type;
    description
        "destination ipv6 address filter-type";
}

identity source-port {
    base classifier:filter-type;
    description
        "source port filter-type";
}

identity destination-port {
    base classifier:filter-type;
    description
        "destination port filter-type";
}

identity protocol {
    base classifier:filter-type;
    description
        "protocol type filter-type";
}

identity traffic-group-name {

base classifier:filter-type;
description
  "traffic-group filter type";
}

identity meter-type {
  description
    "This base identity type defines meter types";
}

identity one-rate-two-color-meter-type {
  base meter-type;
  description
    "one rate two color meter type";
}

identity one-rate-tri-color-meter-type {
  base meter-type;
  description
    "one rate three color meter type";
}

identity two-rate-tri-color-meter-type {
  base meter-type;
  description
    "two rate three color meter action type";
}

grouping dscp-cfg {
  list dscp-cfg {
    key "dscp-min dscp-max";
    description
      "list of dscp ranges";
    leaf dscp-min {
      type inet:dscp;
      description
        "Minimum value of dscp min-max range";
    }
    leaf dscp-max {
      type inet:dscp;
      description
        "maximum value of dscp min-max range";
    }
  }
  description
    "Filter grouping containing list of dscp ranges";
}

grouping source-ipv4-address-cfg {
  list source-ipv4-address-cfg {
    key "source-ipv4-addr";
    description
      "list of source ipv4 address";
  }
}
leaf source-ipv4-addr {
    type inet:ipv4-prefix;
    description "source ipv4 prefix";
}

description
"Filter grouping containing list of source ipv4 addresses";

grouping destination-ipv4-address-cfg {
    list destination-ipv4-address-cfg {
        key "destination-ipv4-addr";
        description "list of destination ipv4 address";
        leaf destination-ipv4-addr {
            type inet:ipv4-prefix;
            description "destination ipv4 prefix";
        }
    }

    description
"Filter grouping containing list of destination ipv4 address";
}

grouping source-ipv6-address-cfg {
    list source-ipv6-address-cfg {
        key "source-ipv6-addr";
        description "list of source ipv6 address";
        leaf source-ipv6-addr {
            type inet:ipv6-prefix;
            description "source ipv6 prefix";
        }
    }

    description
"Filter grouping containing list of source ipv6 addresses";
}

grouping destination-ipv6-address-cfg {
    list destination-ipv6-address-cfg {
        key "destination-ipv6-addr";
        description "list of destination ipv4 or ipv6 address";
        leaf destination-ipv6-addr {
            type inet:ipv6-prefix;
            description "destination ipv6 prefix";
        }
    }
}

description
  "Filter grouping containing list of destination ipv6 address";
}
grouping source-port-cfg {
  list source-port-cfg {
    key "source-port-min source-port-max";
    description
      "list of ranges of source port";
    leaf source-port-min {
      type inet:port-number;
      description
        "minimum value of source port range";
    }
    leaf source-port-max {
      type inet:port-number;
      description
        "maximum value of source port range";
    }
  }
  description
    "Filter grouping containing list of source port ranges";
}
grouping destination-port-cfg {
  list destination-port-cfg {
    key "destination-port-min destination-port-max";
    description
      "list of ranges of destination port";
    leaf destination-port-min {
      type inet:port-number;
      description
        "minimum value of destination port range";
    }
    leaf destination-port-max {
      type inet:port-number;
      description
        "maximum value of destination port range";
    }
  }
  description
    "Filter grouping containing list of destination port ranges";
}
grouping protocol-cfg {
  list protocol-cfg {
    key "protocol-min protocol-max";
    description
      "list of ranges of protocol values";
    leaf protocol-min {
      type uint8 {
      description
        "minimum value of protocol range";
    }
range "0..255";
} description
"minimum value of protocol range";
} leaf protocol-max {
  type uint8 {
    range "0..255";
  } description
"maximum value of protocol range";
}
} description
"Filter grouping containing list of Protocol ranges";
} grouping traffic-group-cfg {
  container traffic-group-cfg {
    leaf traffic-group-name {
      type string;
description
      "This leaf defines name of the traffic group referenced";
    } description
      "traffic group container";
    description
      "traffic group grouping";
  }

  augment "/classifier:classifiers/classifier:classifier-entry" +
  "/classifier:filter-entry" {
    choice filter-param {
description
      "Choice of filter types";
    case dscp {
      uses dscp-cfg;
description
      "Filter containing list of dscp ranges";
    }
    case source-ipv4-address {
      uses source-ipv4-address-cfg;
description
      "Filter containing list of source ipv4 addresses";
    }
    case destination-ipv4-address {
      uses destination-ipv4-address-cfg;
description
      "Filter containing list of destination ipv4 addresses";
    }
  }
"Filter containing list of destination ipv4 address";
}
case source-ipv6-address {
    uses source-ipv6-address-cfg;
    description
    "Filter containing list of source ipv6 addresses";
}
case destination-ipv6-address {
    uses destination-ipv6-address-cfg;
    description
    "Filter containing list of destination ipv6 address";
}
case source-port {
    uses source-port-cfg;
    description
    "Filter containing list of source-port ranges";
}
case destination-port {
    uses destination-port-cfg;
    description
    "Filter containing list of destination-port ranges";
}
case protocol {
    uses protocol-cfg;
    description
    "Filter Type Protocol";
}
case traffic-group {
    uses traffic-group-cfg;
    description
    "Filter Type traffic-group";
}
}  
description
"augments diffserv filters to qos classifier";
}
augment "/policy:policies/policy:policy-entry" + "/policy:classifier-entry/policy:filter-entry" {
    when "../../.policy:policy-type = 
        'diffserv:ipv4-diffserv-policy-type' or 
../../.policy:policy-type = 
        'diffserv:ipv6-diffserv-policy-type' or 
../../.policy:policy-type = 
        'diffserv:diffserv-policy-type'" {
        description
        "Filters can be augmented if policy type is 
ipv4, ipv6 or default diffserv policy types ";
    }
}
description
    "Augments Diffserv Classifier with common filter types";
choice filter-params {
    description
    "Choice of action types";
    case dscp {
        uses dscp-cfg;
        description
        "Filter containing list of dscp ranges";
    }
    case source-ipv4-address {
        when "../../policy:policy-type !=
            'diffserv:ipv6-diffserv-policy-type'" {
            description
            "If policy type is v6, this filter cannot be used.";
        }
        uses source-ipv4-address-cfg;
        description
        "Filter containing list of source ipv4 addresses";
    }
    case destination-ipv4-address {
        when "../../policy:policy-type !=
            'diffserv:ipv6-diffserv-policy-type'" {
            description
            "If policy type is v6, this filter cannot be used.";
        }
        uses destination-ipv4-address-cfg;
        description
        "Filter containing list of destination ipv4 address";
    }
    case source-ipv6-address {
        when "../../policy:policy-type !=
            'diffserv:ipv4-diffserv-policy-type'" {
            description
            "If policy type is v4, this filter cannot be used.";
        }
        uses source-ipv6-address-cfg;
        description
        "Filter containing list of source ipv6 addresses";
    }
    case destination-ipv6-address {
        when "../../policy:policy-type !=
            'diffserv:ipv4-diffserv-policy-type'" {
            description
            "If policy type is v4, this filter cannot be used.";
        }
        uses destination-ipv6-address-cfg;
        description
        "Filter containing list of destination ipv6 addresses";
    }
}
"Filter containing list of destination ipv6 address";
}
case source-port {
  uses source-port-cfg;
  description
  "Filter containing list of source-port ranges";
}
case destination-port {
  uses destination-port-cfg;
  description
  "Filter containing list of destination-port ranges";
}
case protocol {
  uses protocol-cfg;
  description
  "Filter Type Protocol";
}
case traffic-group {
  uses traffic-group-cfg;
  description
  "Filter Type traffic-group";
}
}
}

augment "/policy:policies/policy:policy-entry" +
"/policy:classifier-entry" +
"/policy:classifier-action-entry-cfg" +
"/policy:action-cfg-params" {
  when "./././policy:policy-type =
    'diffserv:ipv4-diffserv-policy-type' or
    ./././policy:policy-type =
    'diffserv:ipv6-diffserv-policy-type' or
    ./././policy:policy-type =
    'diffserv:diffserv-policy-type' " {
    description
    "Actions can be augmented if policy type is ipv4, ipv6 or default diffserv policy types ";
  }
  description
  "Augments Diffserv Policy with action configuration";
}
case dscp-marking {
  uses action:dscp-marking;
}
case meter-inline {
  if-feature action:meter-inline-feature;
  uses action:meter;
}
case meter-reference {
if-feature action:meter-reference-feature;
uses action:meter-reference;
}
case child-policy {
if-feature action:child-policy-feature;
uses action:child-policy;
}
case count {
if-feature action:count-feature;
uses action:count;
}
case named-count {
if-feature action:named-counter-feature;
uses action:named-counter;
}
case queue-inline {
if-feature diffserv-queue-inline-support;
uses action:queue;
}
case scheduler-inline {
if-feature diffserv-scheduler-inline-support;
uses action:scheduler;
}
}

6.6. IETF-QUEUE-POLICY

<CODE BEGINS>file "ietf-queue-policy@2019-03-13.yang"
module ietf-queue-policy {
  yang-version 1.1;
  prefix queue-policy;

  import ietf-qos-policy {
    prefix policy;
    reference "RFC XXXX: YANG Model for QoS";
  }
  import ietf-qos-action {
    prefix action;
    reference "RFC XXXX: YANG Model for QoS";
  }
  import ietf-diffserv {
    prefix diffserv;
    reference "RFC XXXX: YANG Model for QoS";
  }

This module contains a collection of YANG definitions for configuring DiffServ specification implementations. Copyright (c) 2019 IETF Trust and the persons identified as the document authors. 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 2019-03-13 {
  description
    "Latest revision of queuing policy module";
  reference "RFC XXXX: YANG Model for QoS";
}

feature queue-policy-support {
  description
    "This feature allows queue policy configuration as a separate policy type support."
}

feature queue-inline-support {
  description
    "Queue inline support in Queue policy"
}

feature queue-template-support {
  description
    "Queue template support in Queue policy";
}
identity queue-policy-type {
    base policy:policy-type;
    description
        "This defines queue policy-type";
}

augment "/policy:policies/policy:policy-entry" + 
    "/policy:classifier-entry/policy:filter-entry" {
    when "././policy:policy-type =
        'queue-policy:queue-policy-type'" {
        description
            "If policy type is v6, this filter cannot be used.";
    }
    if-feature queue-policy-support;
    choice filter-params {
        description
            "Choice of action types";
        case traffic-group-name {
            uses diffserv:traffic-group-cfg;
            description
                "traffic group name";
        }
    }
    description
        "Augments Queue policy Classifier with common filter types";
}

identity queue-template-name {
    base policy:action-type;
    description
        "queue template name";
}

grouping queue-template-reference {
    container queue-template-reference-cfg {
        leaf queue-template-name {
            type string;
            mandatory true;
            description
                "This leaf defines name of the queue template referenced";
        }
        description
            "queue template reference";
    }
    description
        "queue template reference grouping";
container queue-template {
  if-feature queue-policy-support;
  description "Queue template";
  leaf name {
    type string;
    description "A unique name identifying this queue template";
  }
  uses action:queue;
}

augment "/policy:policies/policy-policy-entry" +
    "/policy:classifier-entry" +
    "/policy:classifier-action-entry-cfg" +
    "/policy:action-cfg-params" {
  when "../../policy:policy-type = 'queue-policy:queue-policy-type'" {
    description "queue policy actions."
  }
  if-feature queue-policy-support;
  case queue-template-name {
    if-feature queue-template-support;
    uses queue-template-reference;
  }
  case queue-inline {
    if-feature queue-inline-support;
    uses action:queue;
  }
  description "augments queue template reference to queue policy";
}

6.7. IETF-SCHEDULER-POLICY

<CODE BEGINS>file "ietf-scheduler-policy@2019-03-13.yang"
module ietf-scheduler-policy {
  yang-version 1.1;
  prefix scheduler-policy;
  import ietf-qos-classifier {

prefix classifier;
  reference "RFC XXXX: YANG Model for QoS";
}
import ietf-qos-policy {
  prefix policy;
  reference "RFC XXXX: YANG Model for QoS";
}
import ietf-qos-action {
  prefix action;
  reference "RFC XXXX: YANG Model for QoS";
}

organization "IETF RTG (Routing Area) Working Group";
contact
  "WG Web: <http://tools.ietf.org/wg/rtgwg/>
  WG List: <mailto:rtgwg@ietf.org>
  WG Chair: Chris Bowers <mailto:cbowers@juniper.net>
  WG Chair: Jeff Tantsura <mailto:jefftant.ietf@gmail.com>
  Editor: Norm Strahle <mailto:nstrahle@juniper.net>
  Editor: Aseem Choudhary <mailto:asechoud@cisco.com>"

description
  "This module contains a collection of YANG definitions for
  configuring diffserv specification implementations.
  Copyright (c) 2019 IETF Trust and the persons identified as
  the document authors. 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
  This version of this YANG module is part of RFC XXXX; see
  the RFC itself for full legal notices.";

revision 2019-03-13 {
  description
    "Latest revision of scheduler policy module";
  reference "RFC XXXX: YANG Model for QoS";
}
feature scheduler-policy-support {
  description
    "This feature allows scheduler policy configuration
     as a separate policy type support.";
}
identity scheduler-policy-type {
    base policy:policy-type;
    description
        "This defines scheduler policy-type";
}

identity filter-match-all {
    base classifier:filter-type;
    description
        "Traffic-group filter type";
}

grouping filter-match-all-cfg {
    container match-all-cfg {
        leaf match-all-action {
            type empty;
            description
                "match all packets";
        }
        description
            "the match-all action";
    }
    description
        "the match-all filter grouping";
}

augment "/policy:policies/policy:policy-entry" + 
    "/policy:classifier-entry/policy:filter-entry" {
    when "./.policy:policy-type = 
        'scheduler-policy:scheduler-policy-type'" {
        description
            "Only when policy type is scheduler-policy";
    }
    choice filter-params {
        description
            "Choice of action types";
        case filter-match-all {
            uses filter-match-all-cfg;
            description
                "filter match-all";
        }
    }
    description
        "Augments Queue policy Classifier with common filter types";
}

identity queue-policy-name {
    base policy:action-type;
description
   "queue policy name";
}

grouping queue-policy-name-cfg {
  container queue-policy-name {
    leaf queue-policy {
      type string;
      mandatory true;
      description
        "This leaf defines name of the queue-policy";
    }
    description
      "container for queue-policy name";
  }
  description
    "queue-policy name grouping";
}

augment "/policy:policies/policy:policy-entry" +
   "/policy:classifier-entry" +
   "/policy:classifier-action-entry-cfg" +
   "/policy:action-cfg-params" {
  when ".../../policy:policy-type =
       'scheduler-policy:scheduler-policy-type'" {
    description
      "Only when policy type is scheduler-policy";
  }
  case scheduler {
    uses action:scheduler;
  }
  case queue-policy-name {
    uses queue-policy-name-cfg;
  }
  description
    "augments scheduler template reference to scheduler policy";
}

7.  IANA Considerations

TBD
8.  Security Considerations

9.  Acknowledgement
   The authors wish to thank Ruediger Geib, Fred Baker, Greg Misky, Tom Petch, many others for their helpful comments.

10. References
10.1.  Normative References


10.2. Informative References


Appendix A. Company A, Company B and Company C examples

Company A, Company B and Company C Diffserv modules augments all the filter types of the QoS classifier module as well as the QoS policy module that allow it to define marking, metering, min-rate, max-rate actions. Queuing and metering counters are realized by augmenting of the QoS target module.

A.1. Example of Company A Diffserv Model

The following Company A vendor example augments the qos and diffserv model, demonstrating some of the following functionality:

- use of template based classifier definitions

- use of single policy type modelling queue, scheduler policy, and a filter policy. All of these policies either augment the qos policy or the diffserv modules

- use of inline actions in a policy

- flexibility in marking dscp or metadata at ingress and/or egress.

```yang
module example-compa-diffserv {
  namespace "urn:ietf:params:xml:ns:yang:example-compa-diffserv";
  prefix example;

  import ietf-qos-classifier {
```

prefix classifier;
    reference "RFC XXXX: YANG Model for QoS";
}
import ietf-qos-policy {
    prefix policy;
    reference "RFC XXXX: YANG Model for QoS";
}
import ietf-qos-action {
    prefix action;
    reference "RFC XXXX: YANG Model for QoS";
}
import ietf-diffserv {
    prefix diffserv;
    reference "RFC XXXX: YANG Model for QoS";
}
organization "Company A";
contact
    "Editor:   XYZ
        <mailto:xyz@compa.com>";
description
    "This module contains a collection of YANG definitions of
    companyA diffserv specification extension.";
revision 2019-03-13 {
    description
        "Initial revision for diffserv actions on network packets";
    reference
        "RFC 6020: YANG - A Data Modeling Language for the
        Network Configuration Protocol (NETCONF)";
}

identity default-policy-type {
    base policy:policy-type;
    description
        "This defines default policy-type";
}

identity qos-group {
    base classifier:filter-type;
    description
        "qos-group filter-type";
}

grouping qos-group-cfg {
    list qos-group-cfg {
        key "qos-group-min qos-group-max";
        description
            "list of dscp ranges";
    }
leaf qos-group-min {
  type uint8;
  description
    "Minimum value of qos-group range";
}
leaf qos-group-max {
  type uint8;
  description
    "maximum value of qos-group range";
}

description
  "Filter containing list of qos-group ranges";
}
grouping wred-threshold {
  container wred-min-thresh {
    uses action:threshold;
    description
      "Minimum threshold";
  }
  container wred-max-thresh {
    uses action:threshold;
    description
      "Maximum threshold";
  }
  leaf mark-probability {
    type uint32 {
      range "1..1000";
    }
    description
      "Mark probability";
  }
  description
    "WRED threshold attributes";
}
grouping randomdetect {
  leaf exp-weighting-const {
    type uint32;
    description
      "Exponential weighting constant factor for wred profile";
  }
  uses wred-threshold;
  description
    "Random detect attributes";
}
augment "/classifier:classifier" +
  "classifier:classifier-entry" +
  "classifier:classifier-entry/diffserv:filter-param" {
  case qos-group {
    uses qos-group-cfg;
    description
      "Filter containing list of qos-group ranges.
       Qos-group represent packet metadata information
       in a device.";
  }
  description
    "augmentation of classifier filters";
}

augment "/policy:policies/policy:policy-entry" +
  "policy:classifier-entry" +
  "policy:classifier-action-entry-cfg" +
  "policy:action-cfg-params" {
  case random-detect {
    uses randomdetect;
  }
  description
    "Augment the actions to policy entry";
}

augment "/policy:policies" +
  "/policy:policy-entry" +
  "/policy:classifier-entry" +
  "/policy:classifier-action-entry-cfg" +
  "/policy:action-cfg-params" +
  "/diffserv:meter-inline" +
  "/diffserv:meter-type" +
  "/diffserv:one-rate-two-color-meter-type" +
  "/diffserv:one-rate-two-color-meter" +
  "/diffserv:conform-action" +
  "/diffserv:conform-2color-meter-action-params" +
  "/diffserv:conform-2color-meter-action-val" {
  description
    "augment the one-rate-two-color meter conform
     with actions";
  case meter-action-drop {
    description
      "meter drop";
    uses action:drop;
  }
  case meter-action-mark-dscp {
    description
      "meter action dscp marking";
uses action:dscp-marking;
}
}
augment "/policy:policies" +
  "/policy:policy-entry" +
  "/policy:classifier-entry" +
  "/policy:classifier-action-entry-cfg" +
  "/policy:action-params" +
  "/diffserv:meter-inline" +
  "/diffserv:meter-type" +
  "/diffserv:one-rate-two-color-meter-type" +
  "/diffserv:one-rate-two-color-meter" +
  "/diffserv:exceed-action" +
  "/diffserv:exceed-2color-meter-action-params" +
  "/diffserv:exceed-2color-meter-action-val" {

description
  "augment the one-rate-two-color meter exceed with actions";

  case meter-action-drop {
    description
      "meter drop";
    uses action:drop;
  }

  case meter-action-mark-dscp {
    description
      "meter action dscp marking";
    uses action:dscp-marking;
  }
}
}
augment "/policy:policies" +
  "/policy:policy-entry" +
  "/policy:classifier-entry" +
  "/policy:classifier-action-entry-cfg" +
  "/policy:action-params" +
  "/diffserv:meter-inline" +
  "/diffserv:meter-type" +
  "/diffserv:one-rate-tri-color-meter-type" +
  "/diffserv:one-rate-tri-color-meter" +
  "/diffserv:conform-action" +
  "/diffserv:conform-3color-meter-action-params" +
  "/diffserv:conform-3color-meter-action-val" {

description
  "augment the one-rate-tri-color meter conform with actions";

  case meter-action-drop {
    description
      "augment the one-rate-two-color meter exceed with actions";

    case meter-action-drop {
      description
        "meter drop";
      uses action:drop;
    }

    case meter-action-mark-dscp {
      description
        "meter action dscp marking";
      uses action:dscp-marking;
    }
  }
}

"meter drop";
  uses action:drop;
}
case meter-action-mark-dscp {
  description
  "meter action dscp marking";
  uses action:dscp-marking;
}
}
augment "/policy:policies" +
  "/policy:policy-entry" +
  "/policy:classifier-entry" +
  "/policy:classifier-action-entry-cfg" +
  "/policy:action-cfg-params" +
  "/difserv:one-rate-tri-color-meter-type" +
  "/difserv:one-rate-tri-color-meter" +
  "/difserv:exceed-action" +
  "/difserv:exceed-3color-meter-action-params" +
  "/difserv:exceed-3color-meter-action-val" {
  description
  "augment the one-rate-tri-color meter exceed
  with actions";
  case meter-action-drop {
    description
    "meter drop";
    uses action:drop;
  }
  case meter-action-mark-dscp {
    description
    "meter action dscp marking";
    uses action:dscp-marking;
  }
}
augment "/policy:policies" +
  "/policy:policy-entry" +
  "/policy:classifier-entry" +
  "/policy:classifier-action-entry-cfg" +
  "/policy:action-cfg-params" +
  "/difserv:one-rate-tri-color-meter-type" +
  "/difserv:one-rate-tri-color-meter" +
  "/difserv:violate-action" +
  "/difserv:violate-3color-meter-action-params" +
  "/difserv:violate-3color-meter-action-val" {
description
    "augment the one-rate-tri-color meter conform
    with actions";

case meter-action-drop {
    description
    "meter drop";
    uses action:drop;
}

case meter-action-mark-dscp {
    description
    "meter action dscp marking";
    uses action:dscp-marking;
}

augment "/policy:policies" +
    "/policy:policy-entry" +
    "/policy:classifier-entry" +
    "/policy:classifier-action-entry-cfg" +
    "/policy:action-cfg-params" +
    "/diffserv:conform-action" +
    "/diffserv:conform-3color-meter-action-params" +
    "/diffserv:conform-3color-meter-action-val" {
        description
        "augment the one-rate-tri-color meter conform
        with actions";

        case meter-action-drop {
            description
            "meter drop";
            uses action:drop;
        }

        case meter-action-mark-dscp {
            description
            "meter action dscp marking";
            uses action:dscp-marking;
        }
    }
"/difserv:meter-type" +
"/difserv:two-rate-tri-color-meter-type" +
"/difserv:two-rate-tri-color-meter" +
"/difserv:exceed-action" +
"/difserv:exceed-3color-meter-action-params" +
"/difserv:exceed-3color-meter-action-val" {

description
"augment the two-rate-tri-color meter exceed with actions";

case meter-action-drop {

description
"meter drop";

uses action:drop;
}

case meter-action-mark-dscp {

description
"meter action dscp marking";

uses action:dscp-marking;
}
}

augment "/policy:policies" +
"/policy:policy-entry" +
"/policy:classifier-entry" +
"/policy:classifier-action-entry-cfg" +
"/policy:action-cfg-params" +
"/difserv:meter-inline" +
"/difserv:meter-type" +
"/difserv:two-rate-tri-color-meter-type" +
"/difserv:two-rate-tri-color-meter" +
"/difserv:violate-action" +
"/difserv:violate-3color-meter-action-params" +
"/difserv:violate-3color-meter-action-val" {

description
"augment the two-rate-tri-color meter violate with actions";

case meter-action-drop {

description
"meter drop";

uses action:drop;
}

case meter-action-mark-dscp {

description
"meter action dscp marking";

uses action:dscp-marking;
}
}
"/policy:policy-entry" +
"/policy:classifier-entry" +
"/policy:classifier-action-entry-cfg" +
"/policy:action-cfg-params" +
"/diffserv:meter-inline" +
"/diffserv:meter-type" +
"/diffserv:one-rate-two-color-meter-type" +
"/diffserv:one-rate-two-color-meter" {
  description
  "augment the one-rate-two-color meter with" +
  "color classifiers";
  container conform-color {
    uses classifier:classifier-entry-generic-attr;
    description
    "conform color classifier container";
  }
  container exceed-color {
    uses classifier:classifier-entry-generic-attr;
    description
    "exceed color classifier container";
  }
}

augment "/policy:policies" +
"/policy:policy-entry" +
"/policy:classifier-entry" +
"/policy:classifier-action-entry-cfg" +
"/policy:action-cfg-params" +
"/diffserv:meter-inline" +
"/diffserv:meter-type" +
"/diffserv:one-rate-tri-color-meter-type" +
"/diffserv:one-rate-tri-color-meter" {
  description
  "augment the one-rate-tri-color meter with" +
  "color classifiers";
  container conform-color {
    uses classifier:classifier-entry-generic-attr;
    description
    "conform color classifier container";
  }
  container exceed-color {
    uses classifier:classifier-entry-generic-attr;
    description
    "exceed color classifier container";
  }
  container violate-color {
    uses classifier:classifier-entry-generic-attr;
    description
    "violate color classifier container";
  }
}
 augment "/policy:policies" + 
 "/policy:policy-entry" + 
 "/policy:classifier-entry" + 
 "/policy:classifier-action-entry-cfg" + 
 "/policy:action-cfg-params" + 
 "/diffserv:meter-inline" + 
 "/diffserv:meter-type" + 
 "/diffserv:two-rate-tri-color-meter-type" + 
 "/diffserv:two-rate-tri-color-meter" {
 description 
 "augment the two-rate-tri-color meter with" + 
 "color classifiers";
 container conform-color {
 uses classifier:classifier-entry-generic-attr;
 description 
 "conform color classifier container";
 }
 container exceed-color {
 uses classifier:classifier-entry-generic-attr;
 description 
 "exceed color classifier container";
 }
 container violate-color {
 uses classifier:classifier-entry-generic-attr;
 description 
 "violate color classifier container";
 }
}

A.2. Example of Company B Diffserv Model

The following vendor example augments the qos and diffserv model, 
demonstrating some of the following functionality:

- use of inline classifier definitions (defined inline in the policy 
  vs referencing an externally defined classifier)

- use of multiple policy types, e.g. a queue policy, a scheduler 
  policy, and a filter policy. All of these policies either augment 
  the qos policy or the diffserv modules

- use of a queue module, which uses and extends the queue grouping 
  from the ietf-qos-action module

- use of meter templates (v.s. meter inline)
module example-compb-diffserv-filter-policy {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:" + "example-compb-diffserv-filter-policy";
  prefix compb-filter-policy;

  import ietf-qos-classifier {
    prefix classifier;
    reference "RFC XXXX: YANG Model for QoS";
  }
  import ietf-qos-policy {
    prefix policy;
    reference "RFC XXXX: YANG Model for QoS";
  }
  import ietf-qos-action {
    prefix action;
    reference "RFC XXXX: YANG Model for QoS";
  }
  import ietf-diffserv {
    prefix diffserv;
    reference "RFC XXXX: YANG Model for QoS";
  }

  organization "Company B";
  contact
    "Editor:   XYZ
<mailto:xyz@compb.com>";

  description
    "This module contains a collection of YANG definitions for configuring
diffserv specification implementations.

Copyright (c) 2019 IETF Trust and the persons identified as
the document authors. 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 2019-03-13 {
description
"Latest revision of diffserv policy";
reference "RFC XXXX";
}

/*************************************************
* Classification types
*************************************************/

identity forwarding-class {
  base classifier:filter-type;
  description
    "Forwarding class filter type";
}

identity internal-loss-priority {
  base classifier:filter-type;
  description
    "Internal loss priority filter type";
}

grouping forwarding-class-cfg {
  list forwarding-class-cfg {
    key "forwarding-class";
    description
      "list of forwarding-classes";
    leaf forwarding-class {
      type string;
      description
        "Forwarding class name";
    }
  }
  description
    "Filter containing list of forwarding classes";
}

grouping loss-priority-cfg {
  list loss-priority-cfg {
    key "loss-priority";
    description
      "list of loss-priorities";
    leaf loss-priority {
      type enumeration {
        enum high {
          description "High Loss Priority";
        }
        enum medium-high {
description "Medium-high Loss Priority";
}
enum medium-low {
    description "Medium-low Loss Priority";
}
enum low {
    description "Low Loss Priority";
}
}

description "Loss-priority";
}

description "Filter containing list of loss priorities";
}

augment "/policy:policies" +
   "/policy:policy-entry" +
   "/policy:classifier-entry" +
   "/policy:filter-entry" +
   "/diffserv:filter-params" {
    case forwarding-class {
        uses forwarding-class-cfg;
        description "Filter Type Internal-loss-priority";
    }
    case internal-loss-priority {
        uses loss-priority-cfg;
        description "Filter Type Internal-loss-priority";
    }
    description "Augments DiffServ Classifier with vendor specific types";
}

/******************************************************************************
* Actions
*******************************************************************************/

identity mark-fwd-class {
    base policy:action-type;
    description "mark forwarding class action type";
}

identity mark-loss-priority {
base policy:action-type;
description
  "mark loss-priority action type";
}

grouping mark-fwd-class {
  container mark-fwd-class-cfg {
    leaf forwarding-class {
      type string;
      description
        "Forwarding class name";
    }
    description
      "mark-fwd-class container";
  }
  description
    "mark-fwd-class grouping";
}

grouping mark-loss-priority {
  container mark-loss-priority-cfg {
    leaf loss-priority {
      type enumeration {
        enum high {
          description "High Loss Priority";
        }
        enum medium-high {
          description "Medium-high Loss Priority";
        }
        enum medium-low {
          description "Medium-low Loss Priority";
        }
        enum low {
          description "Low Loss Priority";
        }
      }
      description
        "Loss-priority";
    }
    description
      "mark-loss-priority container";
  }
  description
    "mark-loss-priority grouping";
}

identity exceed-2color-meter-action-drop {
  base action:exceed-2color-meter-action-type;
description
   "drop action type in a meter";
}

identity meter-action-mark-fwd-class {
   base action:exceed-2color-meter-action-type;
   description
   "mark forwarding class action type";
}

identity meter-action-mark-loss-priority {
   base action:exceed-2color-meter-action-type;
   description
   "mark loss-priority action type";
}

identity violate-3color-meter-action-drop {
   base action:violate-3color-meter-action-type;
   description
   "drop action type in a meter";
}

augment "/policy:policies/policy:policy-entry/" +
   "/policy:classifier-entry/" +
   "/policy:classifier-action-entry-cfg/" +
   "policy:action-cfg-params" {
   case mark-fwd-class {
      uses mark-fwd-class;
      description
      "Mark forwarding class in the packet";
   }
   case mark-loss-priority {
      uses mark-loss-priority;
      description
      "Mark loss priority in the packet";
   }
   case discard {
      uses action:discard;
      description
      "Discard action";
   }
   description
   "Augments common diffserv policy actions";
}

augment "/action:meter-template" +
   "/action:meter-entry" +
leaf one-rate-color-aware {
  type boolean;
  description
    "This defines if the meter is color-aware";
}

augment "/action:meter-template" +
  "/action:meter-entry" +
  "/action:one-rate-tri-color-meter-type" +
  "/action:one-rate-tri-color-meter" {
  leaf one-rate-color-aware {
    type boolean;
    description
      "This defines if the meter is color-aware";
  }
}

/* example of augmenting a meter template with a */
/* vendor specific action */
augment "/action:meter-template" +
  "/action:meter-entry" +
  "/action:one-rate-tri-color-meter-type" +
  "/action:two-rate-tri-color-meter-type" +
  "/action:two-rate-tri-color-meter" {
  leaf two-rate-color-aware {
    type boolean;
    description
      "This defines if the meter is color-aware";
  }
}

case exceed-2color-meter-action-drop {
  description
    "meter drop";
  uses action:drop;
}
case meter-action-mark-fwd-class {
  uses mark-fwd-class;
  description
    "Mark forwarding class in the packet";
}
case meter-action-mark-loss-priority {
  uses mark-loss-priority;
  description
    "Mark loss priority in the packet";
}
augment "/action:meter-template" +
    "/action:meter-entry" +
    "/action:meter-type" +
    "/action:two-rate-tri-color-meter-type" +
    "/action:two-rate-tri-color-meter" +
    "/action:violate-action" +
    "/action:violate-3color-meter-action-params" +
    "/action:violate-3color-meter-action-val" {
  case exceed-3color-meter-action-drop {
    description
      "meter drop";
    uses action:drop;
  }
}

description
  "Augment the actions to the two-color meter";
}

augment "/action:meter-template" +
    "/action:meter-entry" +
    "/action:meter-type" +
    "/action:one-rate-tri-color-meter-type" +
    "/action:one-rate-tri-color-meter" +
    "/action:violate-action" +
    "/action:violate-3color-meter-action-params" +
    "/action:violate-3color-meter-action-val" {
  case exceed-3color-meter-action-drop {
    description
      "meter drop";
    uses action:drop;
  }
}

description
  "Augment the actions to basic meter";
}

module example-compb-queue-policy {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:example-compb-queue-policy";
  prefix queue-plcy;

  import ietf-qos-classifier {
    prefix classifier;
    reference "RFC XXXX: YANG Model for QoS";
  }

  import ietf-qos-policy {
    prefix policy;
}

This module defines a queue policy. The classification is based on a forwarding class, and the actions are queues. Copyright (c) 2019 IETF Trust and the persons identified as the document authors. 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 2019-03-13 {
  description
    "Latest revision of diffserv policy";
  reference "RFC XXXX";
}

identity forwarding-class {
  base classifier:filter-type;
  description
    "Forwarding class filter type";
}

grouping forwarding-class-cfg {
  leaf forwarding-class-cfg {
    type string;
    description
      "forwarding-class name";
  }
  description
    "Forwarding class filter";
}

augment "/policy:policies" +
  "/policy:policy-entry" +
  "/policy:classifier-entry" +
  "/policy:filter-entry" {
/ Does NOT support "logical-not" of forwarding class. Use "must"? */

choice filter-params {
    description "Choice of filters";
    case forwarding-class-cfg {
        uses forwarding-class-cfg;
        description "Filter Type Internal-loss-priority";
    }
}

description "Augments Diffserv Classifier with fwd class filter";

identity compb-queue {
    base policy:action-type;
    description "compb-queue action type";
}

grouping compb-queue-name {
    container queue-name {
        leaf name {
            type string;
            description "Queue class name";
        }
        description "compb queue container";
    }
    description "compb-queue grouping";
}

augment "/policy:policies" + "/policy:policy-entry" + "/policy:classifier-entry" + "/policy:classifier-action-entry-cfg" {
    choice action-cfg-params {
        description "Choice of action types";
        case compb-queue {
            uses compb-queue-name;
        }
    }
    description "Augment the queue actions to queue policy entry";
module example-compb-queue {
  yang-version 1.1;
  prefix compb-queue;

  import ietf-qos-action {
    prefix action;
    reference "RFC XXXX: YANG Model for QoS";
  }

  organization "Company B";
  contact
    "Editor: XYZ
     <mailto:xyz@compb.com>";

  description
    "This module describes a compb queue module. This is a
     template for a queue within a queue policy, referenced
     by name.

     This version of this YANG module is part of RFC XXXX; see
     the RFC itself for full legal notices.";

  revision 2019-03-13 {
    description
      "Latest revision of diffserv based classifier";
    reference "RFC XXXX";
  }

  container compb-queue {
    description
      "Queue used in compb architecture";
    leaf name {
      type string;
      description
        "A unique name identifying this queue";
    }
    uses action:queue;
    container excess-rate {
      choice excess-rate-type {
        case percent {
          leaf excess-rate-percent {
            type uint32 {
              range "1..100";
            }
          }
        }
      }
    }
}
description "excess-rate-percent";
}
}

case proportion {
  leaf excess-rate-proportion {
    type uint32 {
      range "1..1000";
    }
    description "excess-rate-proportion";
  }
}

description "Choice of excess-rate type";
}

description "Excess rate value";

leaf excess-priority {
  type enumeration {
    enum high {
      description "High Loss Priority";
    }
    enum medium-high {
      description "Medium-high Loss Priority";
    }
    enum medium-low {
      description "Medium-low Loss Priority";
    }
    enum low {
      description "Low Loss Priority";
    }
    enum none {
      description "No excess priority";
    }
  }
  description "Priority of excess (above guaranted rate) traffic";
}

container buffer-size {
  choice buffer-size-type {
    case percent {
      leaf buffer-size-percent {
        type uint32 {
          range "1..100";
        }
        description  "buffer-size-percent";
      }
    }
  }
}

case temporal {
    leaf buffer-size-temporal {
        type uint64;
        units "microsecond";
        description "buffer-size-temporal";
    }
}

case remainder {
    leaf buffer-size-remainder {
        type empty;
        description "use remaining of buffer";
    }
}

description "Choice of buffer size type";

description "Buffer size value";
}

augment "/compb-queue" + "/queue-cfg" + "/algorithmic-drop-cfg" + "/drop-algorithm" {
    case random-detect {
        list drop-profile-list {
            key "priority";
            description "map of priorities to drop-algorithms";
            leaf priority {
                type enumeration {
                    enum any {
                        description "Any priority mapped here";
                    }
                    enum high {
                        description "High Priority Packet";
                    }
                    enum medium-high {
                        description "Medium-high Priority Packet";
                    }
                    enum medium-low {

description "Medium-low Priority Packet";
enum low {
    description "Low Priority Packet";
}
}
description "Priority of guaranteed traffic";
leaf drop-profile {
    type string;
    description "drop profile to use for this priority";
}
}
description "compb random detect drop algorithm config";
}
}
module example-compb-scheduler-policy {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:" +
        "example-compb-scheduler-policy";
    prefix scheduler-plcy;
import ietf-qos-action {
    prefix action;
    reference "RFC XXXX: YANG Model for QoS";
}
import ietf-qos-policy {
    prefix policy;
    reference "RFC XXXX: YANG Model for QoS";
}
organization "Company B";
contact
    "Editor:   XYZ
        <mailto:xyz@compb.com>";
    description
        "This module defines a scheduler policy. The classification
        is based on classifier-any, and the action is a scheduler.";
revision 2019-03-13 {
    description
"Latest revision of diffserv policy";
reference "RFC XXXX";
}

identity queue-policy {
    base policy:action-type;
    description
        "forwarding-class-queue action type";
}

grouping queue-policy-name {
    container compb-queue-policy-name {
        leaf name {
            type string;
            description
                "Queue policy name";
        }
        description
            "compb-queue-policy container";
    }
    description
        "compb-queue policy grouping";
}

augment "/policy:policies" +
    "/policy:policy-entry" +
    "/policy:classifier-entry" +
    "/policy:classifier-action-entry-cfg" {
    choice action-cfg-params {
        case schedular {
            uses action:schedular;
        }
        case queue-policy {
            uses queue-policy-name;
        }
        description
            "Augment the scheduler policy with a queue policy";
    }
}

A.3. Example of Company C Diffserv Model

Company C vendor augmentation is based on Ericsson's implementation differentiated QoS. This implementation first sorts traffic based on a classifier, which can sort traffic into one or more traffic forwarding classes. Then, a policer or meter policy references the
classifier and its traffic forwarding classes to specify different service levels for each traffic forwarding class.

Because each classifier sorts traffic into one or more traffic forwarding classes, this type of classifier does not align with ietf-qos-classifier.yang, which defines one traffic forwarding class per classifier. Additionally, Company C’s policing and metering policies relies on the classifier’s pre-defined traffic forwarding classes to provide differentiated services, rather than redefining the patterns within a policing or metering policy, as is defined in ietf-diffserv.yang.

Due to these differences, even though Company C uses all the building blocks of classifier and policy, Company C’s augmentation does not use ietf-diffserv.yang to provide differentiated service levels. Instead, Company C’s augmentation uses the basic building blocks, ietf-qos-policy.yang to provide differentiated services.

module example-compc-qos-policy {
  yang-version 1.1;
  namespace "urn:example-compc-qos-policy";
  prefix "compcqos";

  import ietf-qos-policy {
    prefix "pol";
    reference "RFC XXXX: YANG Model for QoS";
  }

  import ietf-qos-action {
    prefix "action";
    reference "RFC XXXX: YANG Model for QoS";
  }

  organization "";
  contact "";
  description "";

  revision 2019-03-13 {
    description "";
    reference "";
  }

  /* identities */

  identity compc-qos-policy {
    base pol:policy-type;
  }
}
identity mdrq-queuing-policy {
    base compc-qos-policy;
}

identity pwfq-queuing-policy {
    base compc-qos-policy;
}

identity policing-policy {
    base compc-qos-policy;
}

identity metering-policy {
    base compc-qos-policy;
}

identity forwarding-policy {
    base compc-qos-policy;
}

identity overhead-profile-policy {
    base compc-qos-policy;
}

identity resource-profile-policy {
    base compc-qos-policy;
}

identity protocol-rate-limit-policy {
    base compc-qos-policy;
}

identity compc-qos-action {
    base pol:action-type;
}

/* groupings */

grouping redirect-action-grp {
    container redirect {
        /* Redirect options */
    }
}

/* deviations */

declaration "/pol:policies/pol:policy-entry" {
    deviate add {
        "redirect-action-grp">
    }
}
must "pol:type = compc-qos-policy" {
  description
  "Only policy types driven from compc-qos-policy " +
  "are supported";
}
}

deviation "/pol:policies/pol:policy-entry/pol:classifier-entry" {
  deviate add {
    must "../per-class-action = 'true'" {
      description
      "Only policies with per-class actions have classifiers";
    }
    must "((../sub-type != 'mdrr-queuing-policy') and " +
    "(../sub-type != 'pwfq-queuing-policy')) or " +
    "((../sub-type = 'mdrr-queuing-policy') or " +
    "(../sub-type = 'pwfq-queuing-policy')) and " +
    "((classifier-entry-name = '0') or " +
    "(classifier-entry-name = '1') or " +
    "(classifier-entry-name = '2') or " +
    "(classifier-entry-name = '3') or " +
    "(classifier-entry-name = '4') or " +
    "(classifier-entry-name = '5') or " +
    "(classifier-entry-name = '6') or " +
    "(classifier-entry-name = '7') or " +
    "(classifier-entry-name = '8'))" {
      description
      "MDRR queuing policy’s or PWFQ queuing policy’s " +
      "classifier-entry-name is limited to the listed values";
    }
  }
}

deviation "/pol:policies/pol:policy-entry/pol:classifier-entry" +
"/pol:classifier-action-entry-cfg" {
  deviate add {
    max-elements 1;
    must "action-type = 'compc-qos-action'" {
      description
      "Only compc-qos-action is allowed";
    }
  }
}

/* augments */

augment "/pol:policies/pol:policy-entry" {

when "pol:type = 'compc-qos-policy')" {  
    description  
        "Additional nodes only for diffserv-policy";  
}  
leaf sub-type {  
    type identityref {  
        base compc-qos-policy;  
    }  
    mandatory true;  
    /* The value of this leaf must not change once configured */  
}  
leaf per-class-action {  
    mandatory true;  
    type boolean;  
    must "{{(. = 'true') and " +  
        " (.sub-type = 'policing-policy') or " +  
        " (.sub-type = 'metering-policy') or " +  
        " (.sub-type = 'mdrr-queuing-policy') or " +  
        " (.sub-type = 'pwfq-queuing-policy') or " +  
        " (.sub-type = 'forwarding-policy'))} or " +  
        " (. = 'false') and " +  
        " (.sub-type = 'overhead-profile-policy') or " +  
        " (.sub-type = 'resource-profile-policy') or " +  
        " (.sub-type = 'protocol-rate-limit-policy'))" {  
        description  
            "Only certain policies have per-class action";  
    }  
}  
container traffic-classifier {  
    presence true;  
    when "./sub-type = 'policing-policy' or " +  
        " ./sub-type = 'metering-policy' or " +  
        " ./sub-type = 'forwarding-policy'" {  
        description  
            "A classifier for policing-policy or metering-policy";  
    }  
    leaf name {  
        type string;  
        mandatory true;  
        description  
            "Traffic classifier name";  
    }  
    leaf type {  
        type enumeration {  
            enum 'internal-dscp-only-classifier' {  
                value 0;  
                description  
                    "Classify traffic based on (internal) dscp only";  
            }  
        }  
    }  
}
enum 'ipv4-header-based-classifier' {
  value 1;
  description "Classify traffic based on IPv4 packet header fields";
}

enum 'ipv6-header-based-classifier' {
  value 2;
  description "Classify traffic based on IPv6 packet header fields";
}

mandatory true;

description "Traffic classifier type";
}

container traffic-queue {
  when "../sub-type = 'mdrr-queuing-policy'" or "../sub-type = 'pwfq-queuing-policy'" {
    description "Queuing policy properties";
  }
  leaf queue-map {
    type string;
    description "Traffic queue map for queuing policy";
  }
}

container overhead-profile {
  when "../sub-type = 'overhead-profile-policy'" {
    description "Overhead profile policy properties";
  }
}

container resource-profile {
  when "../sub-type = 'resource-profile-policy'" {
    description "Resource profile policy properties";
  }
}

container protocol-rate-limit {
  when "../sub-type = 'protocol-rate-limit-policy'" {
    description "Protocol rate limit policy properties";
  }
}
    when " ../../../pol:type = 'compc-qos-policy'"
    description
    "Configurations for a classifier-policy-type policy";
}

case metering-or-policing-policy {
    when " ../../../sub-type = 'policing-policy' or " + " ../../../sub-type = 'metering-policy'"
    }
    container dscp-marking {
        uses action:dscp-marking;
    }
    container precedence-marking {
        uses action:dscp-marking;
    }
    container priority-marking {
        uses action:priority;
    }
    container rate-limiting {
        uses action:one-rate-two-color-meter;
    }
}

case mdrr-queuing-policy {
    when " ../../../sub-type = 'mdrr-queuing-policy'"
    description
    "MDRR queue handling properties for the traffic " + 
    "classified into current queue";
    leaf mdrr-queue-weight {
        type uint8 {
            range "20..100";
        }
        units percentage;
    }
}

case pwfq-queuing-policy {
    when " ../../../sub-type = 'pwfq-queuing-policy'"
    description
    "PWFQ queue handling properties for traffic " + 
    "classified into current queue";
    leaf pwfq-queue-weight {
        type uint8 {
            range "20..100";
        }
        units percentage;
    }
}
leaf pwfq-queue-priority {
    type uint8;
}
leaf pwfq-queue-rate {
    type uint8;
}

case forwarding-policy {
    when "../../../sub-type = 'forwarding-policy'" {
        description
        "Forward policy handling properties for traffic " +
        "in this classifier";
        uses redirect-action-grp;
        description
        "Add the classify action configuration";
    }
}

Authors’ Addresses

Aseem Choudhary
Cisco Systems
170 W. Tasman Drive
San Jose, CA  95134
US
Email: asechoud@cisco.com

Mahesh Jethanandani
Cisco Systems
170 W. Tasman Drive
San Jose, CA  95134
US
Email: mjethanandani@gmail.com
Abstract

This document describes a YANG model for Quality of Service (QoS) operational parameters.

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 2, 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.
1. Introduction

This document defines a base YANG [RFC6020] [RFC7950] data module for Quality of Service (QoS) operational parameters. Remote Procedure Calls (RPC) or notification definition is currently not part of this document and will be added later if necessary. QoS configuration modules are defined by [I-D.asechoud-rtgwg-qos-model].

This document doesn’t include operational parameters for random-detect (RED), which is left to individual vendor to augment it.

Editorial Note: (To be removed by RFC Editor)

This draft contains several placeholder values that need to be replaced with finalized values at the time of publication. Please apply the following replacements: o "XXXX" --> the assigned RFC value for this draft both in this draft and in the YANG models under the revision statement. o The "revision" date in model, in the format XXXX-XX-XX, needs to be updated with the date the draft gets approved.

The YANG modules in this document conform to the Network Management Datastore Architecture (NMDA) [RFC8342].

1.1. Tree Diagrams

Tree diagrams used in this document follow the notation defined in [RFC8340]
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.

3. QoS Operational Model Design

QoS operational model include QoS policy applied to an interface in each direction of traffic. For each QoS policy applied to an interface the model further includes counters for associated Classifiers, Meters and Queues in a particular direction. To modularize and for reusability, grouping have been defined for various counters of Classifier, Meters and Queues. The target is assumed to be interface but the groupings can be used for any other target type where QoS policy is applied.

[I-D.asechoud-rtgwg-qos-model] defines various building blocks for applying a QoS Policy on a target. It includes QoS Policy configuration, which is a container of various classifiers and corresponding actions which are configured for traffic conditioning. This drafts defines the various counters for these building blocks.ietf-qos-oper module defined in this draft augments ietf-interfaces [RFC8343] module.

Classifier statistics contains counters for packets and bytes matched to the traffic in a direction and also average rate at which traffic is hitting a classifier. Classification criterion may be based on IP, MPLS or Ethernet. Counters defined in this draft are agnostic to underlying dataplane technology.

Statistics of meter is modeled based on commonly used algorithms in industry, Single Rate Tri Color Marking (srTCM) [RFC2697] meter, Two Rate Tri Color Marking (trTCM) [RFC2698] meter. Metering statistics includes counters corresponding to various rates configured. A metering container is referred by a metering identifier. This identifier could be a classifier name if the metering configuration is inline with classifier or it could be metering template name if the metering is configured as separate entity and associated with the classifier.

Queuing statistics includes counters corresponding to various queues associated with the policy. A queuing container is referred by queuing identifier. This identifier could be a classifier name if the queuing configuration is inline with classifier and hence there is one-to-one mapping between a classifier and a queue or it could be
a separate queue identifier if one or more than one classifiers are associated with a queue.

4. Modules Tree Structure

This document defines counters for classifiers, meters and queues.

Classifier statistics consists of list of classifier entries identified by a classifier entry name. Classifier counters include matched packets, bytes and average rate of traffic matching a particular classifier.

Metering statistics consists of meters identified by an identifier. Metering counters include conform, exceed, violate and drop packets and bytes.

Queuing counters include instantaneous, peak, average queue length, as well as output conform, exceed, tail drop packets and bytes.
module: ietf-qos-oper
augment /if:interfaces/if:interface:
  +--ro qos-interface-statistics
    +--ro qos-stats-per-direction*
      +--ro direction? identityref
      +--ro qos-classifier-statistics*
        |  +--ro policy-name? string
        |  +--ro classifier-entry-name? string
        |  +--ro classifier-entry-statistics
        |     +--ro classified-pkts? uint64
        |     +--ro classified-bytes? uint64
        |     +--ro classified-rate? uint64
        +--ro qos-named-statistics*
          |  +--ro stats-name? string
          |  +--ro pkts? uint64
          |  +--ro bytes? uint64
          |  +--ro rate? uint64
          +--ro metering-statistics*
            |  +--ro policy-name? string
            |  +--ro meter-id? string
            |  +--ro conform-pkts? uint64
            |  +--ro conform-bytes? uint64
            |  +--ro conform-rate? uint64
            |  +--ro exceed-pkts? uint64
            |  +--ro exceed-bytes? uint64
            |  +--ro exceed-rate? uint64
            |  +--ro violate-pkts? uint64
            |  +--ro violate-bytes? uint64
            |  +--ro violate-rate? uint64
            |  +--ro meter-drop-pkts? uint64
            |  +--ro meter-drop-bytes? uint64
          +--ro queueing-statistics*
            |  +--ro policy-name? string
            |  +--ro queue-id? string
            |  +--ro queueing-statistics
            |     +--ro output-conform-pkts? uint64
            |     +--ro output-conform-bytes? uint64
            |     +--ro output-exceed-pkts? uint64
            |     +--ro output-exceed-bytes? uint64
            |     +--ro queue-current-size-bytes? uint64
            |     +--ro queue-average-size-bytes? uint64
            |     +--ro queue-peak-size-bytes? uint64
            |     +--ro tailed-drop-pkts? uint64
            |     +--ro tailed-drop-bytes? uint64
5. Modules

5.1. IETF-QOS-OPER

module ietf-qos-oper {
    yang-version 1.1;
    prefix oper;
    import ietf-interfaces {
        prefix if;
        reference "RFC8343: A YANG Data Model for Interface Management";
    }
    organization "IETF RTG (Routing Area) Working Group";
    contact
        "WG Web:  <http://tools.ietf.org/wg/rtgwg/>
        WG List:  <mailto:ietf.org>
        WG Chair: Chris Bowers
                   <mailto:cbowers@juniper.net>
        WG Chair: Jeff Tantsura
                   <mailto:jefftant.ietf@gmail.com>
        Editor:   Aseem Choudhary
                   <mailto:asechoud@cisco.com>"
    description
        "This module contains a collection of YANG definitions for
         configuring qos specification implementations.  
         Copyright (c) 2019 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 2019-06-28 {
        description 
            "Latest revision qos based policy applied to a target";
            reference "RFC XXXX: YANG Model for QoS Operational Parameters";
    }
    identity direction {
        description 
            "This is identity of traffic direction";
    }
    identity inbound {
        base direction;
        description 

"Direction of traffic coming into the network entry";
}
identity outbound {
  base direction;
  description
    "Direction of traffic going out of the network entry";
}

grouping classifier-entry-stats {
  description
    "Classifier Counters";
  container classifier-entry-statistics {
    config false;
    description
      "This group defines the classifier filter statistics of each classifier entry";
    leaf classified-pkts {
      type uint64;
      description
        "Number of total packets which filtered to a classifier-entry";
    }
    leaf classified-bytes {
      type uint64;
      description
        "Number of total bytes which filtered to a classifier-entry";
    }
    leaf classified-rate {
      type uint64;
      units "bits-per-second";
      description
        "Rate of average data flow through a classifier-entry";
    }
  }
}


grouping qos-named-stats {
  description
    "QoS matching statistics associated with a stats-name";
  leaf pkts {
    type uint64;
    description
      "Number of total matched packets associated to a statistics name";
  }
  leaf bytes {

type uint64;
description
"Number of total matched bytes associated to a statistics name";
}
leaf rate {
  type uint64;
  units "bits-per-second";
  description
  "Rate of average matched data which is associated to a statistics name";
}
}
grouping queuing-stats {
  description
  "Queuing Counters";
  container queuing-statistics {
    description
    "queue related statistics ";
    leaf output-conform-pkts {
      type uint64;
      description
      "Number of packets transmitted from queue which conform to min-rate";
    }
    leaf output-conform-bytes {
      type uint64;
      description
      "Number of bytes transmitted from queue conform to min-rate";
    }
    leaf output-exceed-pkts {
      type uint64;
      description
      "Number of packets transmitted from queue which exceed min-rate but conform to max-rate";
    }
    leaf output-exceed-bytes {
      type uint64;
      description
      "Number of bytes transmitted from queue which exceed min-rate but conform to max-rate ";
    }
    leaf queue-current-size-bytes {
      type uint64;
      description
      "Number of bytes currently buffered ";
    }
  }
}
leaf queue-average-size-bytes {
    type uint64;
    description "Average queue size in number of bytes";
}
leaf queue-peak-size-bytes {
    type uint64;
    description "Peak buffer queue size in bytes";
}
leaf tailed-drop-pkts {
    type uint64;
    description "Total number of packets tail-dropped";
}
leaf tailed-drop-bytes {
    type uint64;
    description "Total number of bytes tail-dropped";
}

grouping meter-stats {
    description "Metering Statistics";
    leaf conform-pkts {
        type uint64;
        description "Number of conform packets";
    }
    leaf conform-bytes {
        type uint64;
        description "Number of bytes of conform packets";
    }
    leaf conform-rate {
        type uint64;
        units "bits-per-second";
        description "Traffic Rate measured as conforming";
    }
    leaf exceed-pkts {
        type uint64;
        description "Number of packets counted as exceeding";
    }
    leaf exceed-bytes {
        type uint64;
        description "";
    }
}
leaf exceed-rate {
  type uint64;
  units "bits-per-second";
  description
    "Traffic Rate measured as exceeding";
}
leaf violate-pkts {
  type uint64;
  description
    "Number of packets counted as violating";
}
leaf violate-bytes {
  type uint64;
  description
    "Bytes of packets counted as violating";
}
leaf violate-rate {
  type uint64;
  units "bits-per-second";
  description
    "Traffic Rate measured as violating";
}
leaf meter-drop-pkts {
  type uint64;
  description
    "Number of packets dropped by meter";
}
leaf meter-drop-bytes {
  type uint64;
  description
    "Bytes of packets dropped by meter";
}

augment "/if:interfaces/if:interface" {
  description
    "Augments Qos Target Entry to Interface module";
  container qos-interface-statistics {
    config false;
    description
      "Qos Interface statistics";
    list qos-stats-per-direction {
      description
        "Qos Interface statistics for ingress or egress direction";
    }
  }
}
leaf direction {
  type identityref {
    base direction;
  }
  description
  "Direction for the traffic flow either inbound or outbound";
}

list qos-classifier-statistics {
  description
  "Classifier Statistics for each Classifier Entry in a Policy applied in a particular direction";
  leaf policy-name {
    type string;
    description
    "Policy entry name";
  }
  leaf classifier-entry-name {
    type string;
    description
    "Classifier Entry Name";
  }
  uses classifier-entry-stats;
}

list qos-named-statistics {
  config false;
  description
  "Matched Statistics for a statistics-name";
  leaf stats-name {
    type string;
    description
    "Statistics name";
  }
  uses qos-named-stats;
}

list metering-statistics {
  config false;
  description
  "Statistics for each Meter associated with the Policy";
  reference
  "RFC2697: A Single Rate Three Color Marker, RFC2698: A Two Rate Three Color Marker";
  leaf policy-name {
    type string;
    description
    "Policy entry name";
  }
}
leaf meter-id {
  type string;
  description
    "Meter Identifier";
} uses meter-stats;

list queueing-statistics {
  config false;
  description
    "Statistics for each Queue associated with the Policy";
  leaf policy-name {
    type string;
    description
      "Policy entry name";
  }
  leaf queue-id {
    type string;
    description
      "Queue Identifier";
  }
  uses queuing-stats;
}

6. Security Considerations

7. Acknowledgement

8. References

8.1. Normative References

[I-D.ascheid-rtgwg-qos-model]

8.2. Informative References


Authors' Addresses

Aseem Choudhary
Cisco Systems
170 W. Tasman Drive
San Jose, CA 95134
US

Email: asechoud@cisco.com
Ing-Wher Chen
Jabil

Email: ing-wher_chen@jabil.com
Abstract

This document describes a protocol-independent method that employs two bits to allow endpoints to signal packet loss in a way that can be used by network devices to measure and locate the source of the loss. The signaling method applies to all protocols with a protocol-specific way to identify packet loss. The method is especially valuable when applied to protocols that encrypt transport header and do not allow an alternative method for loss detection.

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 21, 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
1. Introduction

Packet loss is a pervasive problem of day-to-day network operation, and proactively detecting, measuring, and locating it is crucial to maintaining high QoS and timely resolution of crippling end-to-end throughput issues. To this effect, in a TCP-dominated world, network operators have been heavily relying on information present in the clear in TCP headers: sequence and acknowledgment numbers and SACKs when enabled (see [RFC8517]). These allow for quantitative estimation of packet loss by passive on-path observation, and the lossy segment (upstream or downstream from the observation point) can be quickly identified by moving the passive observer around.

With encrypted protocols, the equivalent transport headers are encrypted and passive packet loss observation is not possible, as described in [TRANSPORT-ENCRYPT].
Since encrypted protocols could be routed by the network differently, and the fraction of Internet traffic delivered using encrypted protocols is increasing every year, it is imperative to measure packet loss experienced by encrypted protocol users directly instead of relying on measuring TCP loss between similar endpoints.

Following the recommendation in [RFC8558] of making path signals explicit, this document proposes adding two explicit loss bits to the clear portion of the protocol headers to restore network operators’ ability to maintain high QoS for users of encrypted protocols. These bits can be added to an unencrypted portion of a header belonging to any protocol layer, e.g. IP (see [IP]) and IPv6 (see [IPv6]) headers or extensions, UDP surplus space (see [UDP-OPTIONS] and [UDP-SURPLUS]), reserved bits in a QUIC v1 header (see [QUIC-TRANSPORT]).

2. Notational Conventions

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. Loss Bits

The proposal introduces two bits that are to be present in every packet capable of loss reporting. These are packets that include protocol headers with the loss bits. Only loss of packets capable of loss reporting is reported using loss bits.

Whenever this specification refers to packets, it is referring only to packets capable of loss reporting.

- **Q**: The "sQuare signal" bit is toggled every $N$ outgoing packets as explained below in Section 3.1.

- **L**: The "Loss event" bit is set to 0 or 1 according to the Unreported Loss counter, as explained below in Section 3.2.

Each endpoint maintains appropriate counters independently and separately for each connection (each subflow for multipath connections).

3.1. Setting the sQuare Bit on Outgoing Packets

The sQuare Value is initialized to the Initial Q Value (0 or 1) and is reflected in the Q bit of every outgoing packet. The sQuare value is inverted after sending every $N$ packets ($Q$ Period is $2\times N$). The Q bit represents "packet color" as defined by [RFC8321].
The choice of the Initial Q Value and Q Period is determined by the protocol containing Q and L bits. For example, the values can be protocol constants (e.g. "Initial Q Value" is 0, and "Q Period" is 128), or they can be set explicitly for each connection (e.g. "Initial Q Value" is whatever value the initial packet has, and "Q Period" is set per a dedicated TCP option on SYN and SYN/ACK).

Observation points can estimate the upstream losses by counting the number of packets during a half period of the square signal, as described in Section 4.

3.2. Setting the Loss Event Bit on Outgoing Packets

The Unreported Loss counter is initialized to 0, and L bit of every outgoing packet indicates whether the Unreported Loss counter is positive (L=1 if the counter is positive, and L=0 otherwise).

The value of the Unreported Loss counter is decremented every time a packet with L=1 is sent.

The value of the Unreported Loss counter is incremented for every packet that the protocol declares lost, using whatever loss detection machinery the protocol employs. If the protocol is able to rescind the loss determination later, the Unreported Loss counter SHOULD NOT be decremented due to the rescission.

Observation points can estimate the end-to-end loss, as determined by the upstream endpoint’s loss detection machinery, by counting packets in this direction with a L bit equal to 1, as described in Section 4.

4. Using the Loss Bits for Passive Loss Measurement

There are three sources of observable loss:

- _upstream loss_ - loss between the sender and the observation point (Section 4.2)
- _downstream loss_ - loss between the observation point and the destination (Section 4.4)
- _observer loss_ - loss by the observer itself that does not cause downstream loss (Section 4.5)

The upstream and downstream loss together constitute _end-to-end loss_ (Section 4.1).

The Q and L bits allow detection and measurement of the types of loss listed above.
4.1. End-To-End Loss

The Loss Event bit allows an observer to calculate the end-to-end loss rate by counting packets with L bit value of 0 and 1 for a given connection. The end-to-end loss rate is the fraction of packets with L=1.

The simplifying assumption here is that upstream loss affects packets with L=0 and L=1 equally. This may be a simplification, if some loss is caused by tail-drop in a network device. If the sender congestion controller reduces the packet send rate after loss, there may be a sufficient delay before sending packets with L=1 that they have a greater chance of arriving at the observer.

4.2. Upstream Loss

Blocks of N (half of Q Period) consecutive packets are sent with the same value of the Q bit, followed by another block of N packets with inverted value of the Q bit. Hence, knowing the value of N, an on-path observer can estimate the amount of upstream loss after observing at least N packets. If "p" is the average number of packets in a block of packets with the same Q value, then the upstream loss is "1-p/N".

The observer needs to be able to tolerate packet reordering that can blur the edges of the square signal.

The Q Period needs to be chosen carefully, since the observation could become too unreliable in case of packet reordering and loss if Q Period is too small. However, when Q Period is too large, connections that send fewer than half Q Period packets do not yield a useful upstream loss measurement.

The observer needs to differentiate packets as belonging to different connections, since they use independent counters.

4.3. Correlating End-to-End and Upstream Loss

Upstream loss is calculated by observing the actual packets that did not suffer the upstream loss. End-to-end loss, however, is calculated by observing subsequent packets after the sender’s protocol detected the loss. Hence, end-to-end loss is generally observed with a delay of between 1 RTT (loss declared due to multiple duplicate acknowledgments) and 1 RTO (loss declared due to a timeout) relative to the upstream loss.

The connection RTT can sometimes be estimated by timing protocol handshake messages. This RTT estimate can be greatly improved by
observing a dedicated protocol mechanism for conveying RTT information, such as the Latency Spin bit of [QUIC-TRANSPORT].

Whenever the observer needs to perform a computation that uses both upstream and end-to-end loss rate measurements, it SHOULD use upstream loss rate leading the end-to-end loss rate by approximately 1 RTT. If the observer is unable to estimate RTT of the connection, it should accumulate loss measurements over time periods of at least 4 times the typical RTT for the observed connections.

If the calculated upstream loss rate exceeds the end-to-end loss rate calculated in Section 4.1, then either the Q Period is too short for the amount of packet reordering or there is observer loss, described in Section 4.5. If this happens, the observer SHOULD adjust the calculated upstream loss rate to match end-to-end loss rate.

4.4. Downstream Loss

Because downstream loss affects only those packets that did not suffer upstream loss, the end-to-end loss rate ("e") relates to the upstream loss rate ("u") and downstream loss rate ("d") as 

\[(1-u)(1-d)=1-e\]. Hence, 

\[d=(e-u)/(1-u)\].

4.5. Observer Loss

A typical deployment of a passive observation system includes a network tap device that mirrors network packets of interest to a device that performs analysis and measurement on the mirrored packets. The observer loss is the loss that occurs on the mirror path.

Observer loss affects upstream loss rate measurement since it causes the observer to account for fewer packets in a block of identical Q bit values (see ((upstreamloss))). The end-to-end loss rate measurement, however, is unaffected by the observer loss, since it is a measurement of the fraction of packets with the set L bit value, and the observer loss would affect all packets equally (see Section 4.1).

The need to adjust the upstream loss rate down to match end-to-end loss rate as described in Section 4.3 is a strong indication of the observer loss, whose magnitude is between the amount of such adjustment and the entirety of the upstream loss measured in Section 4.2.
5. Ossification Considerations

Accurate loss information is not critical to the operation of any protocol, though its presence for a sufficient number of connections is important for the operation of the networks.

The loss bits are amenable to "greasing" described in [GREASE], if the protocol designers are not ready to dedicate (and ossify) bits used for loss reporting to this function. The greasing could be accomplished similarly to the Latency Spin bit greasing in [QUIC-TRANSPORT]. Namely, implementations could decide that a fraction of connections should not encode loss information in the loss bits and, instead, the bits would be set to arbitrary values. The observers would need to be ready to ignore connections with loss information more resembling noise than the expected signal.

6. Security Considerations

Passive loss observation has been a part of the network operations for a long time, so exposing loss information to the network does not add new security concerns.

7. Privacy Considerations

Guarding user’s privacy is an important goal for modern protocols and protocol extensions per [RFC7285]. While an explicit loss signal - a preferred way to share loss information per [RFC8558] - helps to minimize unintentional exposure of additional information, implementations of loss reporting must ensure that loss information does not compromise protocol’s privacy goals.

For example, [QUIC-TRANSPORT] allows changing Connection IDs in the middle of a connection to reduce the likelihood of a passive observer linking old and new subflows to the same device. A QUIC implementation would need to reset all counters when it changes the destination (IP address or UDP port) or the Connection ID used for outgoing packets. It would also need to avoid incrementing Unreported Loss counter for loss of packets sent to a different destination or with a different Connection ID.

8. IANA Considerations

This document makes no request of IANA.
9. Change Log

9.1. Since version 00
   - Addressed review comments
   - Improved guidelines for privacy protections for QIUC

10. Acknowledgments

The sSquare bit was originally suggested by Kazuho Oku in early proposals for loss measurement.

11. References

11.1. Normative References


11.2. Informative References


Authors' Addresses

Alexandre Ferrieux
Orange Labs
EMail: alexandre.ferrieux@orange.com

Isabelle Hamchaoui
Orange Labs
EMail: isabelle.hamchaoui@orange.com
Igor Lubashev
Akamai Technologies
150 Broadway
Cambridge, MA  1122
USA

EMail: ilubashe@akamai.com
Some Congestion Experienced One and Two-Flow Tests

draft-heist-tsvwg-sce-one-and-two-flow-tests-00

Abstract

This note presents one and two-flow test results for the SCE (Some Congestion Experienced) reference implementation. These tests are not intended to be a comprehensive real-world evaluation of SCE, but an illustration of SCE’s influence on basic TCP metrics in a controlled environment.

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 4 January 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.
1. Introduction

SCE provides early and proportional feedback to the CC (congestion control) algorithms for transport protocols, including but not limited to TCP. The [sce-repo] is a Linux kernel modified to support SCE, including:

* Enhancements to Linux’s Cake (Common Applications Kept Enhanced) AQM to support SCE signaling

* Modifications to the TCP receive path to reflect SCE signals back to the sender

* The addition of three new TCP CC algorithms that modify the originals to add SCE support: Reno-SCE, DCTCP-SCE and Cubic-SCE (work in progress as of this writing)
In this note we run one and two-flow TCP tests across a range of simulated path bandwidths and RTTs. One-flow tests measure SCE’s impact on TCP throughput and TCP RTT. Two-flow tests evaluate fairness between and among several SCE and non-SCE TCP implementations, while making several adjustments to Cake’s SCE and fair queueing parameters.

It is recognized that these tests do not simulate real-world conditions, and will not be an indication of how SCE will perform in all situations. However, they serve as fundamental tests for the SCE reference implementation. Once the behavior in these tests is well understood and theory and experiment are in agreement, additional complexity can be added to the test procedures with the confidence that the reference implementation’s fundamentals are sound.

2. Terminology

The following terminology is used in this document:

* Path Bandwidth or Cake-limited Bandwidth: The available bandwidth between the sender and receiver, as controlled by Cake on the middlebox, and set by Cake’s bandwidth parameter.

* Path RTT or just RTT (in context): The approximate minimum round-trip time of packets that go from the sender to receiver and back when the path is unloaded.

* Netem bi-directional delay: The total path RTT added by netem in both directions.

* TCP throughput and TCP RTT: Well-known terms that apply specifically to the flows under test.

3. Test Tools and Environment

The [Flent] tool is used for all tests. Flent uses netperf for its TCP tests, and allows for test batches, plotting, the recording of results and the collection of metadata in JSON format [RFC8259]. Flent both captures the measured TCP throughput from netperf, and simultaneously uses the [ss] tool in Linux to passively monitor TCP RTT.

All tests are performed using a three node dumbbell topology:

```
+---------+    +----------+    +----------+
 | Sender  |-------| Middlebox |-------| Receiver |
+---------+    +----------+    +----------+
```
Figure 1: Test topology

* Sender: Runs Flent and sends data to the receiver

* Middlebox:
  - Acts as a router between the sender and receiver
  - Runs Cake on egress of both interfaces for queue management and SCE signaling
  - Runs netem on the ingress of both interfaces for delay simulation, splitting the total delay in half for each interface

* Receiver: Receives data from and reflects SCE signals back to the sender via the ESCE (Echo Some Congestion Experienced) bit

All nodes run the SCE reference implementation kernel as of commit 56915a82 (2019-06-20), and are connected directly via Gigabit Ethernet.

4. Tests

The tests are implemented with a Flent batch file to drive netperf and re-configure Cake and netem on the middlebox with various parameters. Scripts post-process the results and create csv and markdown tables for external use, including by this document.

Unless otherwise mentioned, measurements are obtained from TCP flows from start to finish, not at steady state. This allows for some discussion of the differences in TCP CC algorithm behavior during slow start and congestion avoidance. Typically, each test is run long enough to obtain a reasonable approximation of steady state throughput, but in a few high BDP cases slow start accounts for a significant portion of the test length. When relevant to the analysis of the results, this is stated in the text.

5. Results and Analysis

5.1. One-Flow Tests

The goal of the one-flow tests is to analyze the impact of SCE on the TCP throughput and TCP RTT of single TCP flows across a range of simulated path bandwidths and RTTs. What follows is an analysis of the results. See Section a.1 and Section a.2 for the raw results for TCP throughput and TCP RTT, respectively.
5.1.1. Reno-SCE TCP Throughput

The following table shows the difference in TCP throughput for Reno-SCE vs Reno across the tested range of simulated path bandwidths and RTTs:

|     | 0   | 2   | 5    | 10   | 20   | 40   | 80   | 160  |
|-----+-----+-----+------|------|------|------|------|------|
| 0.5 | 0.000 | 0.000 | 0.000 | 0.000 | -0.100 | -0.180 | -0.160 |
| 1   | 0.000 | 0.000 | -0.010 | -0.120 | -0.210 | -0.280 | -0.280 |
| 2   | 0.000 | 0.000 | -0.010 | -0.120 | -0.210 | -0.375 | -0.370 | -0.405 |
| 5   | 0.000 | 0.052 | 0.142 | 0.228 | -0.178 | -0.032 | -0.006 | 0.016 |
| 10  | 0.000 | 0.071 | 0.139 | 0.143 | -0.046 | 0.062 | 0.116 | 0.168 |
| 25  | 0.000 | -0.006 | 0.010 | 0.073 | 0.143 | 0.196 | 0.259 |
| 50  | 0.000 | 0.001 | 0.029 | 0.102 | 0.169 | 0.235 | 0.272 |
| 100 | 0.000 | 0.004 | 0.043 | 0.103 | 0.215 | 0.238 | 0.289 |

Table 1: Difference in TCP Throughput (reno-sce - reno), normalized to Cake-limited Bandwidth; Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

From the above TCP throughput differences we can observe:

1. Improved utilization for SCE at sufficiently high BDPs. This is due to SCE’s proportional congestion signals, which can significantly reduce the classic Reno throughput sawtooth by making drops or CE marks rare to non-existent. The utilization improvement increases with BDP largely because the TCP window recovery time after a drop or CE mark increases with BDP, deepening the sawtooth.

2. Significant under-utilization at bandwidths <= 10Mbit, which tends to worsen as path RTT increases. Investigation is underway as to the source of this. These drops in utilization are however also accompanied by drops in TCP RTT (see below).
### 5.1.2. Reno-SCE TCP RTT

The following table shows the difference in TCP RTT for Reno-SCE vs Reno across the tested range of simulated path bandwidths and RTTs:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>2</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>80</th>
<th>160</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>-88.41</td>
<td>-87.51</td>
<td>-88.43</td>
<td>-88.59</td>
<td>-62.37</td>
<td>-52.56</td>
<td>-42.23</td>
<td>-31.93</td>
</tr>
<tr>
<td>1</td>
<td>-42.30</td>
<td>-41.94</td>
<td>-42.65</td>
<td>-33.03</td>
<td>-32.92</td>
<td>-27.92</td>
<td>-19.13</td>
<td>-13.20</td>
</tr>
<tr>
<td>10</td>
<td>-2.46</td>
<td>-2.75</td>
<td>-2.95</td>
<td>-3.91</td>
<td>-2.44</td>
<td>-1.42</td>
<td>-0.54</td>
<td>-0.41</td>
</tr>
<tr>
<td>25</td>
<td>-1.87</td>
<td>-2.09</td>
<td>-2.53</td>
<td>-1.74</td>
<td>-0.42</td>
<td>0.01</td>
<td>0.19</td>
<td>-0.01</td>
</tr>
<tr>
<td>50</td>
<td>-2.00</td>
<td>-2.18</td>
<td>-1.48</td>
<td>-0.31</td>
<td>0.59</td>
<td>0.98</td>
<td>0.86</td>
<td>0.57</td>
</tr>
<tr>
<td>100</td>
<td>-1.87</td>
<td>-1.67</td>
<td>-1.04</td>
<td>0.06</td>
<td>0.95</td>
<td>1.42</td>
<td>1.41</td>
<td>1.15</td>
</tr>
</tbody>
</table>

Table 2: Difference in TCP RTT (reno-sce - reno); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

From the above TCP RTT differences we can observe:

1. Significant reductions in TCP RTT for Reno-SCE at most BDPs. SCE’s proportional congestion signals are aiding the sender in managing queue lengths.

2. Greater reductions in TCP RTT at lower path bandwidths and RTTs. This is because Reno-linear growth becomes relatively larger as the path bandwidth decreases. This results in greater queue growth in the interval before AQM activates, and a subsequently longer drain time after congestion is signaled.

3. A slight increase in TCP RTT for SCE at high BDPs. This has been observed to have a possible connection to the ESCE (Echo Some Congestion Experienced) feedback strategy implemented on the TCP receive side. Research and optimization in this area is ongoing.
5.1.3. DCTCP-SCE TCP Throughput

The following table shows the difference in TCP throughput for DCTCP-SCE vs DCTCP across the tested range of simulated path bandwidths and RTTs:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>2</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>80</th>
<th>160</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>-0.120</td>
<td>-0.220</td>
<td>-0.400</td>
</tr>
<tr>
<td>1</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>-0.120</td>
<td>-0.260</td>
<td>-0.420</td>
<td>-0.570</td>
</tr>
<tr>
<td>2</td>
<td>0.000</td>
<td>0.000</td>
<td>-0.015</td>
<td>-0.130</td>
<td>-0.285</td>
<td>-0.505</td>
<td>-0.565</td>
<td>-0.705</td>
</tr>
<tr>
<td>5</td>
<td>0.000</td>
<td>-0.038</td>
<td>-0.150</td>
<td>-0.270</td>
<td>-0.374</td>
<td>-0.588</td>
<td>-0.652</td>
<td>-0.620</td>
</tr>
<tr>
<td>10</td>
<td>0.000</td>
<td>-0.087</td>
<td>-0.197</td>
<td>-0.265</td>
<td>-0.266</td>
<td>-0.257</td>
<td>-0.271</td>
<td>-0.134</td>
</tr>
<tr>
<td>25</td>
<td>-0.006</td>
<td>-0.060</td>
<td>-0.131</td>
<td>-0.167</td>
<td>-0.154</td>
<td>-0.133</td>
<td>-0.067</td>
<td>0.110</td>
</tr>
<tr>
<td>50</td>
<td>0.000</td>
<td>-0.001</td>
<td>-0.006</td>
<td>-0.008</td>
<td>0.005</td>
<td>0.012</td>
<td>0.143</td>
<td>0.180</td>
</tr>
<tr>
<td>100</td>
<td>0.000</td>
<td>0.007</td>
<td>0.018</td>
<td>0.031</td>
<td>0.040</td>
<td>0.087</td>
<td>0.151</td>
<td>0.246</td>
</tr>
</tbody>
</table>

Table 3: Difference in TCP Throughput (dctcp-sce - dctcp), normalized to Cake-limited Bandwidth; Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

From the above TCP throughput differences we can observe:

1. Improved utilization for SCE at higher BDPs. At first glance we might assume that this is due to SCE’s feedback signals improving upon DCTCP’s window recovery time, as with Reno, but another significant part of this increase is due to DCTCP-SCE’s steeper ramp during slow start, and a test length that is short relative to the time spent in slow start. The test is 300 seconds long, and at a path bandwidth of 100Mbit and RTT of 160ms, DCTCP takes a full 225 seconds to ramp up to the BDP, while DCTCP-SCE, with its steeper ramp, takes only 80 seconds. That said, steady state throughput at this BDP is around 87Mbit for DCTCP, and around 96Mbit for DCTCP-SCE, an increase of around 10%, so the proportional congestion control signals also play a part in increasing utilization.

2. Significant under-utilization at bandwidths <= 25Mbit, which tends to worsen as path RTT increases. Investigation is underway.
as to the source of this. These drops in utilization are however also accompanied by drops in TCP RTT (see below).

5.1.4. DCTCP-SCE TCP RTT

The following table shows the difference in TCP RTT for DCTCP-SCE vs DCTCP across the tested range of simulated path bandwidths and RTTs:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>2</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>80</th>
<th>160</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>-82.57</td>
<td>-81.21</td>
<td>-81.37</td>
<td>-81.53</td>
<td>-60.53</td>
<td>-70.29</td>
<td>-74.76</td>
<td>-54.33</td>
</tr>
<tr>
<td>1</td>
<td>-41.78</td>
<td>-42.31</td>
<td>-40.43</td>
<td>-30.98</td>
<td>-35.26</td>
<td>-39.27</td>
<td>-29.95</td>
<td>-15.62</td>
</tr>
<tr>
<td>5</td>
<td>-6.65</td>
<td>-7.27</td>
<td>-7.08</td>
<td>-6.45</td>
<td>-5.57</td>
<td>-3.59</td>
<td>-2.19</td>
<td>-1.31</td>
</tr>
<tr>
<td>10</td>
<td>-1.85</td>
<td>-2.11</td>
<td>-2.85</td>
<td>-2.25</td>
<td>-2.16</td>
<td>-1.51</td>
<td>-1.28</td>
<td>-1.12</td>
</tr>
<tr>
<td>25</td>
<td>-2.47</td>
<td>-2.52</td>
<td>-2.94</td>
<td>-3.13</td>
<td>-2.93</td>
<td>-1.63</td>
<td>-0.98</td>
<td>-0.78</td>
</tr>
<tr>
<td>50</td>
<td>-2.63</td>
<td>-3.15</td>
<td>-2.99</td>
<td>-3.12</td>
<td>-3.37</td>
<td>-3.57</td>
<td>-0.81</td>
<td>-0.63</td>
</tr>
<tr>
<td>100</td>
<td>-2.79</td>
<td>-2.81</td>
<td>-2.82</td>
<td>-3.01</td>
<td>-3.26</td>
<td>-2.52</td>
<td>-0.14</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Table 4: Difference in TCP RTT (dctcp-sce - dctcp); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

From the above TCP RTT differences we can observe:

1. A nearly across the board reduction in TCP RTT for DCTCP-SCE. SCE’s proportional congestion signals are aiding the sender in managing queue lengths.

2. Greater reductions in TCP RTT at lower bandwidths and path RTTs. See Section 5.1.2 for an explanation of this.

3. A slight increase in TCP RTT for DCTCP-SCE at 100Mbit / 160ms. See Section 5.1.2 for a likely explanation of this.
5.2.  Two-Flow Tests

The goal of the two-flow tests is to measure fairness between and among SCE and non-SCE TCP flows, through either a single queue or with fair queueing. What follows is a partial analysis of the results, and see Section a.3 through Section a.8 for the raw results tables.

5.2.1.  Single Queue (Cake "flowblind")

Cake’s flowblind parameter disables fair queueing, so that Cake uses only a single queue. This can be used to evaluate single-queue throughput fairness between SCE and non-SCE flows.

5.2.1.1.  Reno vs Reno

It might be useful to remind ourselves that competition between two (or more) flows of the same TCP CC algorithm is usually designed to yield a high degree of throughput fairness in a single queue. Looking at Jain’s fairness index [RFC5166] for two flows across a range of simulated path bandwidths and RTTs shows this to be true for two TCP Reno flows, for example:

```
+-----+-------+-------+-------+-------+
|     | 0     | 10    | 20    | 80    |
+=====+=======+=======+=======+=======+
| 1   | 0.993 | 1.000 | 1.000 | 0.998 |
+-----+-------+-------+-------+-------+
| 5   | 1.000 | 1.000 | 0.998 | 0.998 |
+-----+-------+-------+-------+-------+
| 10  | 1.000 | 0.999 | 0.998 | 1.000 |
+-----+-------+-------+-------+-------+
| 50  | 1.000 | 0.998 | 0.999 | 0.999 |
+-----+-------+-------+-------+-------+
| 100 | 0.998 | 0.999 | 0.999 | 1.000 |
+-----+-------+-------+-------+-------+
```

Table 5: reno vs reno Jain’s fairness index; Columns: netem bi-dir Delay (ms); Rows: Cake Bandwidth (Mbit)

5.2.1.2.  Reno vs Reno-SCE

Now, we do a comparison of Reno vs Reno-SCE in a single queue, while still using Cake’s default SCE signaling ramp:
Table 6: reno vs reno-sce Jain’s fairness index; Columns: netem bi-dir Delay (ms); Rows: Cake Bandwidth (Mbit)

From the above Jain’s fairness index numbers we can observe that while there can be reasonable fairness at very low BDPs, and while there is no starvation, fairness degrades quickly at higher throughputs and RTTs. This is due to the Cake’s default SCE signaling ramp being tuned to provide an early signal of congestion, to avoid CE marks and packet drops. As a result, SCE enabled flows back off in the face of competition, whereas non-SCE flows fill the queue until a drop or CE mark occurs.

5.2.1.3. Cubic vs DCTCP-SCE

Of particular interest to the congestion control community is competition between the commonly used TCP Cubic algorithm and DCTCP-SCE. It is a well-established fact that classic DCTCP will typically out-compete Cubic in a single queue. It would be valuable if there were a way to improve that fairness with SCE.

When comparing Cubic vs DCTCP-SCE using Cake’s default SCE signaling ramp, we can see that while there is no starvation, fairness does degrade fairly rapidly as BDP increases:
As of today, SCE by default does not lead to fairness at all BDPs between SCE and non-SCE flows. However, efforts are ongoing to improve this, and as we can see in Section 5.2.2, Cake’s signaling ramp can be tuned to improve this fairness.

5.2.2. Single Queue (Cake "sce-single")

As we saw in Section 5.2.1, there is room to improve SCE vs non-SCE fairness in a single queue. One way to do this is to change the SCE signaling ramp to reduce or delay SCE signals until closer to the point where CE signals occur. This is the motivation behind the sce-single and sce-thresh Cake parameters.

By default, Cake begins proportionally signaling SCE when a packet’s sojourn time in the queue is greater than half the CoDel target, and reaches 100% SCE signaling when the sojourn time is greater than or equal to the CoDel target. The sce-single parameter delays the start of the ramp until the sojourn reaches the CoDel target itself, while keeping the ramp slope the same. The sce-thresh parameter, while not evaluated here, allows an intermediate ramp between the default and sce-single, using values from 2-1024, with sce-thresh set to 8 yielding a ramp that’s halfway in-between.

It is expected that the result of signaling SCE later is a subsequent increase in TCP RTT, including for single flows. Thus, the adjustment of the SCE signaling ramp is a tradeoff between the increased utilization and reduced TCP RTT possible with SCE, and fairness with non-SCE flows in a single queue. By using sce-single,
we can show the maximum fairness that can be achieved by tuning the
SCE signaling ramp in this way.

5.2.2.1. Reno vs Reno-SCE

Revisiting the Reno vs Reno-SCE comparison that we did in
Section 5.2.1.2, we now run the same test with Cake’s sce-single
parameter set:

|     | 0     | 10    | 20    | 80    |
|-----|=======|=======|=======|=======|
| 1   | 1.000 | 0.998 | 1.000 | 0.987 |
| 5   | 0.997 | 0.952 | 0.785 | 0.819 |
| 10  | 0.997 | 0.950 | 0.790 | 0.994 |
| 50  | 1.000 | 0.986 | 0.980 | 0.977 |
| 100 | 0.988 | 0.901 | 0.932 | 0.987 |

Table 8: reno vs reno-sce Jain’s
fairness index; Columns: netem bi-
dir Delay (ms); Rows: Cake
Bandwidth (Mbit)

We can see that single queue fairness has improved considerably.
While there are results at a few bandwidth-delay combinations that
are still under investigation, single queue fairness between Reno and
Reno-SCE has largely been achieved.

5.2.2.2. Cubic vs DCTCP-SCE

Revisiting the Cubic vs DCTCP-SCE comparison that we did in
Section 5.2.1.3, we now run the same test with Cake’s sce-single
parameter set:
Table 9: cubic vs dctcp-sce Jain’s fairness index; Columns: netem bi-
dir Delay (ms); Rows: Cake Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.993</td>
<td>0.999</td>
<td>0.995</td>
<td>0.963</td>
</tr>
<tr>
<td>5</td>
<td>1.000</td>
<td>0.898</td>
<td>0.765</td>
<td>0.741</td>
</tr>
<tr>
<td>10</td>
<td>0.975</td>
<td>0.893</td>
<td>0.733</td>
<td>0.885</td>
</tr>
<tr>
<td>50</td>
<td>0.999</td>
<td>0.752</td>
<td>0.799</td>
<td>0.923</td>
</tr>
<tr>
<td>100</td>
<td>0.944</td>
<td>0.714</td>
<td>0.661</td>
<td>0.650</td>
</tr>
</tbody>
</table>

While it can be seen that there is an almost across the board improvement vs Cake’s default SCE ramp, there are still bandwidth-delay combinations that do not yield a sufficient level of fairness in a single queue. Work is ongoing to improve this.

5.2.3. Fair Queue (Cake "triple-isolate")

Cake’s default triple-isolate fairness mode provides fairness among flows and a combination of source and destination IP addresses. For our purposes, this will effectively serve as fair queueing among flows, as we are only using one IP on each of the source and destination hosts.

With fair queueing, we expect to achieve a high level of throughput fairness at most bandwidth-delay combinations.

5.2.3.1. Cubic vs DCTCP-SCE

Revisiting the Cubic vs DCTCP-SCE comparisons that we did in Section 5.2.1.3 and Section 5.2.2.2, we now run the same test with Cake’s fair queueing enabled via triple-isolate:
Table 10: cubic vs dctcp-sce Jain’s fairness index; Columns: netem bi-dir Delay (ms); Rows: Cake Bandwidth (Mbit)

With fair queueing, fairness is achieved for a broad range of bandwidth-delay combinations. We do see a consistent and narrow deviation at around 160ms between 5 and 10Mbit, which happens in competition between SCE and non-SCE flows, regardless of the exact algorithms in use. The cause of this will be investigated.

5.2.3.2. DCTCP vs DCTCP-SCE

As another example of SCE vs non-SCE competition with fair queueing enabled, here we compare DCTCP vs DCTCP-SCE:

Table 11: dctcp vs dctcp-sce Jain’s fairness index; Columns: netem bi-dir Delay (ms); Rows: Cake Bandwidth (Mbit)
As with Cubic vs DCTCP-SCE, a high degree of fairness is achieved for most bandwidth-delay combinations, yet we do see the same small gap in fairness at around 160ms between 5 and 10Mbit as we see with Cubic vs DCTCP-SCE and other SCE vs non-SCE flow tests with fair queueing. It is expected that further investigation into this will lead to a solution.

6. Security Considerations

There are no known security considerations introduced by this note.

7. IANA Considerations

This document has no IANA actions.

8. Acknowledgments

Many thanks go out to Toke Hoiland-Jorgensen for making several key changes to the Flent tool.

9. Informative References


Appendix A. Appendix (Raw Results Tables)
A.1. One-Flow TCP Throughput

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>2</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>80</th>
<th>160</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.47</td>
<td>0.48</td>
<td>0.44</td>
</tr>
<tr>
<td>1</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>0.95</td>
<td>0.94</td>
<td>0.87</td>
<td>0.82</td>
</tr>
<tr>
<td>2</td>
<td>1.91</td>
<td>1.91</td>
<td>1.91</td>
<td>1.91</td>
<td>1.91</td>
<td>1.83</td>
<td>1.82</td>
<td>1.52</td>
</tr>
<tr>
<td>5</td>
<td>4.78</td>
<td>4.78</td>
<td>4.78</td>
<td>4.78</td>
<td>4.76</td>
<td>4.57</td>
<td>4.32</td>
<td>3.28</td>
</tr>
<tr>
<td>10</td>
<td>9.56</td>
<td>9.56</td>
<td>9.56</td>
<td>9.55</td>
<td>9.43</td>
<td>8.85</td>
<td>8.23</td>
<td>6.51</td>
</tr>
<tr>
<td>25</td>
<td>23.91</td>
<td>23.91</td>
<td>23.89</td>
<td>23.82</td>
<td>23.17</td>
<td>22.13</td>
<td>20.83</td>
<td>16.07</td>
</tr>
<tr>
<td>50</td>
<td>47.81</td>
<td>47.80</td>
<td>47.71</td>
<td>47.43</td>
<td>45.87</td>
<td>44.12</td>
<td>40.44</td>
<td>31.64</td>
</tr>
<tr>
<td>100</td>
<td>95.64</td>
<td>95.50</td>
<td>95.17</td>
<td>94.39</td>
<td>92.52</td>
<td>87.73</td>
<td>77.25</td>
<td>82.38</td>
</tr>
</tbody>
</table>

Table 12: cubic Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>2</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>80</th>
<th>160</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.47</td>
<td>0.45</td>
<td>0.40</td>
</tr>
<tr>
<td>1</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>0.95</td>
<td>0.94</td>
<td>0.89</td>
<td>0.82</td>
</tr>
<tr>
<td>2</td>
<td>1.91</td>
<td>1.91</td>
<td>1.91</td>
<td>1.91</td>
<td>1.90</td>
<td>1.85</td>
<td>1.64</td>
<td>1.54</td>
</tr>
<tr>
<td>5</td>
<td>4.78</td>
<td>4.78</td>
<td>4.78</td>
<td>4.78</td>
<td>4.76</td>
<td>4.64</td>
<td>3.91</td>
<td>3.88</td>
</tr>
<tr>
<td>10</td>
<td>9.56</td>
<td>9.56</td>
<td>9.56</td>
<td>9.52</td>
<td>9.52</td>
<td>8.90</td>
<td>8.04</td>
<td>7.44</td>
</tr>
<tr>
<td>25</td>
<td>23.91</td>
<td>23.91</td>
<td>23.89</td>
<td>23.38</td>
<td>21.65</td>
<td>19.51</td>
<td>17.63</td>
<td>14.50</td>
</tr>
<tr>
<td>50</td>
<td>47.82</td>
<td>47.81</td>
<td>47.74</td>
<td>46.21</td>
<td>42.17</td>
<td>38.31</td>
<td>31.97</td>
<td>28.85</td>
</tr>
<tr>
<td>100</td>
<td>95.63</td>
<td>95.53</td>
<td>94.91</td>
<td>90.65</td>
<td>83.51</td>
<td>69.92</td>
<td>63.70</td>
<td>53.89</td>
</tr>
</tbody>
</table>

Table 13: reno Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
### Table 14: reno-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>2</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>80</th>
<th>160</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.42</td>
<td>0.36</td>
<td>0.32</td>
</tr>
<tr>
<td>1</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>0.84</td>
<td>0.73</td>
<td>0.61</td>
</tr>
<tr>
<td>2</td>
<td>1.91</td>
<td>1.91</td>
<td>1.89</td>
<td>1.66</td>
<td>1.42</td>
<td>1.10</td>
<td>0.90</td>
<td>0.73</td>
</tr>
<tr>
<td>5</td>
<td>4.78</td>
<td>4.52</td>
<td>4.07</td>
<td>3.64</td>
<td>3.75</td>
<td>3.75</td>
<td>3.85</td>
<td>3.37</td>
</tr>
<tr>
<td>10</td>
<td>9.56</td>
<td>8.85</td>
<td>8.17</td>
<td>8.09</td>
<td>8.44</td>
<td>8.66</td>
<td>8.60</td>
<td>8.11</td>
</tr>
<tr>
<td>25</td>
<td>23.90</td>
<td>23.84</td>
<td>23.74</td>
<td>23.62</td>
<td>23.47</td>
<td>23.09</td>
<td>22.54</td>
<td>20.97</td>
</tr>
<tr>
<td>50</td>
<td>47.81</td>
<td>47.81</td>
<td>47.78</td>
<td>47.66</td>
<td>46.74</td>
<td>43.70</td>
<td>42.44</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>95.64</td>
<td>95.58</td>
<td>95.30</td>
<td>94.98</td>
<td>93.82</td>
<td>91.37</td>
<td>87.46</td>
<td>82.83</td>
</tr>
</tbody>
</table>

### Table 15: dctcp Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>2</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>80</th>
<th>160</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.47</td>
<td>0.47</td>
</tr>
<tr>
<td>1</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>0.93</td>
<td>0.91</td>
</tr>
<tr>
<td>2</td>
<td>1.91</td>
<td>1.91</td>
<td>1.91</td>
<td>1.91</td>
<td>1.91</td>
<td>1.88</td>
<td>1.78</td>
<td>1.78</td>
</tr>
<tr>
<td>5</td>
<td>4.78</td>
<td>4.78</td>
<td>4.78</td>
<td>4.77</td>
<td>4.74</td>
<td>4.68</td>
<td>4.54</td>
<td>4.10</td>
</tr>
<tr>
<td>10</td>
<td>9.56</td>
<td>9.56</td>
<td>9.55</td>
<td>9.52</td>
<td>9.41</td>
<td>9.23</td>
<td>8.86</td>
<td>7.23</td>
</tr>
<tr>
<td>25</td>
<td>23.91</td>
<td>23.90</td>
<td>23.84</td>
<td>23.68</td>
<td>23.25</td>
<td>22.34</td>
<td>20.33</td>
<td>15.86</td>
</tr>
<tr>
<td>50</td>
<td>47.82</td>
<td>47.75</td>
<td>47.55</td>
<td>46.96</td>
<td>45.33</td>
<td>43.93</td>
<td>34.63</td>
<td>31.54</td>
</tr>
<tr>
<td>100</td>
<td>95.62</td>
<td>94.89</td>
<td>93.59</td>
<td>91.78</td>
<td>90.26</td>
<td>81.84</td>
<td>70.42</td>
<td>57.89</td>
</tr>
</tbody>
</table>

### Table 16: dctcp-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>2</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>80</th>
<th>160</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.42</td>
<td>0.36</td>
<td>0.27</td>
</tr>
<tr>
<td>1</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
<td>0.84</td>
<td>0.69</td>
<td>0.51</td>
<td>0.34</td>
</tr>
<tr>
<td>2</td>
<td>1.91</td>
<td>1.91</td>
<td>1.88</td>
<td>1.65</td>
<td>1.34</td>
<td>0.87</td>
<td>0.65</td>
<td>0.37</td>
</tr>
<tr>
<td>5</td>
<td>4.78</td>
<td>4.59</td>
<td>4.03</td>
<td>3.42</td>
<td>2.87</td>
<td>1.74</td>
<td>1.28</td>
<td>1.00</td>
</tr>
<tr>
<td>10</td>
<td>9.56</td>
<td>8.69</td>
<td>7.58</td>
<td>6.87</td>
<td>6.75</td>
<td>6.66</td>
<td>6.15</td>
<td>5.89</td>
</tr>
<tr>
<td>50</td>
<td>47.81</td>
<td>47.68</td>
<td>47.27</td>
<td>46.54</td>
<td>45.59</td>
<td>44.52</td>
<td>41.77</td>
<td>40.53</td>
</tr>
<tr>
<td>100</td>
<td>95.64</td>
<td>95.59</td>
<td>95.34</td>
<td>94.90</td>
<td>94.30</td>
<td>90.53</td>
<td>85.48</td>
<td>82.53</td>
</tr>
</tbody>
</table>

### Table 17: cubic Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>2</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>80</th>
<th>160</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>121.38</td>
<td>120.07</td>
<td>119.91</td>
<td>121.37</td>
<td>128.70</td>
<td>189.50</td>
<td>230.34</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>68.40</td>
<td>68.16</td>
<td>68.10</td>
<td>69.33</td>
<td>72.97</td>
<td>88.11</td>
<td>116.66</td>
<td>191.18</td>
</tr>
<tr>
<td>2</td>
<td>34.80</td>
<td>35.01</td>
<td>40.33</td>
<td>40.17</td>
<td>52.74</td>
<td>62.05</td>
<td>100.71</td>
<td>175.00</td>
</tr>
<tr>
<td>5</td>
<td>13.79</td>
<td>15.92</td>
<td>18.43</td>
<td>23.83</td>
<td>31.04</td>
<td>49.07</td>
<td>87.79</td>
<td>167.49</td>
</tr>
<tr>
<td>10</td>
<td>7.08</td>
<td>8.95</td>
<td>12.58</td>
<td>17.39</td>
<td>26.12</td>
<td>44.73</td>
<td>84.60</td>
<td>164.49</td>
</tr>
<tr>
<td>25</td>
<td>6.36</td>
<td>8.29</td>
<td>11.31</td>
<td>15.33</td>
<td>24.06</td>
<td>43.37</td>
<td>82.68</td>
<td>162.72</td>
</tr>
<tr>
<td>50</td>
<td>5.88</td>
<td>8.16</td>
<td>10.70</td>
<td>14.76</td>
<td>23.51</td>
<td>42.82</td>
<td>82.10</td>
<td>161.81</td>
</tr>
<tr>
<td>100</td>
<td>5.99</td>
<td>8.08</td>
<td>10.38</td>
<td>14.42</td>
<td>23.69</td>
<td>42.90</td>
<td>82.16</td>
<td>162.01</td>
</tr>
</tbody>
</table>

A.2. One-Flow TCP RTT
Table 18: reno Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|      | 0  | 2   | 5   | 10  | 20  | 40  | 80  | 160 |
|-----+----+-----+-----+-----+-----+-----+-----+-----|
| 0.5 | 138.77 | 137.84 | 138.91 | 138.92 | 132.94 | 140.51 | 170.48 | 235.32 |
| 1   | 66.82 | 66.68 | 68.93 | 69.02 | 77.11 | 91.69 | 123.15 | 196.59 |
| 2   | 41.00 | 41.25 | 40.81 | 42.24 | 47.24 | 65.51 | 99.34 | 176.74 |
| 5   | 16.25 | 18.37 | 19.18 | 25.13 | 32.06 | 49.61 | 87.61 | 167.39 |
| 10  | 7.68 | 9.86 | 13.14 | 18.75 | 26.74 | 45.54 | 84.50 | 164.45 |
| 25  | 6.27 | 8.26 | 11.46 | 15.22 | 23.65 | 42.84 | 82.59 | 162.69 |
| 50  | 5.99 | 8.08 | 10.34 | 14.11 | 22.86 | 42.22 | 81.97 | 162.08 |
| 100 | 5.87 | 7.64 | 9.85 | 13.62 | 22.64 | 41.96 | 81.73 | 161.76 |

Table 19: reno-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|      | 0  | 2   | 5   | 10  | 20  | 40  | 80  | 160 |
|-----+----+-----+-----+-----+-----+-----+-----+-----|
| 0.5 | 50.36 | 50.33 | 50.48 | 50.33 | 70.57 | 87.95 | 128.25 | 203.39 |
| 1   | 24.52 | 24.74 | 26.28 | 35.99 | 44.19 | 63.77 | 104.02 | 183.39 |
| 2   | 11.91 | 13.58 | 18.62 | 23.01 | 32.70 | 55.39 | 95.68 | 175.24 |
| 5   | 7.07 | 8.90 | 11.67 | 16.76 | 26.43 | 46.12 | 85.92 | 166.15 |
| 10  | 5.22 | 7.11 | 10.19 | 14.84 | 24.30 | 44.12 | 83.96 | 164.04 |
| 25  | 4.40 | 6.17 | 8.93 | 13.48 | 23.23 | 42.85 | 82.78 | 162.68 |
| 50  | 3.99 | 5.90 | 8.86 | 13.80 | 23.45 | 43.20 | 82.83 | 162.65 |
| 100 | 4.00 | 5.97 | 8.81 | 13.68 | 23.59 | 43.38 | 83.14 | 162.91 |

Table 19: reno-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
### Table 20: dctcp Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>2</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>80</th>
<th>160</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>132.20</td>
<td>131.17</td>
<td>131.14</td>
<td>130.92</td>
<td>130.17</td>
<td>157.56</td>
<td>199.65</td>
<td>260.24</td>
</tr>
<tr>
<td>1</td>
<td>66.23</td>
<td>66.69</td>
<td>66.65</td>
<td>66.60</td>
<td>79.38</td>
<td>103.86</td>
<td>135.31</td>
<td>203.32</td>
</tr>
<tr>
<td>2</td>
<td>33.84</td>
<td>34.07</td>
<td>39.03</td>
<td>40.24</td>
<td>52.76</td>
<td>69.25</td>
<td>104.09</td>
<td>183.85</td>
</tr>
<tr>
<td>5</td>
<td>13.67</td>
<td>15.87</td>
<td>18.50</td>
<td>23.05</td>
<td>32.73</td>
<td>53.12</td>
<td>89.63</td>
<td>168.33</td>
</tr>
<tr>
<td>10</td>
<td>6.99</td>
<td>9.06</td>
<td>12.82</td>
<td>17.26</td>
<td>26.75</td>
<td>45.79</td>
<td>85.43</td>
<td>165.10</td>
</tr>
<tr>
<td>25</td>
<td>6.57</td>
<td>8.26</td>
<td>11.25</td>
<td>16.07</td>
<td>25.65</td>
<td>44.11</td>
<td>83.45</td>
<td>163.30</td>
</tr>
<tr>
<td>50</td>
<td>6.22</td>
<td>8.39</td>
<td>10.98</td>
<td>15.83</td>
<td>25.65</td>
<td>45.56</td>
<td>82.73</td>
<td>162.51</td>
</tr>
<tr>
<td>100</td>
<td>6.31</td>
<td>8.27</td>
<td>11.16</td>
<td>16.12</td>
<td>25.88</td>
<td>44.90</td>
<td>82.43</td>
<td>162.05</td>
</tr>
</tbody>
</table>

### Table 21: dctcp-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>2</th>
<th>5</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>80</th>
<th>160</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>49.63</td>
<td>49.96</td>
<td>49.77</td>
<td>49.39</td>
<td>69.64</td>
<td>87.27</td>
<td>124.89</td>
<td>205.91</td>
</tr>
<tr>
<td>1</td>
<td>24.45</td>
<td>24.38</td>
<td>26.22</td>
<td>35.62</td>
<td>44.12</td>
<td>64.59</td>
<td>105.36</td>
<td>187.70</td>
</tr>
<tr>
<td>2</td>
<td>11.90</td>
<td>13.78</td>
<td>18.17</td>
<td>22.66</td>
<td>32.83</td>
<td>56.55</td>
<td>96.72</td>
<td>179.88</td>
</tr>
<tr>
<td>5</td>
<td>7.02</td>
<td>8.60</td>
<td>11.42</td>
<td>16.60</td>
<td>27.16</td>
<td>49.53</td>
<td>90.11</td>
<td>170.64</td>
</tr>
<tr>
<td>10</td>
<td>5.14</td>
<td>6.95</td>
<td>9.97</td>
<td>15.01</td>
<td>24.59</td>
<td>44.28</td>
<td>84.15</td>
<td>163.98</td>
</tr>
<tr>
<td>25</td>
<td>4.10</td>
<td>5.74</td>
<td>8.31</td>
<td>12.94</td>
<td>22.72</td>
<td>42.48</td>
<td>82.47</td>
<td>162.52</td>
</tr>
<tr>
<td>50</td>
<td>3.59</td>
<td>5.24</td>
<td>7.99</td>
<td>12.71</td>
<td>22.28</td>
<td>41.99</td>
<td>81.92</td>
<td>161.88</td>
</tr>
<tr>
<td>100</td>
<td>3.52</td>
<td>5.46</td>
<td>8.34</td>
<td>13.11</td>
<td>22.62</td>
<td>42.38</td>
<td>82.29</td>
<td>162.07</td>
</tr>
</tbody>
</table>
### A.3. Two-Flow TCP Throughput (Cake "flowblind")

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>cubic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.46</td>
<td>0.48</td>
<td>0.48</td>
<td>0.47</td>
<td>0.46</td>
</tr>
<tr>
<td>5</td>
<td>2.41</td>
<td>2.39</td>
<td>2.37</td>
<td>2.27</td>
<td>2.37</td>
<td>2.38</td>
<td>2.37</td>
<td>2.19</td>
</tr>
<tr>
<td>10</td>
<td>4.82</td>
<td>4.57</td>
<td>4.70</td>
<td>3.83</td>
<td>4.74</td>
<td>4.93</td>
<td>4.77</td>
<td>4.83</td>
</tr>
<tr>
<td>50</td>
<td>22.64</td>
<td>23.46</td>
<td>24.09</td>
<td>22.26</td>
<td>25.18</td>
<td>24.19</td>
<td>23.07</td>
<td>20.68</td>
</tr>
<tr>
<td>100</td>
<td>47.66</td>
<td>42.88</td>
<td>46.29</td>
<td>44.10</td>
<td>48.01</td>
<td>52.03</td>
<td>48.29</td>
<td>43.72</td>
</tr>
</tbody>
</table>

Table 22: cubic-cubic Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>cubic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.43</td>
<td>0.46</td>
<td>0.45</td>
<td>0.41</td>
<td>0.53</td>
<td>0.50</td>
<td>0.51</td>
<td>0.51</td>
</tr>
<tr>
<td>5</td>
<td>2.10</td>
<td>2.04</td>
<td>2.46</td>
<td>2.30</td>
<td>2.68</td>
<td>2.71</td>
<td>2.22</td>
<td>2.04</td>
</tr>
<tr>
<td>10</td>
<td>4.23</td>
<td>4.61</td>
<td>4.50</td>
<td>4.16</td>
<td>5.34</td>
<td>4.81</td>
<td>4.83</td>
<td>4.17</td>
</tr>
<tr>
<td>50</td>
<td>22.79</td>
<td>24.60</td>
<td>23.55</td>
<td>26.18</td>
<td>25.03</td>
<td>23.05</td>
<td>23.32</td>
<td>14.70</td>
</tr>
<tr>
<td>100</td>
<td>46.82</td>
<td>51.06</td>
<td>48.73</td>
<td>55.96</td>
<td>48.83</td>
<td>43.85</td>
<td>44.96</td>
<td>29.02</td>
</tr>
</tbody>
</table>

Table 23: cubic-reno Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.55</td>
<td>0.55</td>
<td>0.62</td>
<td>0.66</td>
<td>0.41</td>
<td>0.41</td>
<td>0.34</td>
<td>0.24</td>
</tr>
<tr>
<td>5</td>
<td>2.75</td>
<td>3.63</td>
<td>3.96</td>
<td>3.92</td>
<td>2.03</td>
<td>1.14</td>
<td>0.80</td>
<td>0.46</td>
</tr>
<tr>
<td>10</td>
<td>5.50</td>
<td>7.97</td>
<td>8.24</td>
<td>7.57</td>
<td>4.06</td>
<td>1.57</td>
<td>1.26</td>
<td>1.37</td>
</tr>
<tr>
<td>50</td>
<td>43.53</td>
<td>44.29</td>
<td>42.98</td>
<td>39.36</td>
<td>4.30</td>
<td>3.44</td>
<td>4.25</td>
<td>4.89</td>
</tr>
<tr>
<td>100</td>
<td>91.16</td>
<td>89.89</td>
<td>88.69</td>
<td>76.76</td>
<td>4.46</td>
<td>5.28</td>
<td>5.77</td>
<td>11.79</td>
</tr>
</tbody>
</table>

Table 24: cubic-reno-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.48</td>
<td>0.47</td>
<td>0.40</td>
<td>0.33</td>
<td>0.48</td>
<td>0.48</td>
<td>0.56</td>
<td>0.60</td>
</tr>
<tr>
<td>5</td>
<td>2.30</td>
<td>1.53</td>
<td>1.49</td>
<td>1.70</td>
<td>2.49</td>
<td>3.25</td>
<td>3.26</td>
<td>2.87</td>
</tr>
<tr>
<td>10</td>
<td>3.69</td>
<td>2.77</td>
<td>2.81</td>
<td>2.99</td>
<td>5.88</td>
<td>6.77</td>
<td>6.68</td>
<td>5.96</td>
</tr>
<tr>
<td>100</td>
<td>31.33</td>
<td>30.67</td>
<td>33.63</td>
<td>32.23</td>
<td>64.35</td>
<td>63.57</td>
<td>59.81</td>
<td>55.50</td>
</tr>
</tbody>
</table>

Table 25: cubic-dctcp Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
### Table 26: cubic-dctcp-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|       | 0  | 10 | 20 | 80 | 0  | 10 | 20 | 80 |
|-------+----+----+----+----+----+----+----+----|
| 1     | 0.55 | 0.55 | 0.62 | 0.66 | 0.41 | 0.40 | 0.34 | 0.24 |
| 5     | 2.76 | 3.59 | 3.91 | 4.01 | 2.02 | 1.18 | 0.85 | 0.42 |
| 10    | 5.48 | 7.95 | 8.24 | 7.81 | 4.08 | 1.60 | 1.22 | 1.01 |
| 50    | 43.46 | 44.60 | 43.46 | 39.78 | 4.36 | 3.12 | 3.75 | 4.29 |
| 100   | 91.37 | 90.47 | 88.47 | 77.27 | 4.28 | 4.69 | 5.59 | 7.74 |

### Table 27: reno-reno Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|       | 0  | 10 | 20 | 80 | 0  | 10 | 20 | 80 |
|-------+----+----+----+----+----+----+----+----|
| 1     | 0.52 | 0.48 | 0.47 | 0.44 | 0.44 | 0.48 | 0.48 | 0.48 |
| 5     | 2.39 | 2.36 | 2.23 | 1.98 | 2.39 | 2.37 | 2.41 | 2.15 |
| 10    | 4.74 | 4.86 | 4.79 | 3.92 | 4.82 | 4.51 | 4.40 | 3.93 |
| 50    | 23.88 | 22.77 | 22.88 | 18.78 | 23.95 | 24.90 | 24.03 | 17.50 |
| 100   | 49.89 | 45.68 | 48.07 | 36.99 | 45.76 | 49.17 | 44.65 | 37.37 |
### Table 28: `reno-reno-sce` Mean TCP Throughput (Mbit); Columns: \(\text{netem bi-directional Delay (ms)}\); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>reno</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.56</td>
<td>0.56</td>
<td>0.62</td>
<td>0.66</td>
<td>0.39</td>
<td>0.39</td>
<td>0.33</td>
<td>0.24</td>
</tr>
<tr>
<td>5</td>
<td>2.89</td>
<td>3.43</td>
<td>3.82</td>
<td>3.64</td>
<td>1.89</td>
<td>1.26</td>
<td>0.86</td>
<td>0.49</td>
</tr>
<tr>
<td>10</td>
<td>5.77</td>
<td>7.71</td>
<td>7.91</td>
<td>6.96</td>
<td>3.80</td>
<td>1.78</td>
<td>1.38</td>
<td>1.44</td>
</tr>
<tr>
<td>50</td>
<td>43.10</td>
<td>42.13</td>
<td>39.08</td>
<td>28.40</td>
<td>4.72</td>
<td>5.18</td>
<td>6.99</td>
<td>14.62</td>
</tr>
<tr>
<td>100</td>
<td>90.26</td>
<td>83.33</td>
<td>78.25</td>
<td>62.70</td>
<td>5.40</td>
<td>10.72</td>
<td>13.52</td>
<td>19.41</td>
</tr>
</tbody>
</table>

### Table 29: `reno-dctcp` Mean TCP Throughput (Mbit); Columns: `netem` bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>reno</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.55</td>
<td>0.54</td>
<td>0.42</td>
<td>0.37</td>
<td>0.41</td>
<td>0.42</td>
<td>0.53</td>
<td>0.57</td>
</tr>
<tr>
<td>5</td>
<td>2.60</td>
<td>1.65</td>
<td>1.58</td>
<td>1.44</td>
<td>2.19</td>
<td>3.12</td>
<td>3.15</td>
<td>3.09</td>
</tr>
<tr>
<td>10</td>
<td>4.12</td>
<td>3.28</td>
<td>2.98</td>
<td>2.80</td>
<td>5.44</td>
<td>6.24</td>
<td>6.46</td>
<td>5.99</td>
</tr>
<tr>
<td>50</td>
<td>17.17</td>
<td>17.86</td>
<td>16.27</td>
<td>11.57</td>
<td>30.66</td>
<td>29.71</td>
<td>30.63</td>
<td>28.80</td>
</tr>
<tr>
<td>100</td>
<td>36.09</td>
<td>33.70</td>
<td>28.55</td>
<td>18.60</td>
<td>59.58</td>
<td>60.61</td>
<td>64.04</td>
<td>59.50</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>80</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>80</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>reno</td>
<td>reno</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.57</td>
<td>0.56</td>
<td>0.63</td>
<td>0.66</td>
<td>0.39</td>
<td>0.39</td>
<td>0.33</td>
<td>0.24</td>
</tr>
<tr>
<td>5</td>
<td>2.89</td>
<td>3.74</td>
<td>3.88</td>
<td>3.68</td>
<td>1.89</td>
<td>1.03</td>
<td>0.83</td>
<td>0.47</td>
</tr>
<tr>
<td>10</td>
<td>5.70</td>
<td>7.72</td>
<td>7.94</td>
<td>7.03</td>
<td>3.86</td>
<td>1.77</td>
<td>1.32</td>
<td>1.18</td>
</tr>
<tr>
<td>50</td>
<td>2.39</td>
<td>1.99</td>
<td>1.75</td>
<td>2.39</td>
<td>1.77</td>
<td>1.75</td>
<td>1.52</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>4.78</td>
<td>3.80</td>
<td>3.82</td>
<td>4.23</td>
<td>4.78</td>
<td>4.00</td>
<td>3.79</td>
<td>4.06</td>
</tr>
<tr>
<td>24.24</td>
<td>21.17</td>
<td>24.08</td>
<td>24.69</td>
<td>23.60</td>
<td>26.11</td>
<td>22.91</td>
<td>20.34</td>
<td></td>
</tr>
<tr>
<td>42.25</td>
<td>47.10</td>
<td>37.69</td>
<td>44.31</td>
<td>53.39</td>
<td>48.22</td>
<td>57.14</td>
<td>40.92</td>
<td></td>
</tr>
</tbody>
</table>

Table 30: reno-dctcp-sce Mean TCP Throughput (Mbit); Columns: netem bi- directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>reno</td>
<td>reno</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.32</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.34</td>
</tr>
<tr>
<td>5</td>
<td>2.39</td>
<td>1.99</td>
<td>1.75</td>
<td>1.77</td>
<td>2.39</td>
<td>2.04</td>
<td>1.75</td>
<td>1.52</td>
</tr>
<tr>
<td>10</td>
<td>4.78</td>
<td>3.80</td>
<td>3.82</td>
<td>4.23</td>
<td>4.78</td>
<td>4.00</td>
<td>3.79</td>
<td>4.06</td>
</tr>
<tr>
<td>50</td>
<td>24.24</td>
<td>21.17</td>
<td>24.08</td>
<td>24.69</td>
<td>23.60</td>
<td>26.11</td>
<td>22.91</td>
<td>20.34</td>
</tr>
<tr>
<td>100</td>
<td>42.25</td>
<td>47.10</td>
<td>37.69</td>
<td>44.31</td>
<td>53.39</td>
<td>48.22</td>
<td>57.14</td>
<td>40.92</td>
</tr>
</tbody>
</table>

Table 31: reno-sce-reno-sce Mean TCP Throughput (Mbit); Columns: netem bi- directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
### Table 32: reno-sce-dctcp Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.46</td>
<td>0.46</td>
<td>0.31</td>
<td>0.19</td>
<td>0.50</td>
<td>0.50</td>
<td>0.65</td>
<td>0.75</td>
</tr>
<tr>
<td>5</td>
<td>2.16</td>
<td>1.12</td>
<td>0.78</td>
<td>0.38</td>
<td>2.63</td>
<td>3.66</td>
<td>4.00</td>
<td>4.20</td>
</tr>
<tr>
<td>10</td>
<td>3.62</td>
<td>1.57</td>
<td>1.15</td>
<td>0.90</td>
<td>5.94</td>
<td>7.98</td>
<td>8.35</td>
<td>8.33</td>
</tr>
<tr>
<td>50</td>
<td>4.03</td>
<td>2.33</td>
<td>2.98</td>
<td>10.00</td>
<td>43.81</td>
<td>45.32</td>
<td>44.04</td>
<td>32.95</td>
</tr>
<tr>
<td>100</td>
<td>4.38</td>
<td>4.10</td>
<td>3.77</td>
<td>15.34</td>
<td>5.94</td>
<td>7.98</td>
<td>8.35</td>
<td>8.33</td>
</tr>
</tbody>
</table>

### Table 33: reno-sce-dctcp-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.34</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.33</td>
</tr>
<tr>
<td>5</td>
<td>2.39</td>
<td>1.97</td>
<td>1.65</td>
<td>1.78</td>
<td>2.39</td>
<td>2.05</td>
<td>1.73</td>
<td>0.96</td>
</tr>
<tr>
<td>10</td>
<td>4.78</td>
<td>3.97</td>
<td>3.93</td>
<td>5.01</td>
<td>4.78</td>
<td>3.73</td>
<td>3.29</td>
<td>2.60</td>
</tr>
<tr>
<td>50</td>
<td>31.92</td>
<td>39.92</td>
<td>40.62</td>
<td>39.76</td>
<td>15.90</td>
<td>7.24</td>
<td>5.80</td>
<td>4.70</td>
</tr>
<tr>
<td>100</td>
<td>62.33</td>
<td>82.41</td>
<td>85.50</td>
<td>82.05</td>
<td>33.30</td>
<td>12.61</td>
<td>9.49</td>
<td>6.82</td>
</tr>
</tbody>
</table>
### Table 34: dctcp-dctcp Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>dctcp</td>
<td>0.48</td>
<td>0.48</td>
<td>0.50</td>
<td>0.47</td>
<td>0.48</td>
<td>0.48</td>
<td>0.45</td>
<td>0.47</td>
</tr>
<tr>
<td>dctcp</td>
<td>2.41</td>
<td>2.40</td>
<td>2.39</td>
<td>2.32</td>
<td>2.37</td>
<td>2.38</td>
<td>2.37</td>
<td>2.27</td>
</tr>
<tr>
<td>10</td>
<td>4.89</td>
<td>4.88</td>
<td>4.77</td>
<td>4.66</td>
<td>4.68</td>
<td>4.66</td>
<td>4.70</td>
<td>4.38</td>
</tr>
<tr>
<td>50</td>
<td>23.87</td>
<td>23.68</td>
<td>25.27</td>
<td>19.63</td>
<td>23.95</td>
<td>23.72</td>
<td>21.30</td>
<td>21.54</td>
</tr>
<tr>
<td>100</td>
<td>49.20</td>
<td>45.78</td>
<td>47.85</td>
<td>41.79</td>
<td>46.43</td>
<td>47.44</td>
<td>45.23</td>
<td>39.41</td>
</tr>
</tbody>
</table>

### Table 35: dctcp-dctcp-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>dctcp</td>
<td>0.50</td>
<td>0.50</td>
<td>0.63</td>
<td>0.75</td>
<td>0.45</td>
<td>0.46</td>
<td>0.33</td>
<td>0.19</td>
</tr>
<tr>
<td>dctcp-sce</td>
<td>2.62</td>
<td>3.67</td>
<td>4.00</td>
<td>4.26</td>
<td>2.16</td>
<td>1.12</td>
<td>0.76</td>
<td>0.34</td>
</tr>
<tr>
<td>10</td>
<td>5.98</td>
<td>7.98</td>
<td>8.38</td>
<td>8.47</td>
<td>3.59</td>
<td>1.56</td>
<td>1.12</td>
<td>0.69</td>
</tr>
<tr>
<td>50</td>
<td>43.80</td>
<td>45.41</td>
<td>44.22</td>
<td>34.01</td>
<td>4.02</td>
<td>2.21</td>
<td>2.72</td>
<td>9.50</td>
</tr>
<tr>
<td>100</td>
<td>91.58</td>
<td>89.85</td>
<td>89.10</td>
<td>69.89</td>
<td>4.07</td>
<td>3.82</td>
<td>3.61</td>
<td>8.74</td>
</tr>
</tbody>
</table>

### Table 36: dctcp-sce-dctcp-sce Mean TCP Throughput (Mbit);
Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.34</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.33</td>
</tr>
<tr>
<td>5</td>
<td>2.39</td>
<td>1.99</td>
<td>1.70</td>
<td>1.02</td>
<td>2.39</td>
<td>2.04</td>
<td>1.66</td>
<td>1.01</td>
</tr>
<tr>
<td>10</td>
<td>4.78</td>
<td>3.78</td>
<td>3.58</td>
<td>3.04</td>
<td>4.78</td>
<td>3.71</td>
<td>3.23</td>
<td>3.33</td>
</tr>
<tr>
<td>50</td>
<td>24.16</td>
<td>19.48</td>
<td>19.33</td>
<td>18.15</td>
<td>23.61</td>
<td>21.42</td>
<td>19.76</td>
<td>20.18</td>
</tr>
<tr>
<td>100</td>
<td>48.08</td>
<td>43.17</td>
<td>54.14</td>
<td>19.58</td>
<td>47.56</td>
<td>49.83</td>
<td>36.85</td>
<td>19.25</td>
</tr>
</tbody>
</table>

### Table 37: cubic-cubic Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.42</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.51</td>
</tr>
<tr>
<td>5</td>
<td>2.42</td>
<td>2.31</td>
<td>2.44</td>
<td>2.21</td>
<td>2.36</td>
<td>2.46</td>
<td>2.29</td>
<td>2.30</td>
</tr>
<tr>
<td>10</td>
<td>4.83</td>
<td>4.95</td>
<td>4.62</td>
<td>4.15</td>
<td>4.74</td>
<td>4.59</td>
<td>4.88</td>
<td>4.63</td>
</tr>
<tr>
<td>50</td>
<td>23.88</td>
<td>23.31</td>
<td>24.19</td>
<td>23.67</td>
<td>23.95</td>
<td>24.34</td>
<td>22.99</td>
<td>19.38</td>
</tr>
<tr>
<td>100</td>
<td>50.40</td>
<td>47.62</td>
<td>48.40</td>
<td>38.13</td>
<td>45.25</td>
<td>47.30</td>
<td>46.14</td>
<td>48.92</td>
</tr>
</tbody>
</table>
### Table 38: cubic-reno Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|   | 0  | 10 | 20 | 80 | 0  | 10 | 20 | 80 |
|---|===|====|====|====|===|====|====|====|
|   | cubic | | | | reno | | | |
| 1 | 0.55 | 0.46 | 0.44 | 0.40 | 0.41 | 0.50 | 0.51 | 0.52 |
| 5 | 2.14 | 2.06 | 2.23 | 2.08 | 2.64 | 2.68 | 2.46 | 2.18 |
|10 | 4.39 | 4.63 | 4.92 | 4.28 | 5.18 | 4.78 | 4.44 | 4.06 |
|50 | 22.61| 24.49| 25.41| 29.22| 25.22| 23.18| 21.59| 12.14|
|100| 46.37| 48.43| 44.51| 49.30| 46.45| 48.88| 31.90|     |

### Table 39: cubic-reno-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|   | 0  | 10 | 20 | 80 | 0  | 10 | 20 | 80 |
|---|===|====|====|====|===|====|====|====|
|   | cubic | | | | reno-sce | | | |
| 1 | 0.45 | 0.46 | 0.51 | 0.48 | 0.50 | 0.50 | 0.44 | 0.42 |
| 5 | 2.24 | 3.07 | 3.78 | 3.24 | 2.54 | 1.68 | 0.97 | 1.13 |
|10 | 4.16 | 5.59 | 7.04 | 4.21 | 5.40 | 3.78 | 2.36 | 4.18 |
|50 | 23.95| 24.71| 25.16| 17.98| 23.87| 22.53| 21.18| 20.74|
|100| 50.40| 66.75| 65.82| 64.67| 45.26| 28.41| 28.70| 22.52|
### Table 40: cubic-dctcp Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cubic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dctcp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.40</td>
<td>0.39</td>
<td>0.37</td>
<td>0.30</td>
<td>0.56</td>
<td>0.57</td>
<td>0.59</td>
<td>0.65</td>
</tr>
<tr>
<td>5</td>
<td>1.87</td>
<td>1.38</td>
<td>1.25</td>
<td>1.57</td>
<td>2.91</td>
<td>3.40</td>
<td>3.50</td>
<td>3.03</td>
</tr>
<tr>
<td>10</td>
<td>3.37</td>
<td>1.88</td>
<td>1.56</td>
<td>2.00</td>
<td>6.20</td>
<td>7.66</td>
<td>7.89</td>
<td>7.04</td>
</tr>
<tr>
<td>50</td>
<td>6.74</td>
<td>5.79</td>
<td>6.91</td>
<td>11.76</td>
<td>2.91</td>
<td>3.40</td>
<td>3.50</td>
<td>3.03</td>
</tr>
<tr>
<td>100</td>
<td>11.73</td>
<td>12.49</td>
<td>11.51</td>
<td>22.96</td>
<td>83.95</td>
<td>71.89</td>
<td>69.95</td>
<td>60.58</td>
</tr>
</tbody>
</table>

### Table 41: cubic-dctcp-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cubic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dctcp-sce</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.44</td>
<td>0.46</td>
<td>0.51</td>
<td>0.55</td>
<td>0.52</td>
<td>0.49</td>
<td>0.44</td>
<td>0.37</td>
</tr>
<tr>
<td>5</td>
<td>2.40</td>
<td>3.18</td>
<td>3.69</td>
<td>3.46</td>
<td>2.39</td>
<td>1.58</td>
<td>1.06</td>
<td>0.89</td>
</tr>
<tr>
<td>10</td>
<td>4.02</td>
<td>6.35</td>
<td>7.58</td>
<td>5.48</td>
<td>5.54</td>
<td>3.09</td>
<td>1.87</td>
<td>2.58</td>
</tr>
<tr>
<td>50</td>
<td>24.54</td>
<td>37.56</td>
<td>35.51</td>
<td>27.95</td>
<td>23.29</td>
<td>10.16</td>
<td>11.77</td>
<td>15.42</td>
</tr>
<tr>
<td>100</td>
<td>59.52</td>
<td>77.63</td>
<td>80.96</td>
<td>75.79</td>
<td>36.12</td>
<td>17.42</td>
<td>13.35</td>
<td>11.62</td>
</tr>
</tbody>
</table>

Table 41: cubic-dctcp-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
Table 42: reno-reno Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|   | 0   | 10  | 20  | 80  | 0   | 10  | 20  | 80  |
|---+-----+-----+-----+-----+-----+-----+-----+-----|
|   | reno | reno |
| 1 | 0.49 | 0.47 | 0.49 | 0.46 | 0.47 | 0.48 | 0.47 | 0.45 |
| 5 | 2.38 | 2.39 | 2.33 | 1.96 | 2.40 | 2.35 | 2.33 | 2.16 |
| 10| 4.78 | 4.58 | 4.63 | 3.79 | 4.79 | 4.84 | 4.61 | 4.23 |
| 50| 24.05| 23.35| 24.20| 19.51| 23.78| 24.33| 22.69| 16.79|
|100|46.69 |50.26 |46.11 |35.97 |48.97 |44.58 |46.86 |38.56|

Table 43: reno-reno-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|   | 0   | 10  | 20  | 80  | 0   | 10  | 20  | 80  |
|---+-----+-----+-----+-----+-----+-----+-----+-----|
|   | reno | reno-sce |
| 1 | 0.47 | 0.50 | 0.49 | 0.49 | 0.49 | 0.46 | 0.47 | 0.39 |
| 5 | 2.51 | 2.89 | 3.58 | 3.08 | 2.27 | 1.83 | 1.12 | 1.11 |
|10 | 4.51 | 5.67 | 6.96 | 4.39 | 5.06 | 3.55 | 2.22 | 3.74 |
| 50| 23.88| 26.49| 26.33| 15.66| 23.94| 20.93| 19.68| 21.41|
|100|53.06| 62.99| 58.85| 48.15| 42.59| 31.69| 33.83| 38.41|
### Table 44: reno-dctcp Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>reno</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.42</td>
<td>0.42</td>
<td>0.38</td>
<td>0.33</td>
<td>0.54</td>
<td>0.53</td>
<td>0.58</td>
<td>0.62</td>
</tr>
<tr>
<td>5</td>
<td>2.00</td>
<td>1.50</td>
<td>1.47</td>
<td>1.49</td>
<td>2.79</td>
<td>3.28</td>
<td>3.28</td>
<td>3.06</td>
</tr>
<tr>
<td>10</td>
<td>3.62</td>
<td>2.19</td>
<td>1.92</td>
<td>2.07</td>
<td>5.95</td>
<td>7.35</td>
<td>7.53</td>
<td>6.87</td>
</tr>
<tr>
<td>50</td>
<td>7.80</td>
<td>6.96</td>
<td>6.91</td>
<td>8.15</td>
<td>40.04</td>
<td>38.98</td>
<td>37.41</td>
<td>31.25</td>
</tr>
<tr>
<td>100</td>
<td>13.40</td>
<td>13.20</td>
<td>10.37</td>
<td>2.07</td>
<td>5.95</td>
<td>7.35</td>
<td>7.53</td>
<td>6.87</td>
</tr>
</tbody>
</table>

### Table 45: reno-dctcp-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>reno</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.47</td>
<td>0.48</td>
<td>0.48</td>
<td>0.24</td>
<td>0.49</td>
<td>0.47</td>
<td>0.47</td>
<td>0.66</td>
</tr>
<tr>
<td>5</td>
<td>2.50</td>
<td>3.15</td>
<td>3.64</td>
<td>2.92</td>
<td>2.28</td>
<td>1.59</td>
<td>1.07</td>
<td>1.10</td>
</tr>
<tr>
<td>10</td>
<td>4.53</td>
<td>6.77</td>
<td>7.30</td>
<td>5.18</td>
<td>5.04</td>
<td>2.65</td>
<td>1.91</td>
<td>2.55</td>
</tr>
<tr>
<td>50</td>
<td>26.20</td>
<td>35.85</td>
<td>34.81</td>
<td>22.99</td>
<td>21.62</td>
<td>11.46</td>
<td>11.45</td>
<td>19.58</td>
</tr>
<tr>
<td>100</td>
<td>64.59</td>
<td>71.71</td>
<td>71.03</td>
<td>59.15</td>
<td>31.06</td>
<td>22.61</td>
<td>20.74</td>
<td>17.50</td>
</tr>
</tbody>
</table>
Table 46: reno-sce-reno-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.47</td>
<td>0.48</td>
<td>0.50</td>
<td>0.49</td>
<td>0.48</td>
<td>0.45</td>
<td>0.41</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2.39</td>
<td>2.19</td>
<td>2.04</td>
<td>1.81</td>
<td>2.39</td>
<td>2.15</td>
<td>1.85</td>
<td>1.97</td>
</tr>
<tr>
<td>10</td>
<td>4.77</td>
<td>4.13</td>
<td>3.90</td>
<td>3.65</td>
<td>4.79</td>
<td>4.21</td>
<td>4.05</td>
<td>4.16</td>
</tr>
<tr>
<td>50</td>
<td>23.90</td>
<td>23.79</td>
<td>21.48</td>
<td>17.40</td>
<td>23.92</td>
<td>23.94</td>
<td>25.97</td>
<td>21.52</td>
</tr>
<tr>
<td>100</td>
<td>48.57</td>
<td>54.33</td>
<td>48.58</td>
<td>47.14</td>
<td>47.10</td>
<td>46.49</td>
<td>41.08</td>
<td></td>
</tr>
</tbody>
</table>

Table 47: reno-sce-dctcp Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.41</td>
<td>0.38</td>
<td>0.28</td>
<td>0.55</td>
<td>0.55</td>
<td>0.58</td>
<td>0.66</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1.87</td>
<td>1.06</td>
<td>0.67</td>
<td>2.91</td>
<td>3.38</td>
<td>3.69</td>
<td>3.96</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>3.69</td>
<td>2.03</td>
<td>1.17</td>
<td>5.88</td>
<td>7.08</td>
<td>7.37</td>
<td>7.96</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>6.49</td>
<td>3.98</td>
<td>9.37</td>
<td>41.35</td>
<td>41.62</td>
<td>39.72</td>
<td>30.90</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>7.56</td>
<td>6.13</td>
<td>16.00</td>
<td>88.12</td>
<td>76.49</td>
<td>72.46</td>
<td>62.73</td>
<td></td>
</tr>
</tbody>
</table>
Table 48: reno-sce-dctcp-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|   | 0    | 10   | 20   | 80   | 0    | 10   | 20   | 80   |
|---+------|------|------|------|------|------|------|------|------|
| 1 | 0.47 | 0.47 | 0.49 | 0.42 | 0.48 |      |      |      |
| 5 | 2.35 | 2.10 | 1.91 | 2.27 | 2.43 | 2.20 | 1.94 | 1.38 |
| 10| 4.78 | 4.37 | 4.89 | 4.74 | 4.78 | 3.78 | 2.97 | 2.62 |
| 50| 27.23| 20.59| 25.72| 21.65|      | 8.16 | 6.56 | 13.56|
| 100| 74.00| 67.86| 67.86| 21.65|      | 12.49| 15.53| 11.94|

Table 49: dctcp-dctcp Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|   | 0    | 10   | 20   | 80   | 0    | 10   | 20   | 80   |
|---+------|------|------|------|------|------|------|------|------|
| 1 | 0.48 | 0.48 | 0.48 | 0.47 | 0.48 |      |      |      |
| 5 | 2.38 | 2.35 | 2.35 | 2.30 | 2.40 | 2.43 | 2.42 | 2.32 |
| 10| 4.74 | 4.76 | 4.73 | 4.33 | 4.82 | 4.79 | 4.72 | 4.63 |
| 50| 24.18| 22.36| 22.40| 13.88| 23.63| 24.65| 23.05| 26.04|
| 100| 42.61| 32.42| 25.55| 43.30| 53.04| 56.18| 59.67| 35.60|
### Table 50: dctcp-dctcp-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|   | 0   | 10  | 20  | 80  | 0   | 10  | 20  | 80  |
|---+-----+-----+-----+-----+-----+-----+-----+-----|
|   | dctcp |     |     |     | dctcp-sce |     |     |     |
| 1 | 0.56 | 0.56| 0.58| 0.68| 0.40 | 0.41| 0.37| 0.27|
| 5 | 2.89 | 2.38| 3.66| 3.91| 1.90 | 2.54| 1.09| 0.67|
|10 | 5.97 | 5.93| 7.55| 8.00| 3.60 | 3.58| 1.85| 1.05|
|50 | 41.35| 41.32|39.50|31.56| 6.47 | 3.79| 3.92| 9.96|
|100| 89.09| 76.31|73.53|63.92| 6.60 | 6.36| 4.97| 9.40|

### Table 51: dctcp-sce-dctcp-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|   | 0   | 10  | 20  | 80  | 0   | 10  | 20  | 80  |
|---+-----+-----+-----+-----+-----+-----+-----+-----|
|   | dctcp-sce |     |     |     | dctcp-sce |     |     |     |
| 1 | 0.50 | 0.48| 0.49| 0.40| 0.46 | 0.48| 0.47| 0.41|
| 5 | 2.40 | 2.12| 1.90| 1.65| 2.39 | 2.14| 1.92| 1.58|
|10 | 4.78 | 3.91| 3.45| 3.71| 4.79 | 3.92| 3.59| 3.33|
|50 | 23.65| 24.33|22.34|21.85|24.19 |23.17|24.15|21.08|
|100| 53.54| 59.02|54.68|46.28|42.10 |36.29|40.24|43.95|
### A.5. Two-Flow TCP Throughput (Cake "triple-isolate")

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cubic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.46</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.46</td>
</tr>
<tr>
<td>5</td>
<td>2.39</td>
<td>2.39</td>
<td>2.39</td>
<td>2.20</td>
<td>2.39</td>
<td>2.39</td>
<td>2.36</td>
<td>2.21</td>
</tr>
<tr>
<td>10</td>
<td>4.78</td>
<td>4.74</td>
<td>4.80</td>
<td>4.42</td>
<td>4.78</td>
<td>4.79</td>
<td>4.70</td>
<td>4.19</td>
</tr>
<tr>
<td>50</td>
<td>23.91</td>
<td>23.82</td>
<td>23.59</td>
<td>20.89</td>
<td>23.91</td>
<td>23.83</td>
<td>23.56</td>
<td>22.55</td>
</tr>
<tr>
<td>100</td>
<td>47.83</td>
<td>47.40</td>
<td>46.89</td>
<td>43.30</td>
<td>47.82</td>
<td>47.40</td>
<td>47.48</td>
<td>43.60</td>
</tr>
</tbody>
</table>

Table 52: cubic-cubic Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cubic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.46</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.45</td>
</tr>
<tr>
<td>5</td>
<td>2.39</td>
<td>2.39</td>
<td>2.48</td>
<td>2.11</td>
<td>2.39</td>
<td>2.39</td>
<td>2.18</td>
<td>2.16</td>
</tr>
<tr>
<td>10</td>
<td>4.78</td>
<td>4.71</td>
<td>4.81</td>
<td>4.43</td>
<td>4.78</td>
<td>4.80</td>
<td>4.60</td>
<td>4.20</td>
</tr>
<tr>
<td>50</td>
<td>23.91</td>
<td>24.24</td>
<td>24.11</td>
<td>23.11</td>
<td>23.91</td>
<td>23.42</td>
<td>22.23</td>
<td>17.99</td>
</tr>
<tr>
<td>100</td>
<td>47.83</td>
<td>48.65</td>
<td>48.14</td>
<td>48.87</td>
<td>47.83</td>
<td>46.14</td>
<td>43.84</td>
<td>35.92</td>
</tr>
</tbody>
</table>

Table 53: cubic-reno Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
### Table 54: cubic-reno-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>cubic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.57</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.32</td>
</tr>
<tr>
<td>5</td>
<td>2.39</td>
<td>2.70</td>
<td>2.87</td>
<td>3.40</td>
<td>2.39</td>
<td>2.00</td>
<td>1.75</td>
<td>1.00</td>
</tr>
<tr>
<td>10</td>
<td>4.78</td>
<td>5.60</td>
<td>5.38</td>
<td>4.56</td>
<td>4.78</td>
<td>3.75</td>
<td>3.79</td>
<td>4.09</td>
</tr>
<tr>
<td>50</td>
<td>23.92</td>
<td>24.11</td>
<td>23.51</td>
<td>20.30</td>
<td>23.90</td>
<td>23.56</td>
<td>23.72</td>
<td>24.27</td>
</tr>
<tr>
<td>100</td>
<td>47.83</td>
<td>47.45</td>
<td>46.41</td>
<td>42.39</td>
<td>47.81</td>
<td>47.71</td>
<td>48.22</td>
<td>48.49</td>
</tr>
</tbody>
</table>

### Table 55: cubic-dctcp Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>cubic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.44</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.50</td>
</tr>
<tr>
<td>5</td>
<td>2.39</td>
<td>2.38</td>
<td>2.34</td>
<td>2.16</td>
<td>2.39</td>
<td>2.40</td>
<td>2.41</td>
<td>2.35</td>
</tr>
<tr>
<td>10</td>
<td>4.78</td>
<td>4.68</td>
<td>4.77</td>
<td>4.38</td>
<td>4.78</td>
<td>4.85</td>
<td>4.71</td>
<td>4.44</td>
</tr>
<tr>
<td>50</td>
<td>23.91</td>
<td>23.86</td>
<td>23.36</td>
<td>21.81</td>
<td>23.91</td>
<td>23.68</td>
<td>23.58</td>
<td>21.10</td>
</tr>
<tr>
<td>100</td>
<td>47.84</td>
<td>47.77</td>
<td>46.77</td>
<td>47.27</td>
<td>47.82</td>
<td>46.94</td>
<td>47.30</td>
<td>40.48</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>c</td>
<td>cu</td>
<td>cu</td>
<td>cu</td>
<td>cu</td>
<td>cu</td>
<td>cu</td>
<td>cu</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>1</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.58</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.31</td>
</tr>
<tr>
<td>5</td>
<td>2.39</td>
<td>2.69</td>
<td>2.94</td>
<td>3.58</td>
<td>2.39</td>
<td>2.02</td>
<td>1.68</td>
<td>0.74</td>
</tr>
<tr>
<td>10</td>
<td>4.78</td>
<td>5.81</td>
<td>6.15</td>
<td>6.93</td>
<td>4.78</td>
<td>3.57</td>
<td>3.03</td>
<td>1.62</td>
</tr>
<tr>
<td>50</td>
<td>24.02</td>
<td>27.16</td>
<td>26.07</td>
<td>20.95</td>
<td>23.80</td>
<td>20.14</td>
<td>19.16</td>
<td>21.20</td>
</tr>
<tr>
<td>100</td>
<td>47.85</td>
<td>49.02</td>
<td>47.67</td>
<td>42.50</td>
<td>47.80</td>
<td>46.02</td>
<td>45.94</td>
<td>42.97</td>
</tr>
</tbody>
</table>

Table 56: cubic-dctcp-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>r</td>
<td>re</td>
<td>re</td>
<td>re</td>
<td>re</td>
<td>re</td>
<td>re</td>
<td>re</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>1</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.46</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.46</td>
</tr>
<tr>
<td>5</td>
<td>2.39</td>
<td>2.37</td>
<td>2.31</td>
<td>2.04</td>
<td>2.39</td>
<td>2.35</td>
<td>2.33</td>
<td>2.01</td>
</tr>
<tr>
<td>10</td>
<td>4.78</td>
<td>4.81</td>
<td>4.60</td>
<td>4.25</td>
<td>4.78</td>
<td>4.74</td>
<td>4.60</td>
<td>4.33</td>
</tr>
<tr>
<td>50</td>
<td>23.91</td>
<td>23.77</td>
<td>23.07</td>
<td>17.70</td>
<td>23.91</td>
<td>23.77</td>
<td>22.92</td>
<td>17.72</td>
</tr>
<tr>
<td>100</td>
<td>47.83</td>
<td>47.39</td>
<td>45.06</td>
<td>37.23</td>
<td>47.83</td>
<td>47.54</td>
<td>44.58</td>
<td>35.40</td>
</tr>
</tbody>
</table>

Table 57: reno-reno Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
Table 58: reno-reno-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>reno</td>
<td>reno-sce</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.55</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.32</td>
</tr>
<tr>
<td>5</td>
<td>2.39</td>
<td>2.61</td>
<td>2.76</td>
<td>2.91</td>
<td>2.39</td>
<td>2.01</td>
<td>1.76</td>
<td>1.16</td>
</tr>
<tr>
<td>10</td>
<td>4.78</td>
<td>5.39</td>
<td>5.44</td>
<td>4.34</td>
<td>4.78</td>
<td>3.84</td>
<td>3.76</td>
<td>4.24</td>
</tr>
<tr>
<td>50</td>
<td>23.93</td>
<td>23.81</td>
<td>22.02</td>
<td>17.75</td>
<td>23.90</td>
<td>23.72</td>
<td>24.79</td>
<td>25.80</td>
</tr>
<tr>
<td>100</td>
<td>47.83</td>
<td>46.09</td>
<td>42.73</td>
<td>35.26</td>
<td>47.82</td>
<td>49.03</td>
<td>50.97</td>
<td>50.07</td>
</tr>
</tbody>
</table>

Table 59: reno-dctcp Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>reno</td>
<td>dctcp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.43</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.50</td>
</tr>
<tr>
<td>5</td>
<td>2.39</td>
<td>2.30</td>
<td>2.27</td>
<td>1.88</td>
<td>2.39</td>
<td>2.47</td>
<td>2.47</td>
<td>2.44</td>
</tr>
<tr>
<td>10</td>
<td>4.78</td>
<td>4.73</td>
<td>4.66</td>
<td>4.19</td>
<td>4.78</td>
<td>4.79</td>
<td>4.78</td>
<td>4.30</td>
</tr>
<tr>
<td>50</td>
<td>23.91</td>
<td>23.45</td>
<td>21.71</td>
<td>17.98</td>
<td>23.91</td>
<td>24.13</td>
<td>24.75</td>
<td>21.47</td>
</tr>
<tr>
<td>100</td>
<td>47.82</td>
<td>46.42</td>
<td>43.70</td>
<td>35.62</td>
<td>47.81</td>
<td>48.18</td>
<td>49.48</td>
<td>42.19</td>
</tr>
</tbody>
</table>

### Table 60: reno-dctcp-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.54</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.33</td>
</tr>
<tr>
<td>5</td>
<td>2.39</td>
<td>2.60</td>
<td>2.81</td>
<td>3.25</td>
<td>2.39</td>
<td>2.02</td>
<td>1.69</td>
<td>0.75</td>
</tr>
<tr>
<td>10</td>
<td>4.78</td>
<td>5.65</td>
<td>6.18</td>
<td>6.09</td>
<td>4.78</td>
<td>3.60</td>
<td>2.97</td>
<td>1.90</td>
</tr>
<tr>
<td>50</td>
<td>24.05</td>
<td>27.29</td>
<td>24.37</td>
<td>18.32</td>
<td>23.77</td>
<td>19.78</td>
<td>19.76</td>
<td>22.57</td>
</tr>
<tr>
<td>100</td>
<td>47.87</td>
<td>47.75</td>
<td>44.23</td>
<td>35.55</td>
<td>47.79</td>
<td>46.27</td>
<td>47.20</td>
<td>42.70</td>
</tr>
</tbody>
</table>

### Table 61: reno-sce-reno-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.34</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.33</td>
</tr>
<tr>
<td>5</td>
<td>2.39</td>
<td>2.05</td>
<td>1.80</td>
<td>1.75</td>
<td>2.39</td>
<td>2.03</td>
<td>1.89</td>
<td>1.62</td>
</tr>
<tr>
<td>10</td>
<td>4.78</td>
<td>4.15</td>
<td>4.14</td>
<td>4.22</td>
<td>4.78</td>
<td>4.15</td>
<td>4.12</td>
<td>4.31</td>
</tr>
<tr>
<td>50</td>
<td>23.91</td>
<td>23.66</td>
<td>23.54</td>
<td>22.01</td>
<td>23.79</td>
<td>23.69</td>
<td>23.24</td>
<td>22.75</td>
</tr>
<tr>
<td>100</td>
<td>47.82</td>
<td>47.66</td>
<td>47.52</td>
<td>45.05</td>
<td>47.82</td>
<td>47.66</td>
<td>47.54</td>
<td>44.97</td>
</tr>
</tbody>
</table>

### Table 62: reno-sce-dctcp-sce Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.54</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.33</td>
</tr>
<tr>
<td>5</td>
<td>2.39</td>
<td>2.60</td>
<td>2.81</td>
<td>3.25</td>
<td>2.39</td>
<td>2.02</td>
<td>1.69</td>
<td>0.75</td>
</tr>
<tr>
<td>10</td>
<td>4.78</td>
<td>5.65</td>
<td>6.18</td>
<td>6.09</td>
<td>4.78</td>
<td>3.60</td>
<td>2.97</td>
<td>1.90</td>
</tr>
<tr>
<td>50</td>
<td>24.05</td>
<td>27.29</td>
<td>24.37</td>
<td>18.32</td>
<td>23.77</td>
<td>19.78</td>
<td>19.76</td>
<td>22.57</td>
</tr>
<tr>
<td>100</td>
<td>47.87</td>
<td>47.75</td>
<td>44.23</td>
<td>35.55</td>
<td>47.79</td>
<td>46.27</td>
<td>47.20</td>
<td>42.70</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>80</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>80</td>
</tr>
<tr>
<td>---+===+----------+-----+-----+---+----+----+----</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.32</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.61</td>
</tr>
<tr>
<td>5</td>
<td>2.39</td>
<td>1.98</td>
<td>1.71</td>
<td>0.91</td>
<td>2.39</td>
<td>2.78</td>
<td>2.95</td>
<td>3.60</td>
</tr>
<tr>
<td>10</td>
<td>4.78</td>
<td>3.75</td>
<td>3.85</td>
<td>3.99</td>
<td>4.78</td>
<td>5.64</td>
<td>5.41</td>
<td>4.90</td>
</tr>
<tr>
<td>50</td>
<td>23.91</td>
<td>23.71</td>
<td>23.51</td>
<td>22.56</td>
<td>23.91</td>
<td>23.99</td>
<td>23.57</td>
<td>20.79</td>
</tr>
<tr>
<td>100</td>
<td>47.83</td>
<td>48.07</td>
<td>47.63</td>
<td>46.93</td>
<td>47.83</td>
<td>47.03</td>
<td>46.40</td>
<td>39.48</td>
</tr>
</tbody>
</table>

Table 62: reno-sce-dctcp Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
Table 64: dctcp-dctcp Mean TCP Throughput (Mbit); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
Table 66: `dctcp-sce-dctcp-sce` Mean TCP Throughput (Mbit); Columns: `netem` bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>dctcp-sce</code></td>
<td></td>
<td></td>
<td><code>dctcp-sce</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.34</td>
<td>0.48</td>
<td>0.48</td>
<td>0.48</td>
<td>0.34</td>
</tr>
<tr>
<td>5</td>
<td>2.39</td>
<td>2.04</td>
<td>1.71</td>
<td>1.09</td>
<td>2.39</td>
<td>2.03</td>
<td>1.76</td>
<td>1.06</td>
</tr>
<tr>
<td>10</td>
<td>4.78</td>
<td>3.91</td>
<td>3.55</td>
<td>2.97</td>
<td>4.78</td>
<td>4.01</td>
<td>3.59</td>
<td>2.84</td>
</tr>
<tr>
<td>100</td>
<td>47.82</td>
<td>45.19</td>
<td>43.83</td>
<td>36.16</td>
<td>47.82</td>
<td>46.37</td>
<td>44.59</td>
<td>37.07</td>
</tr>
</tbody>
</table>

Table 67: `cubic-cubic` Mean TCP RTT (ms); Columns: `netem` bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>cubic</code></td>
<td></td>
<td></td>
<td><code>cubic</code></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>70.73</td>
<td>69.97</td>
<td>83.70</td>
<td>139.06</td>
<td>69.07</td>
<td>70.38</td>
<td>86.15</td>
<td>140.64</td>
</tr>
<tr>
<td>5</td>
<td>14.09</td>
<td>25.54</td>
<td>35.72</td>
<td>91.62</td>
<td>14.26</td>
<td>25.99</td>
<td>35.59</td>
<td>91.28</td>
</tr>
<tr>
<td>10</td>
<td>7.88</td>
<td>18.69</td>
<td>28.67</td>
<td>86.54</td>
<td>7.79</td>
<td>18.46</td>
<td>28.60</td>
<td>85.74</td>
</tr>
<tr>
<td>50</td>
<td>6.53</td>
<td>16.30</td>
<td>25.55</td>
<td>82.98</td>
<td>6.47</td>
<td>16.37</td>
<td>25.66</td>
<td>83.06</td>
</tr>
<tr>
<td>100</td>
<td>6.35</td>
<td>16.40</td>
<td>25.49</td>
<td>83.27</td>
<td>6.43</td>
<td>16.29</td>
<td>25.57</td>
<td>83.36</td>
</tr>
</tbody>
</table>

A.6. Two-Flow TCP RTT (Cake "flowblind")
Table 68: cubic-reno Mean TCP RTT (ms); Columns: netem bidirectional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>65.97</td>
<td>71.14</td>
<td>83.63</td>
<td>139.14</td>
<td>66.25</td>
<td>72.11</td>
<td>87.70</td>
<td>140.12</td>
</tr>
<tr>
<td>5</td>
<td>14.00</td>
<td>25.94</td>
<td>35.16</td>
<td>91.73</td>
<td>14.87</td>
<td>25.03</td>
<td>35.98</td>
<td>92.22</td>
</tr>
<tr>
<td>10</td>
<td>7.60</td>
<td>18.82</td>
<td>28.77</td>
<td>86.37</td>
<td>7.99</td>
<td>19.39</td>
<td>29.11</td>
<td>86.39</td>
</tr>
<tr>
<td>50</td>
<td>6.43</td>
<td>16.26</td>
<td>25.45</td>
<td>82.72</td>
<td>6.41</td>
<td>16.38</td>
<td>25.46</td>
<td>83.06</td>
</tr>
<tr>
<td>100</td>
<td>6.31</td>
<td>16.20</td>
<td>25.09</td>
<td>83.07</td>
<td>6.47</td>
<td>16.09</td>
<td>25.11</td>
<td>83.19</td>
</tr>
</tbody>
</table>

Table 69: cubic-reno-sce Mean TCP RTT (ms); Columns: netem bidirectional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>71.98</td>
<td>73.29</td>
<td>93.38</td>
<td>127.68</td>
<td>61.37</td>
<td>61.69</td>
<td>71.07</td>
<td>121.98</td>
</tr>
<tr>
<td>5</td>
<td>14.58</td>
<td>25.06</td>
<td>33.50</td>
<td>88.85</td>
<td>11.80</td>
<td>21.16</td>
<td>32.19</td>
<td>94.96</td>
</tr>
<tr>
<td>10</td>
<td>7.70</td>
<td>17.86</td>
<td>27.34</td>
<td>85.73</td>
<td>6.98</td>
<td>18.59</td>
<td>30.43</td>
<td>90.51</td>
</tr>
<tr>
<td>50</td>
<td>5.95</td>
<td>15.00</td>
<td>24.47</td>
<td>83.61</td>
<td>6.01</td>
<td>16.71</td>
<td>26.93</td>
<td>86.54</td>
</tr>
<tr>
<td>100</td>
<td>6.01</td>
<td>14.78</td>
<td>24.44</td>
<td>83.56</td>
<td>6.06</td>
<td>15.99</td>
<td>26.21</td>
<td>85.14</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>80</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>80</td>
</tr>
<tr>
<td>---+-----+-----+-----+-----+-----+-----+-----+-----</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cubic</td>
<td></td>
<td></td>
<td></td>
<td>dctcp</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>61.63</td>
<td>62.74</td>
<td>79.63</td>
<td>144.41</td>
<td>81.28</td>
<td>81.99</td>
<td>88.30</td>
<td>142.69</td>
</tr>
<tr>
<td>5</td>
<td>12.75</td>
<td>27.02</td>
<td>38.94</td>
<td>94.68</td>
<td>18.98</td>
<td>24.23</td>
<td>33.58</td>
<td>92.59</td>
</tr>
<tr>
<td>10</td>
<td>7.19</td>
<td>20.98</td>
<td>30.61</td>
<td>87.70</td>
<td>8.12</td>
<td>17.94</td>
<td>27.88</td>
<td>86.25</td>
</tr>
<tr>
<td>50</td>
<td>6.84</td>
<td>16.43</td>
<td>26.24</td>
<td>83.90</td>
<td>6.38</td>
<td>16.05</td>
<td>26.01</td>
<td>84.39</td>
</tr>
<tr>
<td>100</td>
<td>6.49</td>
<td>16.34</td>
<td>26.42</td>
<td>85.08</td>
<td>6.46</td>
<td>16.42</td>
<td>26.19</td>
<td>84.83</td>
</tr>
</tbody>
</table>

Table 70: cubic-dctcp Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|   | 0    | 10   | 20   | 80   | 0    | 10   | 20   | 80   |
|---+-----+-----+-----+-----+-----+-----+-----+-----|
|   | cubic|     |     |     | dctcp-sce|     |     |     |     |
| 1 | 73.64| 72.10| 92.35| 127.57| 60.95 | 63.18| 73.14| 117.81|
| 5 | 14.10| 23.88| 32.68| 89.27 | 11.73| 21.25| 32.24| 95.10 |
| 10| 7.99 | 17.94| 27.11| 85.58 | 6.94 | 18.23| 30.11| 91.01 |
| 50| 5.91 | 14.97| 24.16| 83.11 | 5.97 | 16.67| 26.92| 86.08 |
| 100|6.13 | 14.98| 24.27| 82.93| 6.08 | 16.17| 26.17| 84.85 |

Table 71: cubic-dctcp-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
|   |0    | 10   | 20   | 80   | 0    | 10   | 20   | 80   |
|---+-----+-----+-----+-----+-----+-----+-----+-----|
|reno|      |      |      |      | reno|      |      |      |
|1  |64.31 | 68.79| 84.19| 141.79| 66.79| 71.57| 83.55| 140.03|
|5  |14.46 | 25.39| 35.62| 92.66 | 14.61| 25.49| 35.57| 92.29 |
|10 |8.15  | 18.91| 29.00| 86.57 | 8.08 | 18.83| 29.11| 86.26 |
|50 |6.49  | 14.43| 24.97| 82.44 | 6.44 | 16.23| 25.06| 82.48 |
|100|6.43  | 16.20| 24.68| 82.39 | 6.50 | 16.18| 24.91| 82.39 |

Table 72: reno-reno Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|   |0    | 10   | 20   | 80   | 0    | 10   | 20   | 80   |
|---+-----+-----+-----+-----+-----+-----+-----+-----|
|reno|      |      |      |      | reno-sce|      |      |      |
|1  |68.15 | 70.79| 91.27| 130.38| 62.79| 62.52| 75.28| 122.99|
|5  |14.95 | 23.90| 34.12| 89.47 | 12.89| 21.95| 33.51| 95.43 |
|10 |8.05  | 18.27| 28.12| 85.51 | 7.22 | 18.71| 29.40| 90.04 |
|50 |5.99  | 14.84| 24.17| 83.36 | 6.09 | 16.08| 25.91| 84.37 |
|100|6.19  | 14.61| 24.02| 83.03 | 6.05 | 15.50| 25.06| 83.81 |

Table 73: reno-reno-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
Table 74: reno-dctcp Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|   | 0    | 10   | 20   | 80 | 0    | 10   | 20   | 80 |
|---+-----+-----+-----+-----+-----+-----+-----+-----|
|   | reno |     |     |     | dctcp |     |     |     |
| 1 | 59.96| 62.40| 83.02| 144.86| 114.59| 108.05| 87.97| 143.70|
| 5 | 12.78| 27.64| 38.54| 95.38 | 22.21 | 24.00 | 33.12| 92.06 |
|10 | 8.10 | 20.18| 30.66| 88.42 | 8.31  | 18.10 | 27.56| 86.64 |
|50 | 6.87 | 16.58| 26.04| 83.56 | 6.48  | 16.36 | 25.81| 83.38 |
|100| 6.38 | 16.54| 26.00| 84.56 | 6.48  | 16.43 | 25.93| 83.75 |

Table 75: reno-dctcp-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|   | 0    | 10   | 20   | 80 | 0    | 10   | 20   | 80 |
|---+-----+-----+-----+-----+-----+-----+-----+-----|
|   | reno |     |     |     | dctcp-sce |     |     |     |
| 1 | 67.25| 70.19| 93.80| 131.16| 63.59  | 63.29| 75.04| 120.28|
| 5 | 15.01| 26.28| 34.17| 89.37 | 12.68  | 23.72| 33.43| 94.57 |
|10 | 7.89 | 18.30| 27.91| 85.30 | 7.11   | 18.84| 29.84| 90.39 |
|50 | 6.09 | 14.74| 23.74| 82.58 | 6.07   | 16.17| 25.95| 84.17 |
|100| 5.78 | 14.32| 23.62| 82.36 | 5.95   | 15.20| 24.81| 83.98 |

|     | 0 | 10 | 20 | 80 | 0  | 10 | 20 | 80 |
|-----+---+----+----+----+---+----+----+----|
|     | reno-sce |     |     |     | reno-sce |     |     |     |
| 1   | 49.45     | 50.66 | 50.71 | 114.04 | 50.74 | 50.87 | 50.54 | 113.68 |
| 5   | 9.58      | 19.47 | 30.20 | 88.97  | 9.66  | 19.40 | 30.39 | 89.81  |
| 10  | 6.55      | 16.99 | 27.00 | 85.35  | 6.55  | 16.61 | 26.91 | 85.29  |
| 50  | 4.46      | 13.52 | 23.27 | 82.93  | 4.54  | 13.35 | 23.26 | 83.11  |
| 100 | 4.40      | 14.06 | 23.83 | 82.74  | 4.21  | 14.07 | 23.74 | 82.71  |

Table 76: reno-sce-reno-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|     | 0 | 10 | 20 | 80 | 0  | 10 | 20 | 80 |
|-----+---+----+----+----+---+----+----+----|
|     | reno-sce |     |     |     | dctcp |     |     |     |
| 1   | 53.35     | 54.29 | 82.98 | 132.00 | 93.33 | 94.62 | 84.60 | 141.33 |
| 5   | 11.15     | 21.45 | 31.72 | 93.81  | 18.29 | 23.96 | 33.21 | 90.14  |
| 10  | 7.04      | 18.09 | 30.04 | 92.23  | 8.00  | 17.48 | 27.41 | 86.25  |
| 50  | 6.15      | 16.43 | 27.42 | 85.57  | 6.32  | 15.91 | 25.72 | 83.84  |
| 100 | 6.29      | 16.54 | 26.52 | 85.06  | 6.41  | 16.26 | 26.07 | 83.57  |

Table 77: reno-sce-dctcp Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
### Table 78: reno-sce-dctcp-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>49.87</td>
<td>51.12</td>
<td>50.45</td>
<td>113.39</td>
<td>50.23</td>
<td>50.54</td>
<td>50.13</td>
<td>112.50</td>
</tr>
<tr>
<td>5</td>
<td>9.57</td>
<td>19.48</td>
<td>30.47</td>
<td>88.94</td>
<td>9.64</td>
<td>19.11</td>
<td>29.76</td>
<td>91.18</td>
</tr>
<tr>
<td>10</td>
<td>6.58</td>
<td>16.86</td>
<td>27.26</td>
<td>84.81</td>
<td>6.63</td>
<td>16.66</td>
<td>27.39</td>
<td>85.95</td>
</tr>
<tr>
<td>50</td>
<td>4.20</td>
<td>12.98</td>
<td>22.81</td>
<td>82.45</td>
<td>4.59</td>
<td>14.89</td>
<td>24.87</td>
<td>84.83</td>
</tr>
<tr>
<td>100</td>
<td>4.15</td>
<td>13.47</td>
<td>23.29</td>
<td>82.64</td>
<td>4.11</td>
<td>14.16</td>
<td>24.11</td>
<td>83.99</td>
</tr>
</tbody>
</table>

### Table 79: dctcp-dctcp Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>94.39</td>
<td>94.71</td>
<td>87.26</td>
<td>145.73</td>
<td>95.80</td>
<td>95.81</td>
<td>92.26</td>
<td>144.97</td>
</tr>
<tr>
<td>5</td>
<td>18.87</td>
<td>25.91</td>
<td>35.08</td>
<td>93.41</td>
<td>19.48</td>
<td>26.10</td>
<td>34.76</td>
<td>93.83</td>
</tr>
<tr>
<td>10</td>
<td>9.29</td>
<td>18.56</td>
<td>27.94</td>
<td>87.31</td>
<td>9.40</td>
<td>18.65</td>
<td>28.15</td>
<td>87.12</td>
</tr>
<tr>
<td>50</td>
<td>6.78</td>
<td>16.54</td>
<td>26.18</td>
<td>84.83</td>
<td>6.74</td>
<td>16.48</td>
<td>26.21</td>
<td>84.80</td>
</tr>
<tr>
<td>100</td>
<td>6.69</td>
<td>16.43</td>
<td>25.89</td>
<td>85.15</td>
<td>6.75</td>
<td>16.30</td>
<td>25.94</td>
<td>85.35</td>
</tr>
</tbody>
</table>
### Table 80: dctcp-dctcp-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|   | 0    | 10   | 20   | 80    | 0    | 10   | 20   | 80    |
|---+------|------|------|-------|------|------|------|-------|
| 1 | 91.81| 93.17| 83.18| 141.45| 53.62| 53.10| 76.65| 133.28|
| 5 | 18.22| 23.79| 33.56| 91.14 | 11.04| 21.38| 31.87| 93.16 |
| 10| 8.06 | 17.46| 27.21| 86.24 | 7.25 | 18.04| 29.56| 92.76 |
| 50| 6.14 | 15.92| 25.60| 83.50 | 6.17 | 16.47| 27.02| 86.00 |
| 100|6.35 | 16.04| 25.95| 83.06 | 6.43 | 16.35| 26.71| 85.56 |

### Table 81: dctcp-sce-dctcp-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|   | 0    | 10   | 20   | 80    | 0    | 10   | 20   | 80    |
|---+------|------|------|-------|------|------|------|-------|
| 1 | 48.67| 49.51| 50.22| 115.03| 49.42| 49.95| 49.96| 113.68|
| 5 | 9.56 | 19.62| 29.79| 91.63 | 9.63 | 19.19| 30.02| 91.44 |
| 10| 6.52 | 16.78| 27.09| 85.64 | 6.61 | 16.65| 27.42| 85.73 |
| 50| 4.16 | 13.13| 22.39| 82.21 | 4.18 | 12.96| 22.47| 82.09 |
| 100|3.90 | 13.02| 22.61| 82.16 | 3.97 | 12.94| 22.72| 82.04 |

### A.7. Two-Flow TCP RTT (Cake "flowblind sce-single")

|   | 0    | 10   | 20   | 80    | 0    | 10   | 20   | 80    |
|---+------|------|------|-------|------|------|------|-------|
| 1 | 68.26| 69.38| 83.80| 142.98| 69.18| 68.64| 83.08| 142.21|
Table 82: cubic-cubic Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|   | 0    | 10    | 20    | 80    | 0    | 10    | 20    | 80    |
|---+-----+-------+-------+--------+-----+-------+-------+--------+       |
| 5 | 13.86| 25.36  | 35.54  | 92.39  | 13.92| 24.90  | 36.61  | 92.30  |
| 10| 8.39 | 18.95  | 28.76  | 86.67  | 8.19 | 19.22  | 28.77  | 86.23  |
| 50| 6.52 | 16.30  | 25.85  | 83.11  | 6.59 | 16.26  | 25.84  | 83.18  |
| 100| 6.42 | 16.49  | 25.57  | 83.19  | 6.40 | 16.38  | 25.65  | 83.22  |

Table 83: cubic-reno Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|   | 0    | 10    | 20    | 80    | 0    | 10    | 20    | 80    |
|---+-----+-------+-------+--------+-----+-------+-------+--------+       |
| 5 | 13.62| 25.89  | 35.86  | 92.45  | 14.26| 25.25  | 36.40  | 93.45  |
| 10| 7.82 | 18.99  | 28.74  | 86.24  | 8.23 | 19.64  | 29.36  | 86.42  |
| 50| 6.57 | 16.15  | 25.30  | 82.60  | 6.53 | 16.17  | 25.47  | 83.21  |
| 100| 6.46 | 16.14  | 24.89  | 83.04  | 6.54 | 16.23  | 24.88  | 83.28  |

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>67.62</td>
<td>68.51</td>
<td>83.34</td>
<td>138.15</td>
<td>68.37</td>
<td>69.98</td>
<td>84.17</td>
<td>139.38</td>
</tr>
<tr>
<td>5</td>
<td>13.56</td>
<td>24.05</td>
<td>34.64</td>
<td>91.07</td>
<td>13.81</td>
<td>24.46</td>
<td>37.77</td>
<td>96.38</td>
</tr>
<tr>
<td>10</td>
<td>8.16</td>
<td>18.42</td>
<td>28.22</td>
<td>86.47</td>
<td>8.07</td>
<td>19.70</td>
<td>31.56</td>
<td>86.75</td>
</tr>
<tr>
<td>50</td>
<td>6.44</td>
<td>15.91</td>
<td>25.22</td>
<td>82.90</td>
<td>6.81</td>
<td>16.17</td>
<td>25.55</td>
<td>82.74</td>
</tr>
<tr>
<td>100</td>
<td>6.47</td>
<td>16.73</td>
<td>26.43</td>
<td>84.25</td>
<td>6.62</td>
<td>16.63</td>
<td>26.62</td>
<td>84.78</td>
</tr>
</tbody>
</table>

Table 84: cubic-reno-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>64.14</td>
<td>64.56</td>
<td>77.80</td>
<td>145.02</td>
<td>65.75</td>
<td>64.93</td>
<td>80.94</td>
<td>142.91</td>
</tr>
<tr>
<td>5</td>
<td>13.23</td>
<td>25.90</td>
<td>38.90</td>
<td>96.18</td>
<td>13.84</td>
<td>23.58</td>
<td>33.08</td>
<td>93.08</td>
</tr>
<tr>
<td>10</td>
<td>7.73</td>
<td>21.30</td>
<td>34.10</td>
<td>90.52</td>
<td>7.86</td>
<td>18.08</td>
<td>27.45</td>
<td>87.45</td>
</tr>
<tr>
<td>50</td>
<td>7.93</td>
<td>18.01</td>
<td>28.17</td>
<td>86.63</td>
<td>7.02</td>
<td>16.02</td>
<td>26.69</td>
<td>85.82</td>
</tr>
<tr>
<td>100</td>
<td>7.66</td>
<td>17.36</td>
<td>27.50</td>
<td>86.24</td>
<td>6.87</td>
<td>16.47</td>
<td>26.59</td>
<td>85.79</td>
</tr>
</tbody>
</table>

Table 85: cubic-dctcp Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
|   | 0    | 10   | 20   | 80   | 0    | 10   | 20   | 80   |
|---+-----+-----+-----+-----+-----+-----+-----+-----|
| cubic |       |       |      |      |      |      |      |      |
| 1  | 67.09| 68.20| 83.79| 137.00| 68.78| 84.89| 142.55|
| 5  | 14.22| 24.20| 34.59| 90.97 | 13.60| 24.91| 36.51| 96.69|
| 10 | 8.16 | 18.76| 28.16| 86.07 | 8.16 | 20.29| 31.97| 88.89|
| 50 | 6.42 | 16.27| 25.71| 83.55 | 6.69 | 17.27| 26.47| 84.08|
| 100| 6.54 | 16.35| 25.54| 83.88 | 6.79 | 16.65| 26.21| 84.75|

Table 86: cubic-dctcp-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|   | 0    | 10   | 20   | 80   | 0    | 10   | 20   | 80   |
|---+-----+-----+-----+-----+-----+-----+-----+-----|
| reno |       |       |      |      |      |      |      |      |
| 1  | 67.27| 69.27| 83.70| 145.08| 67.81| 68.53| 84.48| 145.65|
| 5  | 14.07| 26.21| 36.48| 93.63 | 14.28| 26.00| 36.21| 93.05|
| 10 | 8.21 | 19.06| 29.68| 86.73 | 8.20 | 18.96| 29.25| 86.39|
| 50 | 6.62 | 16.41| 24.97| 82.36 | 6.68 | 16.32| 25.01| 82.48|
| 100| 6.57 | 15.91| 24.64| 82.45 | 6.45 | 15.90| 24.65| 82.56|

Table 87: reno-reno Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
### Table 88: reno-reno-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>reno</td>
<td>67.65</td>
<td>68.24</td>
<td>83.30</td>
<td>139.85</td>
<td>68.25</td>
<td>70.37</td>
<td>79.56</td>
<td>142.52</td>
</tr>
<tr>
<td>reno-sce</td>
<td>14.09</td>
<td>24.97</td>
<td>35.97</td>
<td>91.33</td>
<td>14.55</td>
<td>24.35</td>
<td>37.58</td>
<td>96.81</td>
</tr>
<tr>
<td>1</td>
<td>8.54</td>
<td>19.17</td>
<td>28.84</td>
<td>86.91</td>
<td>8.21</td>
<td>19.96</td>
<td>32.27</td>
<td>87.41</td>
</tr>
<tr>
<td>5</td>
<td>6.44</td>
<td>15.55</td>
<td>24.88</td>
<td>82.79</td>
<td>6.56</td>
<td>16.08</td>
<td>25.22</td>
<td>82.59</td>
</tr>
</tbody>
</table>

### Table 89: reno-dctcp Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>reno</td>
<td>62.85</td>
<td>65.76</td>
<td>79.20</td>
<td>146.75</td>
<td>65.00</td>
<td>66.92</td>
<td>78.64</td>
<td>140.83</td>
</tr>
<tr>
<td>dctcp</td>
<td>13.65</td>
<td>26.18</td>
<td>38.66</td>
<td>97.29</td>
<td>14.02</td>
<td>23.92</td>
<td>33.54</td>
<td>93.24</td>
</tr>
<tr>
<td>1</td>
<td>8.12</td>
<td>21.46</td>
<td>32.84</td>
<td>91.43</td>
<td>7.89</td>
<td>17.89</td>
<td>27.67</td>
<td>87.40</td>
</tr>
<tr>
<td>5</td>
<td>7.90</td>
<td>18.20</td>
<td>28.50</td>
<td>85.93</td>
<td>6.86</td>
<td>16.53</td>
<td>26.78</td>
<td>84.89</td>
</tr>
<tr>
<td>10</td>
<td>7.58</td>
<td>17.41</td>
<td>27.77</td>
<td>85.89</td>
<td>7.06</td>
<td>16.76</td>
<td>27.43</td>
<td>85.22</td>
</tr>
</tbody>
</table>
Table 90: reno-dctcp-sce Mean TCP RTT (ms); Columns: netem bidirectional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>66.94</td>
<td>68.38</td>
<td>84.13</td>
<td>443.94</td>
<td>68.15</td>
<td>70.50</td>
<td>80.58</td>
<td>1036.03</td>
</tr>
<tr>
<td>5</td>
<td>14.59</td>
<td>25.96</td>
<td>35.41</td>
<td>91.34</td>
<td>14.17</td>
<td>25.50</td>
<td>36.79</td>
<td>96.14</td>
</tr>
<tr>
<td>10</td>
<td>8.76</td>
<td>19.46</td>
<td>28.55</td>
<td>86.17</td>
<td>8.35</td>
<td>21.14</td>
<td>31.51</td>
<td>88.70</td>
</tr>
<tr>
<td>50</td>
<td>6.32</td>
<td>15.99</td>
<td>25.40</td>
<td>83.48</td>
<td>6.63</td>
<td>17.15</td>
<td>26.42</td>
<td>83.55</td>
</tr>
<tr>
<td>100</td>
<td>6.33</td>
<td>15.98</td>
<td>25.24</td>
<td>83.42</td>
<td>6.67</td>
<td>16.22</td>
<td>25.74</td>
<td>84.21</td>
</tr>
</tbody>
</table>

Table 91: reno-sce-reno-sce Mean TCP RTT (ms); Columns: netem bidirectional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>71.92</td>
<td>72.60</td>
<td>85.12</td>
<td>140.88</td>
<td>71.48</td>
<td>71.82</td>
<td>82.18</td>
<td>138.74</td>
</tr>
<tr>
<td>5</td>
<td>14.11</td>
<td>24.23</td>
<td>34.56</td>
<td>92.98</td>
<td>14.22</td>
<td>24.21</td>
<td>35.30</td>
<td>92.20</td>
</tr>
<tr>
<td>10</td>
<td>9.86</td>
<td>19.83</td>
<td>29.38</td>
<td>86.98</td>
<td>9.89</td>
<td>19.80</td>
<td>29.32</td>
<td>86.79</td>
</tr>
<tr>
<td>50</td>
<td>6.63</td>
<td>17.01</td>
<td>26.57</td>
<td>82.78</td>
<td>6.69</td>
<td>16.95</td>
<td>26.48</td>
<td>82.69</td>
</tr>
<tr>
<td>100</td>
<td>6.49</td>
<td>16.69</td>
<td>26.68</td>
<td>85.00</td>
<td>6.53</td>
<td>16.93</td>
<td>26.69</td>
<td>85.15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>reno-sce</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dctcp</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>69.23</td>
<td>66.60</td>
<td>78.75</td>
<td>146.62</td>
<td>67.72</td>
<td>67.64</td>
<td>79.32</td>
<td>139.43</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>13.35</td>
<td>24.63</td>
<td>36.42</td>
<td>98.92</td>
<td>13.90</td>
<td>23.35</td>
<td>32.95</td>
<td>92.69</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>8.00</td>
<td>19.99</td>
<td>30.77</td>
<td>94.50</td>
<td>7.85</td>
<td>18.12</td>
<td>27.44</td>
<td>87.54</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>8.06</td>
<td>19.71</td>
<td>31.42</td>
<td>88.87</td>
<td>7.06</td>
<td>16.79</td>
<td>26.70</td>
<td>85.95</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>8.34</td>
<td>19.77</td>
<td>32.99</td>
<td>91.65</td>
<td>7.55</td>
<td>16.70</td>
<td>27.21</td>
<td>87.35</td>
<td></td>
</tr>
</tbody>
</table>

Table 92: reno-sce-dctcp Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>reno-sce</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dctcp-sce</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>71.87</td>
<td>72.66</td>
<td>83.24</td>
<td>139.33</td>
<td>72.10</td>
<td>73.13</td>
<td>86.11</td>
<td>136.98</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>15.51</td>
<td>24.77</td>
<td>34.98</td>
<td>91.65</td>
<td>15.39</td>
<td>23.89</td>
<td>34.00</td>
<td>93.84</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>9.89</td>
<td>19.46</td>
<td>28.37</td>
<td>87.12</td>
<td>9.73</td>
<td>19.48</td>
<td>30.06</td>
<td>89.37</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>6.53</td>
<td>16.33</td>
<td>25.97</td>
<td>82.70</td>
<td>6.66</td>
<td>17.52</td>
<td>27.41</td>
<td>83.22</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>6.81</td>
<td>16.30</td>
<td>26.07</td>
<td>83.48</td>
<td>6.96</td>
<td>16.88</td>
<td>26.35</td>
<td>84.50</td>
<td></td>
</tr>
</tbody>
</table>

Table 93: reno-sce-dctcp-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dctcp</td>
<td>dctcp-sce</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>68.18</td>
<td>68.95</td>
<td>80.54</td>
<td>143.93</td>
<td>69.62</td>
<td>69.18</td>
<td>80.52</td>
<td>141.76</td>
</tr>
<tr>
<td>5</td>
<td>15.31</td>
<td>24.50</td>
<td>34.27</td>
<td>94.45</td>
<td>15.27</td>
<td>24.16</td>
<td>33.93</td>
<td>95.11</td>
</tr>
<tr>
<td>10</td>
<td>8.30</td>
<td>18.85</td>
<td>28.20</td>
<td>87.75</td>
<td>8.44</td>
<td>18.78</td>
<td>28.46</td>
<td>87.50</td>
</tr>
<tr>
<td>50</td>
<td>7.47</td>
<td>16.58</td>
<td>26.42</td>
<td>85.35</td>
<td>7.49</td>
<td>16.75</td>
<td>26.67</td>
<td>85.34</td>
</tr>
<tr>
<td>100</td>
<td>7.64</td>
<td>16.55</td>
<td>26.77</td>
<td>86.26</td>
<td>7.52</td>
<td>17.00</td>
<td>26.75</td>
<td>86.42</td>
</tr>
</tbody>
</table>

Table 94: dctcp-dctcp Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dctcp</td>
<td>dctcp-sce</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>66.02</td>
<td>66.25</td>
<td>78.29</td>
<td>139.85</td>
<td>67.98</td>
<td>65.82</td>
<td>77.89</td>
<td>145.81</td>
</tr>
<tr>
<td>5</td>
<td>13.88</td>
<td>207.36</td>
<td>32.89</td>
<td>92.04</td>
<td>13.55</td>
<td>380.51</td>
<td>35.59</td>
<td>98.73</td>
</tr>
<tr>
<td>10</td>
<td>7.81</td>
<td>80.88</td>
<td>27.60</td>
<td>87.58</td>
<td>7.95</td>
<td>154.57</td>
<td>30.74</td>
<td>93.93</td>
</tr>
<tr>
<td>50</td>
<td>6.98</td>
<td>16.12</td>
<td>26.46</td>
<td>85.87</td>
<td>8.00</td>
<td>19.22</td>
<td>32.18</td>
<td>88.74</td>
</tr>
<tr>
<td>100</td>
<td>7.72</td>
<td>16.92</td>
<td>27.67</td>
<td>86.88</td>
<td>8.67</td>
<td>19.60</td>
<td>32.95</td>
<td>91.79</td>
</tr>
</tbody>
</table>

Table 95: dctcp-dctcp-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
Table 96: dctcp-sce-dctcp-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

A.8. Two-Flow TCP RTT (Cake "triple-isolate")

Table 97: cubic-cubic Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
<table>
<thead>
<tr>
<th>Netem Bi-Directional Delay (ms)</th>
<th>Cake-limited Bandwidth (Mbit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>cubic</td>
<td>reno</td>
</tr>
<tr>
<td>1</td>
<td>90.34</td>
</tr>
<tr>
<td>5</td>
<td>17.62</td>
</tr>
<tr>
<td>10</td>
<td>8.49</td>
</tr>
<tr>
<td>50</td>
<td>6.48</td>
</tr>
<tr>
<td>100</td>
<td>6.15</td>
</tr>
</tbody>
</table>

Table 98: cubic-reno Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th>Netem Bi-Directional Delay (ms)</th>
<th>Cake-limited Bandwidth (Mbit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>cubic</td>
<td>reno-sce</td>
</tr>
<tr>
<td>1</td>
<td>92.71</td>
</tr>
<tr>
<td>5</td>
<td>16.87</td>
</tr>
<tr>
<td>10</td>
<td>8.53</td>
</tr>
<tr>
<td>50</td>
<td>6.55</td>
</tr>
<tr>
<td>100</td>
<td>6.03</td>
</tr>
</tbody>
</table>

Table 99: cubic-reno-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
### Table 100: cubic-dctcp Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>92.60</td>
<td>89.79</td>
<td>87.75</td>
<td>135.20</td>
<td>97.84</td>
<td>96.66</td>
<td>96.75</td>
<td>149.61</td>
</tr>
<tr>
<td>5</td>
<td>14.45</td>
<td>28.19</td>
<td>37.86</td>
<td>91.20</td>
<td>19.07</td>
<td>25.58</td>
<td>35.17</td>
<td>94.03</td>
</tr>
<tr>
<td>10</td>
<td>8.63</td>
<td>18.12</td>
<td>28.34</td>
<td>86.15</td>
<td>8.16</td>
<td>17.58</td>
<td>27.57</td>
<td>87.68</td>
</tr>
<tr>
<td>50</td>
<td>6.57</td>
<td>15.82</td>
<td>24.40</td>
<td>82.81</td>
<td>6.80</td>
<td>16.42</td>
<td>25.94</td>
<td>84.05</td>
</tr>
<tr>
<td>100</td>
<td>6.24</td>
<td>15.32</td>
<td>23.81</td>
<td>82.84</td>
<td>6.52</td>
<td>16.12</td>
<td>25.83</td>
<td>82.92</td>
</tr>
</tbody>
</table>

### Table 101: cubic-dctcp-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>92.73</td>
<td>87.54</td>
<td>87.60</td>
<td>126.28</td>
<td>50.28</td>
<td>49.77</td>
<td>49.69</td>
<td>115.82</td>
</tr>
<tr>
<td>5</td>
<td>14.77</td>
<td>24.31</td>
<td>34.63</td>
<td>89.26</td>
<td>9.54</td>
<td>20.04</td>
<td>29.91</td>
<td>93.48</td>
</tr>
<tr>
<td>10</td>
<td>8.46</td>
<td>18.88</td>
<td>28.82</td>
<td>85.49</td>
<td>6.87</td>
<td>16.46</td>
<td>26.73</td>
<td>88.22</td>
</tr>
<tr>
<td>50</td>
<td>6.61</td>
<td>17.05</td>
<td>24.37</td>
<td>82.43</td>
<td>4.19</td>
<td>13.11</td>
<td>22.72</td>
<td>82.28</td>
</tr>
<tr>
<td>100</td>
<td>6.41</td>
<td>15.44</td>
<td>23.87</td>
<td>82.35</td>
<td>3.66</td>
<td>12.84</td>
<td>22.70</td>
<td>82.27</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>80</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>80</td>
</tr>
<tr>
<td>---</td>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>0</td>
<td>reno</td>
<td>reno</td>
<td>reno</td>
<td>reno</td>
<td>reno</td>
<td>reno</td>
<td>reno</td>
<td>reno</td>
</tr>
<tr>
<td>1</td>
<td>68.39</td>
<td>66.58</td>
<td>73.72</td>
<td>142.08</td>
<td>68.61</td>
<td>66.66</td>
<td>73.74</td>
<td>142.94</td>
</tr>
<tr>
<td>5</td>
<td>15.73</td>
<td>25.66</td>
<td>38.41</td>
<td>92.20</td>
<td>14.62</td>
<td>25.71</td>
<td>37.82</td>
<td>92.07</td>
</tr>
<tr>
<td>10</td>
<td>8.48</td>
<td>20.48</td>
<td>29.68</td>
<td>86.62</td>
<td>8.86</td>
<td>20.37</td>
<td>29.82</td>
<td>86.93</td>
</tr>
<tr>
<td>50</td>
<td>6.58</td>
<td>15.60</td>
<td>23.71</td>
<td>82.41</td>
<td>6.54</td>
<td>15.64</td>
<td>23.77</td>
<td>82.36</td>
</tr>
<tr>
<td>100</td>
<td>6.32</td>
<td>14.10</td>
<td>23.44</td>
<td>82.17</td>
<td>6.36</td>
<td>14.13</td>
<td>23.13</td>
<td>81.99</td>
</tr>
</tbody>
</table>

Table 102: reno-reno Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>reno</td>
<td>reno</td>
<td>reno</td>
<td>reno</td>
<td>reno</td>
<td>reno</td>
<td>reno</td>
<td>reno</td>
</tr>
<tr>
<td>1</td>
<td>68.77</td>
<td>66.59</td>
<td>73.78</td>
<td>137.04</td>
<td>50.65</td>
<td>50.32</td>
<td>50.23</td>
<td>116.09</td>
</tr>
<tr>
<td>5</td>
<td>16.12</td>
<td>24.59</td>
<td>37.04</td>
<td>90.48</td>
<td>9.41</td>
<td>19.69</td>
<td>30.14</td>
<td>91.04</td>
</tr>
<tr>
<td>10</td>
<td>8.46</td>
<td>19.67</td>
<td>31.33</td>
<td>86.72</td>
<td>6.76</td>
<td>16.53</td>
<td>26.33</td>
<td>85.41</td>
</tr>
<tr>
<td>50</td>
<td>6.59</td>
<td>15.65</td>
<td>23.98</td>
<td>82.46</td>
<td>4.43</td>
<td>13.54</td>
<td>23.21</td>
<td>82.73</td>
</tr>
<tr>
<td>100</td>
<td>6.35</td>
<td>14.57</td>
<td>23.37</td>
<td>82.16</td>
<td>4.11</td>
<td>13.62</td>
<td>23.45</td>
<td>82.85</td>
</tr>
</tbody>
</table>

Table 103: reno-reno-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
|   | 0    | 10   | 20   | 80   | 0    | 10   | 20   | 80   |
|---+-----+-----+-----+-----+-----+-----+-----+-----|
|   | reno |     |     |     |     |     |     |     |
| 1 | 68.04| 66.19| 73.70| 140.21| 96.81| 96.80| 96.58| 143.33|
| 5 | 16.18| 26.99| 38.88| 92.89 | 18.99| 24.82| 35.27| 93.97 |
|10 | 8.59 |19.99 |30.15 |86.79 | 8.27 | 17.92| 27.99| 88.08 |
|50 | 6.38 | 15.57| 23.94| 82.44 | 6.78 | 16.46| 25.77| 83.82 |
|100| 6.24 | 14.72| 23.45| 86.79 | 6.30 | 16.11| 24.98| 82.96 |

Table 104: reno-dctcp Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

|   | 0    | 10   | 20   | 80   | 0    | 10   | 20   | 80   |
|---+-----+-----+-----+-----+-----+-----+-----+-----|
|   | reno |     |     |     |     |     |     |     |
|10 | 69.06|67.33|65.57|133.89|50.17|49.67 |49.59|115.33|
| 5 | 16.04|24.23|36.62| 89.94| 9.57 | 19.90|30.12| 93.42|
|10 | 8.51 |19.93|29.67| 85.60| 6.65 | 16.53|26.99| 87.11|
|50 | 6.47 | 16.89|24.02| 82.35| 4.19 | 13.04|22.72| 82.35|
|100| 6.35 | 14.57|23.39| 81.90| 3.74 | 12.59|22.53| 82.02|

Table 105: reno-dctcp-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)
### Table 106: reno-sce-reno-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50.10</td>
<td>50.27</td>
<td>50.22</td>
<td>117.12</td>
</tr>
<tr>
<td>5</td>
<td>9.54</td>
<td>19.29</td>
<td>29.77</td>
<td>88.86</td>
</tr>
<tr>
<td>10</td>
<td>6.89</td>
<td>16.55</td>
<td>26.16</td>
<td>85.44</td>
</tr>
<tr>
<td>50</td>
<td>4.50</td>
<td>13.31</td>
<td>22.99</td>
<td>82.65</td>
</tr>
<tr>
<td>100</td>
<td>4.25</td>
<td>13.89</td>
<td>23.70</td>
<td>83.01</td>
</tr>
</tbody>
</table>

### Table 107: reno-sce-dctcp Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>49.93</td>
<td>50.27</td>
<td>50.22</td>
<td>119.15</td>
</tr>
<tr>
<td>5</td>
<td>9.45</td>
<td>19.45</td>
<td>30.20</td>
<td>92.32</td>
</tr>
<tr>
<td>10</td>
<td>6.71</td>
<td>16.62</td>
<td>26.26</td>
<td>85.57</td>
</tr>
<tr>
<td>50</td>
<td>4.42</td>
<td>13.69</td>
<td>23.24</td>
<td>82.83</td>
</tr>
<tr>
<td>100</td>
<td>4.07</td>
<td>13.79</td>
<td>23.61</td>
<td>82.79</td>
</tr>
</tbody>
</table>
### Table 108: reno-sce-dctcp-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50.11</td>
<td>50.26</td>
<td>50.22</td>
<td>115.48</td>
<td>50.26</td>
<td>49.68</td>
<td>49.61</td>
<td>114.34</td>
</tr>
<tr>
<td>5</td>
<td>9.52</td>
<td>19.16</td>
<td>29.74</td>
<td>88.60</td>
<td>9.61</td>
<td>19.32</td>
<td>29.60</td>
<td>91.03</td>
</tr>
<tr>
<td>10</td>
<td>6.49</td>
<td>16.48</td>
<td>25.83</td>
<td>84.54</td>
<td>6.77</td>
<td>16.38</td>
<td>26.33</td>
<td>87.03</td>
</tr>
<tr>
<td>50</td>
<td>4.41</td>
<td>13.17</td>
<td>22.98</td>
<td>82.60</td>
<td>4.12</td>
<td>13.08</td>
<td>22.99</td>
<td>82.38</td>
</tr>
<tr>
<td>100</td>
<td>4.16</td>
<td>13.48</td>
<td>23.17</td>
<td>82.59</td>
<td>3.78</td>
<td>12.83</td>
<td>22.57</td>
<td>82.04</td>
</tr>
</tbody>
</table>

### Table 109: dctcp-dctcp Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>97.66</td>
<td>96.66</td>
<td>96.60</td>
<td>151.58</td>
<td>97.03</td>
<td>96.71</td>
<td>96.65</td>
<td>150.04</td>
</tr>
<tr>
<td>5</td>
<td>18.76</td>
<td>25.73</td>
<td>35.30</td>
<td>93.14</td>
<td>19.22</td>
<td>25.75</td>
<td>35.17</td>
<td>93.24</td>
</tr>
<tr>
<td>10</td>
<td>8.15</td>
<td>18.04</td>
<td>27.99</td>
<td>87.51</td>
<td>8.36</td>
<td>17.57</td>
<td>27.57</td>
<td>87.54</td>
</tr>
<tr>
<td>50</td>
<td>6.62</td>
<td>16.44</td>
<td>25.70</td>
<td>83.54</td>
<td>6.56</td>
<td>16.35</td>
<td>25.91</td>
<td>84.12</td>
</tr>
<tr>
<td>100</td>
<td>6.46</td>
<td>16.25</td>
<td>26.01</td>
<td>82.89</td>
<td>6.56</td>
<td>16.07</td>
<td>25.86</td>
<td>82.93</td>
</tr>
</tbody>
</table>

Table 110: dctcp-dctcp-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dctcp-sce</td>
<td></td>
<td></td>
<td></td>
<td>dctcp-sce</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>97.80</td>
<td>96.84</td>
<td>96.75</td>
<td>137.02</td>
<td>50.27</td>
<td>49.76</td>
<td>49.67</td>
<td>114.01</td>
</tr>
<tr>
<td>5</td>
<td>18.89</td>
<td>23.68</td>
<td>33.54</td>
<td>91.13</td>
<td>9.53</td>
<td>19.80</td>
<td>29.74</td>
<td>93.86</td>
</tr>
<tr>
<td>10</td>
<td>8.09</td>
<td>17.86</td>
<td>28.37</td>
<td>85.94</td>
<td>6.63</td>
<td>16.42</td>
<td>26.54</td>
<td>87.48</td>
</tr>
<tr>
<td>50</td>
<td>6.74</td>
<td>16.51</td>
<td>25.50</td>
<td>83.04</td>
<td>4.06</td>
<td>13.08</td>
<td>22.68</td>
<td>82.42</td>
</tr>
<tr>
<td>100</td>
<td>6.44</td>
<td>16.27</td>
<td>26.02</td>
<td>83.05</td>
<td>3.74</td>
<td>12.88</td>
<td>22.68</td>
<td>82.08</td>
</tr>
</tbody>
</table>

Table 111: dctcp-sce-dctcp-sce Mean TCP RTT (ms); Columns: netem bi-directional Delay (ms); Rows: Cake-limited Bandwidth (Mbit)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
<th>0</th>
<th>10</th>
<th>20</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dctcp-sce</td>
<td></td>
<td></td>
<td></td>
<td>dctcp-sce</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>50.89</td>
<td>49.74</td>
<td>49.64</td>
<td>116.00</td>
<td>50.28</td>
<td>49.72</td>
<td>49.68</td>
<td>113.76</td>
</tr>
<tr>
<td>10</td>
<td>6.66</td>
<td>16.46</td>
<td>26.39</td>
<td>86.01</td>
<td>6.77</td>
<td>16.44</td>
<td>26.44</td>
<td>86.28</td>
</tr>
<tr>
<td>50</td>
<td>4.14</td>
<td>12.85</td>
<td>22.68</td>
<td>82.30</td>
<td>4.19</td>
<td>13.06</td>
<td>22.55</td>
<td>82.25</td>
</tr>
<tr>
<td>100</td>
<td>3.78</td>
<td>12.57</td>
<td>22.34</td>
<td>81.86</td>
<td>3.80</td>
<td>12.56</td>
<td>22.26</td>
<td>81.96</td>
</tr>
</tbody>
</table>

Authors’ Addresses

Peter G. Heist  
Redacted  
463 11 Liberec 30  
Czech Republic  

Email: pete@heistp.net
Abstract

As communication devices become more hybrid, smart devices include more media-rich communication applications, and the boundaries between telecommunication and other applications becomes less clear. Simultaneously, as the end-devices become more mobile, application traffic transits more often between enterprise networks, the Internet, and cellular telecommunication networks. In this context, it is crucial that quality of service be aligned between these different environments. However, this is not always the case by default, and cellular communication networks use a different QoS nomenclature from the Internet and enterprise networks. This document specifies a set of 3rd Generation Partnership Project (3GPP) Quality of Service (QoS) Class Identifiers (QCI) to Differentiated Services Code Point (DSCP) mappings, to reconcile the marking recommendations offered by the 3GPP with the recommendations offered by the IETF, so as to maintain a consistent QoS treatment between cellular networks and the 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 http://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 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 (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
   1.1. Related Work ...................................... 4
   1.2. Applicability Statement .......................... 4
   1.3. Document Organization ............................. 5
   1.4. Requirements language ............................... 5
   1.5. Terminology Used in this Document ................. 5
2. Service Comparison and Default Interoperation of Diffserv and 3GPP LTE ............................................ 7
   2.1. Diffserv Domain Boundaries ........................ 7
   2.2. QCI and Bearer Model in 3GPP ....................... 7
   2.3. QCI Definition and Logic ............................ 9
       2.3.1. Conversational ............................... 9
       2.3.2. Streaming .................................... 9
       2.3.3. Interactive ................................. 10
       2.3.4. Background .................................. 10
   2.4. QCI implementations .................................. 13
   2.5. GSMA IPX Guidelines Interpretation and Conflicts .... 13
3. P-GW Device Marking and Mapping Capability Recommendations ... 14
4. DSCP-to-QCI Mapping Recommendations .......................... 15
   4.1. Control Traffic .................................... 15
       4.1.1. Network Control Protocols ................. 15
       4.1.2. Operations, Administration, and Maintenance (OAM) .. 16
   4.2. User Traffic ...................................... 16
       4.2.1. Telephony .................................... 16
       4.2.2. Signaling .................................... 17
       4.2.3. Multimedia Conferencing .................... 17
       4.2.4. Real-Time Interactive ....................... 18
       4.2.5. Multimedia Streaming ....................... 18
       4.2.6. Broadcast Video ............................. 19
       4.2.7. Low-Latency Data ............................ 19
       4.2.8. High-Throughput Data ....................... 20
1. Introduction

3GPP has become the preferred set of standards to define cellular communication principles and protocols. With the augmented capabilities of smartphones, cellular networks increasingly carry non-communication traffic and interconnect with the Internet and Enterprise IP networks. The access networks defined by the 3GPP present several design challenges for ensuring end-to-end quality of service when these networks interconnect with the Internet or to enterprise networks. Some of these challenges relate to the nature of the cellular network itself, being centrally controlled, collision-free and primarily designed around subscription level and associated services, while other challenges relate to the fact that the 3GPP standards are not administered by the same standards body as Internet protocols. While 3GPP has developed tools to enabled QoS over cellular networks, little guidance exists on how to maintain consistency of QoS treatment between cellular networks and the Internet, or IP-based Enterprise networks. The purpose of this document is to provide such guidance.
1.1. Related Work

Several RFCs outline Diffserv QoS recommendations over IP networks, including:


Note: [RFC4594] is intended to be viewed as a framework for supporting Diffserv in any network, regardless of the underlying data-link or physical layer protocols. As such, its principles could apply to IP traffic carried over cellular DataLink and Physical Layer mediums. Additionally, the principles of [RFC4594] apply to any traffic entering the Internet, regardless of its original source location. Thus, [RFC4594] describes different types of traffic expected in IP networks and provides guidance as to what DSCP marking(s) should be associated with each traffic type. As such, this document draws heavily on [RFC4594], as well as [RFC5127], and [RFC8100].

In turn, the relevant standard for cellular QoS is 3GPP [TS 23.107], which defines more than 1600 General Packet Radio Service (GPRS) QoS profiles across multiple classes and associated attributes. As this quantity is large and source of potential complexity, the 3GPP Technical Specification Group Services and System Aspects, defining the Policy Charging Control Architecture, leverages a subset of QoS profiles used as QoS Class Identifiers (QCI). This document draws on this specification, which is being progressively updated; the current version of which (at the time of writing) is 3GPP [TS 23.203] v16.0.

1.2. Applicability Statement

This document is applicable to the use of Differentiated Services that interconnect with 3GPP cellular networks (referred to as cellular, throughout this document, for simplicity). These guidelines are applicable whether cellular network endpoints are IP-enabled, in which case these guidelines can apply end-to-end, starting from the endpoint operating system, or whether cellular network endpoints are either not IP-enabled, or do not enable QoS, in which case these guidelines apply at the interconnection point between the cellular access network and the Internet or IP network.
Such interconnection point can commonly occur at the infrastructure Radio Unit (eNodeB), within the infrastructure core network (CN), or at the edge of the core network toward the Internet or an Enterprise IP network, for example within the Packet Data Network Gateway (P-GW).

1.3. Document Organization

This document is organized as follows:

- Section 2 introduces the QoS logic marking applicable to each domain. We introduce the general logic of Diffserv and the notion of domain boundary. We then examine the 3GPP QoS logic, detailing the concept of bearer and QCI, and showing how QCIs are implemented and used.

- Section 3 provides general recommendations for QoS support at the 3GPP / Diffserv domains boundaries.

- Section 4 proposes a Diffserv to QCI translation scheme, so as to carry DSCP values to the 3GPP domains when QCI must be used.

- Section 5 proposes a reverse mapping, from QCI to Diffserv. As many QCIs intents do not match existing DSCP values, new DSCP values are proposed wherever needed.

- Section 6 underlines the resulting IANA requirements for this mapping.

- Section 7 and Section 8 examine the security consequences of these new mapping schemes.

1.4. 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.

1.5. Terminology Used in this Document

Key terminology used in this document includes:

- EPS Bearer: a path that user traffic (IP flows) uses between the UE and the PGW.
GGSN: Gateway GPRS Support Node, responsible for the internetworking between the GPRS network and external networks. PGW performs the GGSN functionalities in EPC.

IP BS Manager: Internet Protocol Bearer Service Manager, a function that manages the IP bearer services. Part of this function can include translation of QoS parameters between EPS and external networks.

UE: User Equipment, the end-device.

EPS Session: a PDN connection, comprised of one or more IP flows, that a UE established and maintains to the EPS.

SAE: System Architecture Evolution.

RAN: Radio access network, the radio segment of the LTE network EPS.

EPC: Evolved Packet Core, the core segment of the LTE network EPS.

EPS: Evolved Packet System, the LTE network, comprised of the RANs and EPC.

HSS: Home Subscriber Server, the database that contains user-related and subscriber-related information.

LUS: Live Uplink Streaming, a video flow (often real-time) sent from a source to a sink.

SGW: Serving Gateway, the point of interconnection between the RAN and the EPC.

PGW: Packet Data Network Gateway, point of interconnection between the EPC and external IP networks.

MME: Mobility Management Entity: software function that handles the signaling related to mobility and security for the access network.

PCEF: Policy and Charging Enforcement Function, provides user traffic handling and QoS within the PGW.

PCRF: Policy and Charging Rules Function, a functional entity that provides policy, bandwidth and charging functions for each EPS user.
2. Service Comparison and Default Interoperation of Diffserv and 3GPP LTE

2.1. Diffserv Domain Boundaries

It is important to recognize that LTE standards allow support for principles of [RFC2475]. The user equipment (UE) application function may have no active QoS support, or may support Diffserv or IntServ functions [TS 23.207] v16 5.2.2. When Diffserv is supported, an Internet Protocol Bearer Service Manager (IP BS Manager) function integrated to the UE can translate Diffserv parameters into LTE QoS parameters (e.g. QCI). As such, the UE IP BS Manager function may act as a Diffserv domain boundary (as defined in [RFC2475]) between a Diffserv domain present within the UE networking stack and the LTE Radio Access Network.

Additionally, the P-GW interconnects the UE data plane to the external networks. The P-GW is the element that implements Gateway GPRS (General Packet Radio Service) Support Node (GGSN) functionalities in Evolved Packet Core (EPS) networks. The GGSN includes an IP BS manager function that acts as a Diffserv Edge function, and can translate Diffserv parameters to LTE QoS parameters (e.g. QCI) and vice versa.

As such, LTE standards allow the existence of a Diffserv domain within the UE and outside of the EPS boundaries. The Diffserv domain is not considered within the EPS, where QCIs are used to define and transport QoS parameters.

2.2. QCI and Bearer Model in 3GPP

It is important to note that LTE standards (4G, 5G) are an evolution of UMTS standards (2G, 3G) developed in the 1990s. As such, these standards recognize [RFC2475] (1998), but not [RFC4594] (2006). EPS networks rely on the notion of bearers. A bearer is a conduit between the UE and the P-GW, and LTE supports two types of bearers:

- GBR: Guaranteed Bit Rate bearers. These bearers allocate network resources associated to a GBR value associated to the bearer. These resources stay allocated (reserved) for the duration of the existence of the GBR bearer and the flow it carries.

- Non-GBR bearers: also called default bearers, non-GBR are bearers for which network resources are not permanently allocated during the existence of the bearer and the flow it carries. As such, one or more non-GBR bearer may share the same set of temporal resources.
Each EPS bearer is identified by a name and number, and is associated with specific QoS parameters of various types:

1. **QoS Class Identifiers (QCI).** A QCI is a scalar associated to a bearer, and is used to define the type of traffic and service expected in the bearer. [TS 23.107] v15 defines 4 basic classes: conversational, streaming, interactive and background. These classes are defined more in details in Section 2.3. Each class includes multiple types of traffic, each associated with sets of attributes, thus permitting the definition of more than 1600 different QoS profiles. [TS 23.203] v16 6.1.7.2 reduces the associated complexity by characterizing traffic based on up to 6 attributes, resulting in 21 types of traffic and their associated expected service requirements through the use of 21 scalars (QCI). Each QCI is defined in the relation to the following six performance characteristics:

2. **Resource Type (GBR or Non-GBR).**

3. **Priority:** a scalar used as a tie breaker if two packets compete for a given network resource. A lower value indicates a higher priority.

4. **Packet Delay Budget:** marks the upper bound for the time that a packet may be delayed between the UE and the PCRF (Policy and Charging Rules Function) or the PCEF function (Policy and Charging Enforcement Function) residing inside the P-GW. PCEF supports offline and online charging while PCRF is real-time. Either component, being in charge of policing and charging, can determine resource reservation actions and policies.

5. **Packet Error Loss Rate,** defines an upper bound for a rate of non-congestion related packet losses. The purpose of the PELR is to allow for appropriate link layer protocol configurations when needed.

6. **Maximum Burst Size** (only for some GBR QCIs), defines the amount of data which the Radio Access Network (RAN) is expected to deliver within the part of the Packet Delay Budget allocated to the link between the UE and the radio base station. If more data is transmitted from the application, the Packet Delay Budget may be exceeded.

7. **Data rate Averaging Window** (only for some GBR QCIs), defines the ‘sliding window’ duration over which the GBR and MBR are calculated.
Although [TS 23.203] v16 6.1.7.2 associates each QCI with up to 6 characteristics, it is clear that these characteristics are constrained by bandwidth allocation, in particular on the radio link that are associated with three commonly used parameters:

1. Maximum Bit Rate (MBR), only valid for GBR bearers, defines the maximum sustained traffic rate that the bearer can support.

2. Guaranteed Bit Rate (GBR), only valid for GBR bearers, defines the minimum traffic rate reserved for the bearer.

3. Aggregate MBR (AMBR), defines the total amount of bit rate available for a group of non-GBR bearers. AMBR is often used to provide differentiated service levels to different types of customers.

2.3. QCI Definition and Logic

[TS 23.107] v15 6.3 defines four possible traffic classes. These four general classes are used as the foundation from which QCI categories are defined in [TS 23.203]. The categorization is made around the notion of sensitivity to delay.

2.3.1. Conversational

The conversational class is intended to carry real-time traffic flows. The expectation of such class is a live conversation between two humans or a group. Examples of such flows include [TS 23.107] v15 6.3.1 telephony speech, but also VoIP and video conferencing. Video conference would be seen as a different class from telephony in the DiffServ model. However, 3GPP positions them in the same general class, as all of them include live conversations. Sensitivity to delay is high because of the real-time nature of the flows. The time relation between the stream entities have to be preserved (to maintain the same experience for all flows and all parties involved in the conversation).

2.3.2. Streaming

The streaming class is intended for flows where the user is watching real-time video, or listening to real-time audio (or both). The real-time data flow is always aiming at a live (human) destination. It is important to note that the Streaming class is intended to be both a real-time flow and a one-way transport. Two-way real-time traffic belongs to the conversational class, and non-real-time flows belong to the interactive or the background classes. The delay sensitivity is lower than that of Conversational flows, because it is expected that the receiving end includes a time alignment function.
As the flow is unidirectional, variations in delay do not conversely affect the user experience as long as the variation is within the alignment function boundaries.

2.3.3. Interactive

The interactive class is intended for flows where a machine or human is requesting data from a remote equipment (e.g. a server). Examples of human interaction with the remote equipment are: web browsing, data base retrieval, server access. Examples of machines interaction with remote equipment are: polling for measurement records and automatic data base enquiries (tele-machines). Delay sensitivity is average, and is based on round trip time (overall time between emission of the request and reception of the response).

2.3.4. Background

The background class applies to flows where the equipment is sending or receiving data files without direct user interaction (e.g. emails, SMS, database transfers etc.) As such, delay sensitivity is low. Background is described as delivery-time insensitive.

Based upon the above principles, [TS 23.203] has defined several QCI. [TS 23.203] Release 16 6.1.7-A defines 26 QCI:

<table>
<thead>
<tr>
<th>QCI</th>
<th>Resource Type</th>
<th>Priority Level</th>
<th>Packet Delay Budget</th>
<th>Packet Error Loss</th>
<th>Example Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GBR</td>
<td>2</td>
<td>100 ms</td>
<td>10.E-2</td>
<td>Conversational Voice</td>
</tr>
<tr>
<td>2</td>
<td>GBR</td>
<td>4</td>
<td>150 ms</td>
<td>10.E-3</td>
<td>Conversational Video (Live Streaming)</td>
</tr>
<tr>
<td>3</td>
<td>GBR</td>
<td>3</td>
<td>50 ms</td>
<td>10.E-3</td>
<td>Real Time Gaming, V2X messages, Electricity distribution (medium voltage) Process automation (monitoring)</td>
</tr>
<tr>
<td>4</td>
<td>GBR</td>
<td>5</td>
<td>300 ms</td>
<td>10.E-6</td>
<td>Non-Conversational Video (Buffered Streaming)</td>
</tr>
<tr>
<td>65</td>
<td>GBR</td>
<td>0.7</td>
<td>75 ms</td>
<td>10.E-2</td>
<td>Mission Critical</td>
</tr>
<tr>
<td>QCI</td>
<td>Area</td>
<td>Priority</td>
<td>Delay</td>
<td>Reliability</td>
<td>Service Type</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>----------</td>
<td>-------</td>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td>66</td>
<td>GBR</td>
<td>2</td>
<td>100 ms</td>
<td>10.E-2</td>
<td>Non-Mission-Critical user plane Push To Talk voice (e.g., MCPTT)</td>
</tr>
<tr>
<td>67</td>
<td>GBR</td>
<td>1.5</td>
<td>100 ms</td>
<td>10.E-3</td>
<td>Mission Critical Video user plane</td>
</tr>
<tr>
<td>75</td>
<td>GBR</td>
<td>2.5</td>
<td>50 ms</td>
<td>10.E-2</td>
<td>V2X messages</td>
</tr>
<tr>
<td>71</td>
<td>GBR</td>
<td>5.6</td>
<td>150 ms</td>
<td>10.E-6</td>
<td>&quot;Live&quot; Uplink Streaming</td>
</tr>
<tr>
<td>72</td>
<td>GBR</td>
<td>5.6</td>
<td>300 ms</td>
<td>10.E-4</td>
<td>&quot;Live&quot; Uplink Streaming</td>
</tr>
<tr>
<td>73</td>
<td>GBR</td>
<td>5.6</td>
<td>300 ms</td>
<td>10.E-8</td>
<td>&quot;Live&quot; Uplink Streaming</td>
</tr>
<tr>
<td>74</td>
<td>GBR</td>
<td>5.6</td>
<td>500 ms</td>
<td>10.E-8</td>
<td>&quot;Live&quot; Uplink Streaming</td>
</tr>
<tr>
<td>76</td>
<td>GBR</td>
<td>5.6</td>
<td>500 ms</td>
<td>10.E-4</td>
<td>&quot;Live&quot; Uplink Streaming</td>
</tr>
<tr>
<td>5</td>
<td>Non-GBR</td>
<td>1</td>
<td>100 ms</td>
<td>10.E-6</td>
<td>IMS Signalling</td>
</tr>
<tr>
<td>6</td>
<td>Non-GBR</td>
<td>6</td>
<td>300 ms</td>
<td>10.E-6</td>
<td>Video (Buffered Streaming) TCP-based (e.g. www, email, chat, ftp, p2p file sharing, progressive video)</td>
</tr>
<tr>
<td>7</td>
<td>Non-GBR</td>
<td>7</td>
<td>100 ms</td>
<td>10.E-3</td>
<td>Voice, Video (live streaming), interactive gaming</td>
</tr>
<tr>
<td>8</td>
<td>Non-GBR</td>
<td>8</td>
<td>300 ms</td>
<td>10.E-6</td>
<td>Video (buffered streaming) TCP-based (e.g. www, email, chat, ftp, p2p file sharing, progressive video)</td>
</tr>
<tr>
<td>QCI</td>
<td>Class</td>
<td>Value</td>
<td>Delay</td>
<td>Per</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>--------</td>
<td>-------</td>
<td>--------</td>
<td>-----</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>9</td>
<td>Non-GBR</td>
<td>9</td>
<td>300 ms</td>
<td>10E-6</td>
<td>Same as 8</td>
</tr>
<tr>
<td>69</td>
<td>Non-GBR</td>
<td>0.5</td>
<td>60 ms</td>
<td>10E-6</td>
<td>Mission Critical delay sensitive signalling (e.g., MC-PTT signalling, MC Video signalling)</td>
</tr>
<tr>
<td>70</td>
<td>Non-GBR</td>
<td>5.5</td>
<td>200 ms</td>
<td>10E-6</td>
<td>Mission Critical Data (e.g. example services are the same as QCI 6/8/9)</td>
</tr>
<tr>
<td>79</td>
<td>Non-GBR</td>
<td>6.5</td>
<td>50 ms</td>
<td>10E-2</td>
<td>V2X messages</td>
</tr>
<tr>
<td>80</td>
<td>Non-GBR</td>
<td>6.8</td>
<td>10 ms</td>
<td>10E-2</td>
<td>Low latency eMMB applications (TCP/UDP-based); augmented reality</td>
</tr>
<tr>
<td>82</td>
<td>GBR</td>
<td>1.9</td>
<td>10 ms</td>
<td>10E-6</td>
<td>Discrete automation (small packets)</td>
</tr>
<tr>
<td>83</td>
<td>GBR</td>
<td>2.2</td>
<td>10 ms</td>
<td>10E-4</td>
<td>Discrete automation (large packets)</td>
</tr>
<tr>
<td>84</td>
<td>GBR</td>
<td>2.4</td>
<td>30 ms</td>
<td>10E-5</td>
<td>Intelligent Transport Systems</td>
</tr>
<tr>
<td>85</td>
<td>GBR</td>
<td>2.1</td>
<td>5 ms</td>
<td>10E-5</td>
<td>Electricity Distribution - High Voltage</td>
</tr>
</tbody>
</table>

Several QCI's cover the same application types. For example, QCI's 6, 8 and 9 all apply to buffered streaming video and web applications. However, LTE context distinguishes several types of customers and environments. As such, QCI 6 can be used for the prioritization of non-real-time data (i.e. most typically TCP-based services/applications) of MPS (multimedia priority services) subscribers, when the network supports MPS. QCI 8 can be used for a dedicated "premium bearer" (e.g. associated with premium content) for any subscriber or subscriber group, while QCI 9 can be used for the default bearer for non-privileged subscribers.
2.4. QCI implementations

[TS 23.203] v16 defines multiple QCI s. However, a UE or a EPS does not need to implement all supported QCI s, even when all matching types of traffic are expected between the UE and the network. In practical implementations, it is common for an EPS to implement one GBR bearer where at least QCI 1 is directed (and optionally other GBR QCI s), and another default bearer where all other traffic to and from the same UE is directed. The QCI associated to that second bearer may depend on the subscriber category. As such, the QCI listed in Section 2.3 are indicative of performance and traffic type classifications, and are not strict in their implementation mandate.

2.5. GSMA IPX Guidelines Interpretation and Conflicts

3GPP standards do not define or recommend any specific mapping between each QCI and Diffserv, and leaves that mapping choice to the operator of the Edge domain boundary (e.g. UE software stack developer, P-GW operator). However, 3GPP defines that "for the IP based backbone, Differentiated Services defined by IETF shall be used" ([TS 23.107] v15 6.4.7).

The GSM Association (GSMA) has published an Inter-Service Provider IP Backbone Guideline reference document [IR.34] that provides technical guidance to participating service providers for connecting IP based networks and services to achieve roaming and inter-working services. The document built upon [RFC3246] and [RFC2597], and upon the initial definition of 4 service classes in [TS 23.107] v15 to recommend a mapping to EF for conversational traffic, to AF41 for Streaming traffic, to AF31, AF21 and AF11 for different traffic in the Interactive class, and to BE for background traffic.

These GSMA Guidelines were developed without reference to existing IETF specifications for various services, referenced in Section 1.1. Additionally, the same recommendations remained while new traffic types under each 3GPP general class were added. As such, the GSMA recommendations yield to several inconsistencies with [RFC4594], including:

- Recommending EF for real-time (conversational) video, for which [RFC4594] recommends AF41.
- Recommending AF31 for DNS traffic, for which [RFC4594] recommends the standard service class (DF)
- Recommending AF31 for all types of signaling traffic, thus losing the ability to differentiate between the various types of signaling flows, as recommended in[RFC4594] section 5.1.
o Recommending AF21 for WAP browsing and WEB browsing, for which [RFC4594] recommends the High Throughput data class.

o Recommending AF11 for remote connection protocols, such as telnet or SSH, for which [RFC4594] recommends the OAM class.

o Recommending DF for file transfers, for which [RFC4594] recommends the High Throughput Data class.

o Recommending DF for email exchanges, for which [RFC4594] recommends the High Throughput Data class.

o Recommending DF for MMS exchanged over SMTP, for which [RFC4594] recommends the High Throughput Data class.

The document [IR.34] also does not provide guidance for QCIs other than 1 to 9, leaving the case of the 12 other QCIs unaddressed.

Thus, document [IR.34] conflicts with the overall Diffserv traffic-conditioning service plan, both in the services specified and the code points specified for them. As such, these two plans cannot be normalized. Rather, as discussed in [RFC2474] Section 2, the two domains (GSMA and other IP networks) are different Differentiated Services Domains separated by a Differentiated Services Boundary. At that boundary, code points from one domain are translated to code points for the other, and maybe to Default (zero) if there is no corresponding service to translate to.

3. P-GW Device Marking and Mapping Capability Recommendations

This document assumes and RECOMMENDS that all P-GWs (as the interconnects between cellular and other IP networks) and all other interconnection points between cellular and other IP networks support the ability to:

o mark DSCP, per Diffserv standards

o mark QCI, per the [TS 23.203] standard

o support fully-configurable mappings between DSCP and QCI

o process DSCP markings set by cellular endpoint devices

This document further assumes and RECOMMENDS that all cellular endpoint devices (UE) support the ability to:

o mark DSCP, per Diffserv standards
Having made the assumptions and recommendations above, it bears mentioning that while the mappings presented in this document are RECOMMENDED to replace the current common default practices (as discussed in Section 2.3 and Section 2.4), these mapping recommendations are not expected to fit every last deployment model, and as such MAY be overridden by network administrators, as needed.

4. DSCP-to-QCI Mapping Recommendations

4.1. Control Traffic

4.1.1. Network Control Protocols

The Network Control service class is used for transmitting packets between network devices (e.g., routers) that require control (routing) information to be exchanged between nodes within the administrative domain, as well as across a peering point between different administrative domains.

[RFC4594] Section 3.2 recommends that Network Control Traffic be marked CS6 DSCP. Additionally, as stated in [RFC4594] Section 3.1: "CS7 DSCP value SHOULD be reserved for future use, potentially for future routing or control protocols."

Network Control service is not directly called by any specific QCI description, because 3GPP network control does not operate over UE data channels. It should be noted that encapsulated routing protocols for encapsulated or overlay networks (e.g., VPN, Network Virtualization Overlays, etc.) are not Network Control Traffic for any physical network at the cellular space; hence, they SHOULD NOT be marked with CS6 in the first place, and are not expected to be forwarded to the cellular data plane.

However, when such network control traffic is forwarded, it is expected to receive a high priority and level of service. As such, packets marked to CS7 DSCP are RECOMMENDED to be mapped to QCI 82, thus benefiting from a dedicated bearer with low packet error loss rate (10^{-4}) and low budget delay (10 ms). Similarly, it is RECOMMENDED to map Network Control Traffic marked CS6 to QCI 82, thereby admitting it to the Discrete Automation (GBR) category with a relative priority level of 1.9.
4.1.2. Operations, Administration, and Maintenance (OAM)

The OAM (Operations, Administration, and Maintenance) service class is recommended for OAM&P (Operations, Administration, and Maintenance and Provisioning). The OAM service class can include network management protocols, such as SNMP, Secure Shell (SSH), TFTP, Syslog, etc., as well as network services, such as NTP, DNS, DHCP, etc.

[RFC4594] Section 3.3, recommends that OAM traffic be marked CS2 DSCP.

Applications using this service class require a low packet loss but are relatively not sensitive to delay. This service class is configured to provide good packet delivery for intermittent flows. As such, packets marked to CS2 are RECOMMENDED to be mapped to QCI 9, thus admitting it to the non-GBR Buffered video traffic, with a relative priority of 9.

4.2. User Traffic

User traffic is defined as packet flows between different users or subscribers. It is the traffic that is sent to or from end-terminals and that supports a very wide variety of applications and services [RFC4594] Section 4.

Network administrators can categorize their applications according to the type of behavior that they require and MAY choose to support all or a subset of the defined service classes.

4.2.1. Telephony

The Telephony service class is recommended for applications that require real-time, very low delay, very low jitter, and very low packet loss for relatively constant-rate traffic sources (inelastic traffic sources). This service class SHOULD be used for IP telephony service. The fundamental service offered to traffic in the Telephony service class is minimum jitter, delay, and packet loss service up to a specified upper bound. [RFC4594] Section 4.1 recommends that Telephony traffic be marked EF DSCP.

3GPP [TS 23.203] describes two QCIs adapted to Voice traffic: QCI 1 (GBR) and QCI 7 (non-GBR). However, Telephony traffic as intended in [RFC4594] supposes resource allocation control. Telephony SHOULD be configured to receive guaranteed forwarding resources so that all packets are forwarded quickly. The Telephony service class SHOULD be configured to use Priority Queuing system. QCI 7 does not match these conditions. As such, packets marked to EF are RECOMMENDED to
be mapped to QCI 1, thus admitting it to the GBR Conversational Voice category, with a relative priority of 2.

4.2.2. Signaling

The Signaling service class is recommended for delay-sensitive client-server (e.g., traditional telephony) and peer-to-peer application signaling. Telephony signaling includes signaling between 1) IP phone and soft-switch, 2) soft-client and soft-switch, and 3) media gateway and soft-switch as well as peer-to-peer using various protocols. This service class is intended to be used for control of sessions and applications. [RFC4594] Section 4.2 recommends that Signaling traffic be marked CS5 DSCP.

While Signaling is recommended to receive a superior level of service relative to the default class (i.e., relative to QCI 7), it does not require the highest level of service (i.e., GBR and very high priority). As such, it is RECOMMENDED to map Signaling traffic marked CS5 DSCP to QCI 4, thereby admitting it to the GBR Non-conversational video category, with a relative priority level of 5.

Note: Signaling traffic for native Voice dialer applications should be exchanged over a control channel, and is not expected to be forwarded in the data-plane. However, Signaling for non-native (OTT) applications may be carried in the data-plane. In this case, Signaling traffic is control-plane traffic from the perspective of the voice/video telephony overlay-infrastructure. As such, Signaling should be treated with preferential servicing versus other data-plane flows.

4.2.3. Multimedia Conferencing

The Multimedia Conferencing service class is recommended for applications that require real-time service for rate-adaptive traffic. [RFC4594] Section 4.3 recommends Multimedia Conferencing traffic be marked AF4x (that is, AF41, AF42, and AF43, according to the rules defined in [RFC2475]). The Diffserv model allows for three values to allow for different relative priorities of flows of the same nature.

The primary media type typically carried within the Multimedia Conferencing service class marked AF41 is video intended to be a component of a real-time exchange; as such, it is RECOMMENDED to map AF41 into the Conversational Video (Live Streaming) category, with a GBR. Specifically, it is RECOMMENDED to map AF41 to QCI 2, thereby admitting AF41 into the GBR Conversational Video, with a relative priority of 4.
AF42 is typically reserved for video intended to be a component of real-time exchange, but which criticality is less than traffic carried with a marking of AF41. As such, it is RECOMMENDED to map AF42 into the Conversational Video (Live Streaming) category, with a GBR, but a lower priority than QCI 2. Specifically, it is RECOMMENDED to map AF42 to QCI 4, thereby admitting AF42 into the GBR Conversational Video, with a relative priority of 5.

Traffic marked AF43 is typically used for real-time video exchange of lower criticality. As such, it is RECOMMENDED to map AF43 into the Conversational Video (Live Streaming) category, but without a GBR. Specifically, it is RECOMMENDED to map AF43 to QCI 7, thereby admitting AF43 to the non-GBR Voice, Video and Interactive gaming, with a relative priority of 7.

4.2.4. Real-Time Interactive

The Real-Time Interactive service class is recommended for applications that require low loss and jitter and very low delay for variable-rate inelastic traffic sources. Such applications may include inelastic video-conferencing applications, but may also include gaming applications (as pointed out in [RFC4594] Sections 2.1 through 2.3 and Section 4.4. [RFC4594] Section 4.4 recommends Real-Time Interactive traffic be marked CS4 DSCP.

The primary media type typically carried within the Real-Time Interactive service class is video; as such, it is RECOMMENDED to map this class into a low latency Category. Specifically, it is RECOMMENDED to map CS4 to QCI 80, thereby admitting Real-Time Interactive traffic into the non-GBR category Low Latency eMBB (enhanced Mobile Broadband) applications with a relative priority of 6.8. In cases where GBR is required, for example because a single bearer is allocated for all non-GBR traffic, using a GBR equivalent is also acceptable. In this case, it is RECOMMENDED to map CS4 to QCI 3, thereby admitting Real-Time Interactive traffic into the GBR category Real-time gaming, with a relative priority of 3.

4.2.5. Multimedia Streaming

The Multimedia Streaming service class is recommended for applications that require near-real-time packet forwarding of variable-rate elastic traffic sources. Typically, these flows are unidirectional. [RFC4594] Section 4.5 recommends Multimedia Streaming traffic be marked AF3x (that is, AF31, AF32, and AF33, according to the rules defined in [RFC2475]).

The primary media type typically carried within the Multimedia Streaming service class is video; as such, it is RECOMMENDED to map
this class into a Video Category. Specifically, it is RECOMMENDED to map AF31 to QCI 4, thereby admitting AF31 into the GBR Non Conversational Video category, with a relative priority of 5.

Flows marked with AF32 are expected to be of the same nature as flows marked with AF32, but with a lower criticality. As such, these flows may not require a dedicated bearer with GBR. Therefore, it is RECOMMENDED to map AF32 to QCI 6, thereby admitting AF32 traffic into the non-GBR category Video (Buffered Streaming) with a relative priority of 6.

Flows marked with AF33 are expected to be of the same nature as flows marked with AF31 and AF32, but with the lowest criticality. As such, it is RECOMMENDED to map AF33 to QCI 8, thereby admitting AF33 traffic into the non-GBR category Video (Buffered Streaming) with a relative priority of 8.

4.2.6. Broadcast Video

The Broadcast Video service class is recommended for applications that require near-real-time packet forwarding with very low packet loss of constant rate and variable-rate inelastic traffic sources. Typically, these flows are unidirectional. [RFC4594] Section 4.6 recommends Broadcast Video traffic be marked CS3 DSCP.

As directly implied by the name, the primary media type typically carried within the Broadcast Video service class is video; as such, it is RECOMMENDED to map this class into a Video Category. Specifically, it is RECOMMENDED to map CS3 to QCI 4, thereby admitting Multimedia Streaming into the GBR Non Conversational Video category, with a relative priority of 5. In cases where GBR availability is constrained, using a non-GBR equivalent is also acceptable. In this case, it is RECOMMENDED to map CS3 to QCI 6, thereby admitting Real-Time Interactive traffic into the non-GBR category Video with a relative priority of 6.

4.2.7. Low-Latency Data

The Low-Latency Data service class is recommended for elastic and time-sensitive data applications, often of a transactional nature, where a user is waiting for a response via the network in order to continue with a task at hand. As such, these flows are considered foreground traffic, with delays or drops to such traffic directly impacting user productivity. [RFC4594] Section 4.7 recommends Low-Latency Data be marked AF2x (that is, AF21, AF22, and AF23, according to the rules defined in [RFC2475]).
The primary media type typically carried within the Low-Latency Data service class is data; as such, it is RECOMMENDED to map this class into a data Category. Specifically, it is RECOMMENDED to map AF21 to QCI 70, thereby admitting AF21 into the non-GBR Mission Critical Data category, with a relative priority of 5.5.

Flows marked with AF22 are expected to be of the same nature as flows marked with AF21, but with a lower criticality. Therefore, it is RECOMMENDED to map AF22 to QCI 6, thereby admitting AF22 traffic into the non-GBR category Video and TCP-based traffic, with a relative priority of 6.

Flows marked with AF23 are expected to be of the same nature as flows marked with AF21 and AF22, but with the lowest criticality. As such, it is RECOMMENDED to map AF23 to QCI 8, thereby admitting AF23 traffic into the non-GBR category Video and TCP-based traffic, with a relative priority of 8.

It should be noted that a consequence of such classification is that AF22 is mapped to the same QCI as CS3, and AF23 is mapped to the same QCI as AF33. However, this overlap is unavoidable, as some QCIs express intents that are expressed in the Diffserv domain through distinct marking values, grouped in the 3GPP domain under the same general category.

4.2.8. High-Throughput Data

The High-Throughput Data service class is recommended for elastic applications that require timely packet forwarding of variable-rate traffic sources and, more specifically, is configured to provide efficient, yet constrained (when necessary) throughput for TCP longer-lived flows. These flows are typically not user interactive.

According to [RFC4594] Section 4.8 it can be assumed that this class will consume any available bandwidth and that packets traversing congested links may experience higher queuing delays or packet loss. It is also assumed that this traffic is elastic and responds dynamically to packet loss. [RFC4594] Section 4.8 recommends High-Throughput Data be marked AF1x (that is, AF11, AF12, and AF13, according to the rules defined in [RFC2475]).

The primary media type typically carried within the High-Throughput Data service class is data; as such, it is RECOMMENDED to map this class into a data Category. Specifically, it is RECOMMENDED to map AF11 to QCI 6, thereby admitting AF11 into the non-GBR Video and TCP-based traffic category, with a relative priority of 6.
Flows marked with AF12 are expected to be of the same nature as flows marked with AF11, but with a lower criticality. Therefore, it is RECOMMENDED to map AF12 to QCI 8, thereby admitting AF12 traffic into the non-GBR category Video and TCP-based traffic, with a relative priority of 8.

Flows marked with AF13 are expected to be of the same nature as flows marked with AF11 and AF12, but with the lowest criticality. As such, it is RECOMMENDED to map AF13 to QCI 9, thereby admitting AF13 traffic into the non-GBR category Video and TCP-based traffic, with a relative priority of 9.

It should be noted that a consequence of such classification is that AF11 is mapped to the same QCI as CS3 and AF22, AF12 is mapped to the same QCI as AF33 and AF23, and AF13 is mapped to the same QCI as CS2. However, this overlap is unavoidable, as some QCIs express intents that are expressed in the Diffserv domain through distinct marking values, grouped in the 3GPP domain under the same general category.

4.2.9. Standard

The Standard service class is recommended for traffic that has not been classified into one of the other supported forwarding service classes in the Diffserv network domain. This service class provides the Internet’s "best-effort" forwarding behavior. [RFC4594] Section 4.9 states that the "Standard service class MUST use the Default Forwarding (DF) PHB".

The Standard service class loosely corresponds to the default non-GBR bearer practice in 3GPP. Therefore, it is RECOMMENDED to map Standard service class traffic marked DF DSCP to QCI 9, thereby admitting it to the low priority Video and TCP-based traffic category, with a relative priority of 9.

4.2.10. Low-Priority Data

The Low-Priority Data service class serves applications that the user is willing to accept without service assurances. This service class is specified in [RFC3662] and [LE-PHB]. [RFC3662] and [RFC4594] both recommend Low-Priority Data be marked CS1 DSCP.

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.

The Low-Priority Data service class does not have equivalent in the 3GPP domain, where all service is controlled and allocated differentially. As such, there is no clear QCI that could be
labelled low priority below the best effort category. As such, it is RECOMMENDED to map Low-Priority Data traffic marked CS1 DSCP to QCI 9, thereby admitting it to the low priority Video and TCP-based traffic category, with a relative priority of 9.

4.3. Summary of Recommendations for DSCP-to-QCI Mapping

The table below summarizes the [RFC4594] DSCP marking recommendations mapped to 3GPP:
<table>
<thead>
<tr>
<th>DSCP</th>
<th>Recommended QCI</th>
<th>Resource Type</th>
<th>Priority Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS7</td>
<td>82</td>
<td>GBR</td>
<td>1.9</td>
</tr>
<tr>
<td>CS6</td>
<td>82</td>
<td>GBR</td>
<td>1.9</td>
</tr>
<tr>
<td>EF</td>
<td>1</td>
<td>GBR</td>
<td>2</td>
</tr>
<tr>
<td>CS5</td>
<td>4</td>
<td>GBR</td>
<td>5</td>
</tr>
<tr>
<td>AF43</td>
<td>7</td>
<td>non-GBR</td>
<td>7</td>
</tr>
<tr>
<td>AF42</td>
<td>4</td>
<td>GBR</td>
<td>5</td>
</tr>
<tr>
<td>AF41</td>
<td>2</td>
<td>GBR</td>
<td>4</td>
</tr>
<tr>
<td>CS4</td>
<td>80 3</td>
<td>non-BGR GBR</td>
<td>6.8 3</td>
</tr>
<tr>
<td>AF33</td>
<td>8</td>
<td>non-GBR</td>
<td>8</td>
</tr>
<tr>
<td>AF32</td>
<td>6</td>
<td>non-GBR</td>
<td>6</td>
</tr>
<tr>
<td>AF31</td>
<td>4</td>
<td>GBR</td>
<td>5</td>
</tr>
<tr>
<td>CS3</td>
<td>85</td>
<td>GBR</td>
<td>2.1</td>
</tr>
<tr>
<td>AF23</td>
<td>8</td>
<td>Non-GBR</td>
<td>8</td>
</tr>
<tr>
<td>AF22</td>
<td>6</td>
<td>Non-GBR</td>
<td>6</td>
</tr>
<tr>
<td>AF21</td>
<td>70</td>
<td>Non-GBR</td>
<td>5.5</td>
</tr>
<tr>
<td>CS2</td>
<td>9</td>
<td>Non-GBR</td>
<td>9</td>
</tr>
<tr>
<td>AF13</td>
<td>9</td>
<td>Non-GBR</td>
<td>9</td>
</tr>
<tr>
<td>AF12</td>
<td>8</td>
<td>Non-GBR</td>
<td>8</td>
</tr>
<tr>
<td>AF11</td>
<td>6</td>
<td>Non-GBR</td>
<td>6</td>
</tr>
<tr>
<td>CS0</td>
<td>9</td>
<td>Non-GBR</td>
<td>9</td>
</tr>
<tr>
<td>CS1</td>
<td>9</td>
<td>Non-GBR</td>
<td>6.8</td>
</tr>
</tbody>
</table>
5. QCI-to-DSCP Mapping Recommendations

Traffic travelling from the 3GPP domain toward the Internet or the enterprise domain may already display DSCP marking, if the UE is capable of marking DSCP along with, or without, upstream QCI marking, as detailed in Section 2.1.

When Diffserv marking is present in the flows originating from the UE and transiting through the CN (Core Network), and if Diffserv marking are not altered or removed on the path toward the Diffserv domain, then the network can be considered as end-to-end Diffserv compliant. In this case, it is RECOMMENDED that the entity providing the translation from QCI to Diffserv ignores the QCI value and simply forwards unchanged the Diffserv values expressed by the UE in its various flows.

This general recommendation is not expected to fit every last deployment model, and as such Diffserv marking MAY be overridden by network administrators, as needed, before the flows are forwarded to the Internet, the enterprise network or the Diffserv domain in general. Additionally, within a given Diffserv domain, it is generally NOT RECOMMENDED to pass through DSCP markings from unauthenticated, unidentified or unauthorized devices, as these are typically considered untrusted sources, as detailed in Section 7. Such risk is limited within the 3GPP domain where no upstream traffic is admitted without prior authentication of the UE. However, this risk exists when UE traffic is forwarded to an enterprise domain to which the UE does not belong.

In cases where the UE is unable to apply Diffserv marking, or if these markings are modified or removed within the 3GPP domain, such that these markings may not represent the intent expressed by the UE, and in cases where the QCI is available to represent the flow intent, the recommendations in this section apply.

5.1. QCI and Diffserv Logic Reconciliation

The QCIs are defined as relative priorities for traffic flows which are described by combinations of 6 or more parameters, as expressed in Section 2.2. As such, QCIs also represent flows in terms of multi-dimensional needs, not just in terms of relative priorities. This multi-dimensional logic is different from the Diffserv logic, where each traffic class is represented as a combination of needs relative to delay, jitter and loss. This characterization around three parameters allows for the construction of a fairly hierarchical traffic categorization infrastructure, where traffic with high sensitivity to delay and jitter also typically has high sensitivity to loss.
By contrast, the 3GPP QCI structure presents multiple points where dimensions cross one another with different or opposing vectors. For example, IMS signaling (QCI 5) is defined with very high priority (1), low loss tolerance (10^-6), but is non-GBR and belongs to the signaling category. By contrast, Conversational voice (QCI 1) has lower priority (2) than IMS signaling, higher loss tolerance (10^-2), yet benefits from a GBR. Fitting both QCIs 5 and 1 in a hierarchical model is challenging.

At the same time, QCIs represents needs that can apply to different applications of various criticality but sending flows of the same nature. For example, QCIs 6, 8 and 9 all include voice traffic, video traffic, but also email or FTP. What distinguish these QCIs is the criticality of the associated traffic. Diffserv does not envisions voice and FTP as possibly belonging to the same class. As the same time, QCI 2 and QCI 9 include real-time voice traffic. Diffserv does not allow a type of traffic with stated sensitivity to loss, delay and jitter to be split into categories at both end of the priority spectrum.

As such, it is not expected that QCIs can be mapped to the Diffserv model strictly and hierarchically. Instead, a better approach is to observe the various QCI categories, and analyze their intent. This process allows for the grouping of several QCIs into hierarchical groups, that can then be translated into ensembles coherent with the Diffserv logic. This approach, in turn, allows for incorporation of new QCIs as the 3GPP model continues to evolve.

It should be noted, however, that such approach results in partial incompatibility. Some QCIs represent an intent that is simply not present in the Diffserv model. In that case, attempting to artificially stitch the QCI to an existing Diffserv traffic class and marking would be dangerous. QCI traffic forwarded to the Diffserv domain would be mixed with Diffserv traffic that would represent a very different intent.

As such, the result of this classification is that some QCIs call for new Diffserv traffic classes and markings. This consequence is preferable to mixing traffic of different natures into the same pre-existing category.

Each QCI is represented with 6 parameters, including an Example Services value. This parameter is representative of the QCI intent. Although [TS 23.203] summarizes each QCI intent, this standard is only a summary of more complex classification expressed in other 3GPP standards. It is often necessary to refer to these other standards to obtain a more complex description of each QCI and the multiple type of flows that each QCI represents.
For the purpose of this document, the QCI intent is the primary classification driver, along with the priority level. The secondary elements, such as priority, delay budget and loss tolerance allow for better refinement of the relative classifications of the QCIs. The resource types (GBR, non-GBR) provide additional visibility into the intent.

Although 26 QCIs are listed in [TS 23.203], representing two bearer types (GBR, non-GBR), 21 priority values, 9 delay budget values, and 7 loss tolerance values, examining the intent surfaces 9 traffic families:

1. Voice QCI [1] (dialer / conversational voice) is its own group
2. Voice signaling [5] (IMS) is its own group
3. Voice related (other voice applications, including PTT) [65, 66, 69]
4. Video (conversational or not, mission critical or not) [67, 2, 4, 71, 72, 73, 74, 76]
5. Live streaming / interactive gaming is its own group [7]
6. Low latency eMBB, AR/VR is its own group [80]
7. V2X messaging [75, 3, 9]
8. Automation and Transport [82, 83, 84, 85]
9. Non-mission-critical data [6, 8, 9]
10. Mission-critical data is its own group [70]

5.2. Voice QCI [1]

Several QCIs are intended to carry voice traffic. However, QCI 1 stands apart from the others. Its category is Conversational Voice, but this QCI is intended to represent the VoLTE voice bearer, for dialer and emergency services. QCI 1 uses a GBR, and has a priority level of 2. Its packet delay budget is 100 ms (from UE to P-GW) with a packet error loss of at most 10.E-2. As the GBR is allocated by the infrastructure, QCI 1 is both admitted and allocated dedicated resources. As such, QCI 1 maps in intent and function to [RFC5865], Admitted Voice, and is RECOMMENDED for mapping to DSCP 44.
5.3.  IMS Signaling [5]

QCI 5 is intended for Signaling. This category does not represent signaling for VoLTE, as such signaling is not conducted over the UE data channels. Instead, QCI 5 is intended for IMS services. IP Multimedia System (IMS) is a framework for delivering multimedia services over IP networks. These services include real-time and video applications, and their signaling is recommended to be carried, whenever possible, using IETF protocols such as SIP. Being of signaling nature, QCI 5 is non-GBR. However, being critical to enabling IMS real-time applications, QCI has a high priority of 1. Its packet delay budget is 100 ms, but packet error loss rate very low, at less than 10.E-6. Overall, QCI 5 maps rather well to the intent of [RFC4594] signaling for real time applications, and as such is RECOMMENDED to map to [RFC4594] Signaling, CS5.

5.4.  Voice-related QCIs [65, 66, 69]

Several QCIs display the commonality of targeting voice (non-VoLTE) traffic:

- QCI 65 is GBR, mission critical PTT voice, priority 0.7
- QCI 66 is GBR, non-mission critical PTT voice, priority 2
- QCI 69 is non-GBR, mission-critical PTT signaling, priority 0.5

These QCI are Voice in nature, and naturally fit into a proximity marking model with DSCP 46 and 44.

Additionally, lower priority marks higher precedence intent in QCI. However, there is no model in [RFC4594] that distinguishes 3 classes of voice traffic. Therefore, new markings are unavoidable. As such, there is a need to group these markings in the Voice category (101 xxx), and to order 69, 65 and 66 with different markings to reflect their different priority levels.

Among these three QCIs, 69 is non-GBR, intended for mission-critical PTT signaling, with the highest priority of the three, at 0.5. QCI 69 is intended for signaling, but is latency sensitive, with a low 60 ms delay budget and a low 10.E-6 loss tolerance. Being of Signaling nature for real time applications, QCI 69 has proximity of intent with CS5 (Voice signaling, 40), but this marking is already used by QCI 5. Therefore, it is RECOMMENDED to map QCI 69 to a new DSCP marking, 41.

Similarly, QCI 66 is GBR and targeted for non-mission critical PTT voice, with a priority level of 2. QCI 66 is Voice in nature, and
GBR. However, QCI 66 is intended for non-mission-critical traffic, and has a lower priority than mission-critical Voice, a higher tolerance for delay (100 ms vs 75). As such, QCI 66 cannot fit within [RFC4594] model mapping real-time voice to the class EF (DSCP 46). Here again, a new marking is needed. As such, this QCI fits in intent and proximity closest to Admitted Voice, but is non-GBR, and therefore non-admitted, guiding a new suggested marking of 43.

Then, QCI 65 is GBR, intended for mission critical PTT voice, with a relative low priority index of 0.7. QCI 65 receives GBR and is intended for mission critical traffic. Its priority is higher (0.7 vs 2) than QCI 66, but a lower priority (0.7 vs 0.5) than QCI 69. Additionally, QCI 65 cannot be represented by DSCP 44 (used by QCI 1), or DSCP 46 (use by non-GBR voice). As such, QCI 65 fits between QCI 69 and QCI 66, with a new suggested marking of 42.

5.5. Video QCIs [67, 2, 4, 71, 72, 73, 74, 76]

Although six different QCIs have example services that include some form of video traffic, eight QCIs are video in nature, QCIs 67, 2, 4, 71, 72, 73, 74, and 76.

All eight QCIs represent video streams and fit naturally in the AF4x category. However, these QCIs do not match [RFC4594] intent for multimedia conferencing, in that they are all admitted (being associated to a GBR). They also do not match the category described by [RFC5865] for capacity-admitted traffic. Therefore, there is not a clear possible mapping for any of these QCIs to an existing AF4x category. In order to avoid mixing admitted and non-admitted video in the same class, it is necessary to associate these QCIs to new Diffserv classes.

In particular, QCI 67 is GBR, intended for mission-critical video user plane. This QCI is video in nature, and matches traffic that is rate-adaptive, and real time. QCI 67 priority is high (1.5), with a tolerant delay budget (100ms) and rather low loss tolerance (10.E-3). QCI 67 is GBR.

As such, its RECOMMENDED to map QCI 67 against the DSCP value closest to AF4x video with lowest discard eligibility (AF41), namely category 33.

Similarly, QCI 2 is intended for conversational video (live streaming). This QCI 2 is also video in nature and associated to a GBR, however its priority is lower than QCI 67 (4 vs 1.5). Additionally, its delay budget is also larger (150 ms vs 100 ms). Its packet error loss is also 10.E-3. As such, QCI 2 fits well within a video queue, with a larger drop probability than QCI 67.
Therefore, it is RECOMMENDED to map QCI 2 to the video category with a Diffserv marking of 35.

QCI 4 is intended for non-conversational video (buffered streaming), with a priority of 5. This QCI is also video in nature. Although it is buffered, it is admitted, being associated to a GBR. QCI 4 as a lower priority than QCI 67 and QCI 2, and a larger delay budget (300 ms vs 150/100). However, its packet loss tolerance is low (10.E-6). This combination makes it eligible for a video category, but with a higher drop probability than QCI 67 and 2. Therefore, it is RECOMMENDED to map QCI 4 to DSCP 37.

QCIs 71, 72, 73, 74 and 76 are intended for “Live” Uplink Streaming (LUS) services, where an end-user with a radio connection (for example a reporter or a drone) streams live video feed into the network or to a second party ([TS 26.939]). This traffic is GBR. However, [TS 26.239] defines LUS and also differentiates GBR from MBR and TBR. At the time of the admission, the infrastructure can offer a Guaranteed Bit Rate, which should match the bare minimum rate expected by the application (and its codec). Because of the burstiness nature of video, the Maximum Bit Rate (MBR) available to the transmission should be much higher than the GBR. In fact, the Target Bit Rate (TBR), which is the preferred service operation point for that application, is likely close to the MBR. Thus, the application will receive a treatment between the GBR and the TBR. This allocated bit rate will directly translate in video quality changes, where an available bit rate close to the GBR will result in a lower Mean Opinion Score than a bit rate close to the TBR. As the application detects the constraints on the available bit rate, it may adapt by changing its codec and compression scheme accordingly. Flows with higher compression will have higher delay tolerance and budget (as a single packet burst represents a larger segment of the video flow) but lower loss tolerance (as each lost packet represents a larger segment of the video flow).

5.6. Live streaming and interactive gaming [7]

QCI 7 is non-GBR and intended for live streaming voice or video interactive gaming. Its priority is 7. It is the only QCI targeting this particular traffic mix. In the Diffserv model, voice and video are different categories, and are also different from interactive gaming (real time interactive). In the 3GPP model, live streaming video and mission-critical video are defined in other queues with high priority (e.g. QCI 2 for video Live streaming, with a priority of 2, or QCI 67 for mission-critical video, with a priority of 1.5). By comparison, QCI 7 priority is relatively low (7), with a 100 ms budget delay and a comparatively rather high loss tolerance (10.E-3).
As such, QCI 7 first well with bursty (e.g. video) and possibly rate adaptive flows, with possible drop probability. It is also non-admitted (non-GBR), and as such, fits close to [RFC4594] intent for multimedia conferencing, with high discard eligibility. Therefore, it is RECOMMENDED to map QCI 7 to the existing Diffserv category AF43.

5.7. Low latency eMBB and AR/VR [80]

QCI 80 is intended for low latency eMBB (enhanced Mobile Broadband) applications, such as Augmented Reality of Virtual Reality (AR/VR). This QCI priority is 6.8, with a low packet delay budget of 10 ms, and a packet error loss rate of at most 10.E-6.

QCI 80 is non-GBR, yet intended for real time applications. Traffic in the AR/VR category typically does not react dynamically to losses, requires bandwidth and a low and predictable delay.

As such, QCI 80 matches closely the specifications for CS4. Therefore, it is RECOMMENDED to map QCI 80 to the existing category CS4.

5.8. V2X messaging [75, 3, 9]

Three QCIs are intended specifically to carry Vehicle to Anything (V2X) traffic, QCIs 75, 3, and 79. All 3 QCIs are data in nature, and fit naturally into the AF2x category. However, two of these QCIs (75 and 3) are admitted (GBR), and therefore do not fit in the current Diffserv model. QCI 79 is non-admitted, but matches none of the AF2X categories in [RFC4594].

In particular, QCI 75 is GBR, with a rather high priority (2.5), a low delay budget (50 ms), but tolerance to losses (10E-2). Being low latency data in nature, QCI 75 fits well in the AF2X category. However, being admitted, it fits none of the existing markings. Being the highest traffic (in priority) in this low latency data family, QCI 75 is recommended to be mapped to a new category, as close as possible to the AF2X class, and with a low drop probability. As such, it is RECOMMENDED to map QCI 75 to DSCP 17.

Similarly, QCI 3 is intended for V2X messages, but can also be used for Real time gaming, or Utility traffic (medium voltage distribution) or process automation monitoring. QCI 3 priority is 3. QCI 3 is data in nature, but GBR. Its delay budget is low (50 ms), but with some tolerance to loss (10E-3).

QCI 3 is of the same type as QCI 75, but with a lower priority. Therefore, QCI 3 should be mapped to a category close to the category
to which 75 is mapped, but with a higher drop probability. As such, it is RECOMMENDED to map QCI 3 to DSCP 19.

Additionally, QCI 79 is also intended for V2X messages. QCI 79 similar in nature to QCIs 75 and 3, but is non-critical (non-GBR). Its priority is also lower (6.5). Its budget delay is similar to that of QCIs 75 and 3 (50 ms), and its packet error loss rate is similar to that of QCI 75 (10.E-2).

QCI 79 partially matches AF2X, but is not elastic, and therefore cannot fit exactly in [RFC4594] model. As such, it is recommended to a mapping similar to QCI 75 and 3, with a higher drop probability. Therefore, it is RECOMMENDED to map QCI 79 to DSCP 21.

5.9. Automation and Transport [82, 83, 84, 85]

QCI 84 is intended for intelligent transport systems. As such, its intent is close to the V2X messaging category. QCI 84 is also admitted (GBR). However, QCI 84 is intended for traffic with a smaller packet delay budget (30 ms vs 50 ms for QCI 75) and a smaller packet error loss maximum rate (10.E-6 vs 10.E-2 for QCI 75). As such, QCI 84 should be mapped against a category above that of QCIs 75 or QCI 3. Being admitted, QCI 84 does not map easily into an existing category. As such, it is RECOMMENDED to map QCI 84 to category 31.

QCI 85 is intended for electricity distribution (high voltage) communication. As such, it is close in intent to QCI 3. QCI 85 is also GBR. However, QCI 85 priority is lower than that of QCI 3 (2.1 vs 3). QCI 85 has also a very low packet delay budget (5 ms vs 50 ms for QCI 3) and low packet error loss rate (10.E-6 vs 10.E-3 for QCI 3). As such, QCI 84 should be mapped to a category higher than that of QCI 3, with a very low drop probability. As such, it is RECOMMENDED to map QCI 85 to category 25.

QCIs 82 and 83 are both intended for discrete automation control traffic. QCI 82 represents traffic with a higher priority (1.9) than traffic matched to QCI 83 (priority 2.2). QCI 82 also expects smaller data bursts (255 bytes) than QCI 83 (1358 bytes). However, both QCIs are admitted (GBR), with the same low packet delay budget (10 ms) and packet error loss maximum rate (10.E-4).

As such, QCIs 82 and 83 fit in the same general category, with a higher drop probability assigned to QCI 83. They also fit the general intent category of automation traffic types, with a priority higher than that of other M2M traffic types (e.g. V2X messages). As such, they fit well into the AF3X category. However, being both admitted (GBR), they do not easily map to any existing AF3X category, and require new categories.
As such, it is RECOMMENDED to map QCI 82 to category 27. Similarly, it is RECOMMENDED to map QCI 83 to category 29.

5.10. Non-mission-critical data [6,8,9]

QCIs 6, 8 and 9 are intended for non-GBR, Video or TCP data traffic. All 3 QCIs are data in nature, non-mission critical, relative low priority and therefore fit naturally into the AF1x category. The inclusion in these QCIs’ intent of buffered video is an imperfect fit for AF1X. However, the intent of these QCIs is to match buffered, and non-mission critical traffic. As such, they match the intent of AF1X, even if the Diffserv model would not associate buffered video to non-mission critical, buffered and low priority traffic.

The intent of all three QCIs is similar. The difference lies in their priority and criticality.

QCI 6 has priority 6, a packet delay budget of 300 ms, and a packet error loss rate of at most 10.E-6. QCI 8 has a priority 8, a packet delay budget of 300 ms, and a packet error loss rate of at most 10.E-6. QCI 9 has priority 9, and also a packet delay budget of 300 ms and a packet error loss rate of at most 10.E-6. As these three QCIs represent the same intent and are only different in their priority level, using discard eligibility to differentiate them is logical. As such, it is RECOMMENDED to map QCI 6 to category AF11. Similarly, it is RECOMMENDED to map QCI 8 to AF12. And logically, it is RECOMMENDED to map QCI 9 to AF13.

5.11. Mission-critical data [70]

QCI 70 is non-GBR, intended for mission critical data, with a priority of 5.5, a packet delay budget of 200 ms and a packet error loss rate tolerance of at most 10.E-6. The traffic types intended for QCI 70 are the same as for QCIs 6,8,9 categories, namely buffered streaming video and TCP-based traffic, such as www, email, chat, FTP, P2P and other file sharing applications. However, QCI 70 is specifically intended for applications that are mission critical. For this reason, QCI 70 priority is higher than QCIs 6, 8 or 9 priorities (5.5 vs 6, 8 and 9 respectively). Therefore, QCI 70 fits well in the AF2x family, while 6,8,9 are in AF1x. As QCI 70 displays intermediate differentiated treatment, if also fits well with an intermediate discard eligibility. As such, it is RECOMMENDED to map QCI 70 to DSCP 20 (AF22).
### 5.12. Summary of Recommendations for QCI-to-DSCP Mapping

The table below summarizes the 3GPP QCI to [RFC4594] DSCP marking recommendations:

<table>
<thead>
<tr>
<th>QCI</th>
<th>Resource Type</th>
<th>Priority Level</th>
<th>Example Services</th>
<th>Recommended DSCP (PHB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GBR</td>
<td>2</td>
<td>Conversational Voice</td>
<td>44 (VA)</td>
</tr>
<tr>
<td>2</td>
<td>GBR</td>
<td>4</td>
<td>Conversational Video (Live Streaming)</td>
<td>35 (N.A.)</td>
</tr>
<tr>
<td>3</td>
<td>GBR</td>
<td>3</td>
<td>Real Time Gaming, V2X messages, Electricity distribution (medium voltage) Process automation (monitoring)</td>
<td>19 (N.A.)</td>
</tr>
<tr>
<td>4</td>
<td>GBR</td>
<td>5</td>
<td>Non-Conversational Video (Buffered Streaming)</td>
<td>37 (N.A.)</td>
</tr>
<tr>
<td>65</td>
<td>GBR</td>
<td>0.7</td>
<td>Mission Critical user plane Push To Talk voice (e.g., MCPTT)</td>
<td>42 (N.A.)</td>
</tr>
<tr>
<td>66</td>
<td>GBR</td>
<td>2</td>
<td>Non-Mission-Critical user plane Push To Talk voice</td>
<td>43 (N.A.)</td>
</tr>
<tr>
<td>67</td>
<td>GBR</td>
<td>1.5</td>
<td>Mission Critical Video user plane</td>
<td>33 (N.A.)</td>
</tr>
<tr>
<td>75</td>
<td>GBR</td>
<td>2.5</td>
<td>V2X messages</td>
<td>17 (N.A.)</td>
</tr>
<tr>
<td>82</td>
<td>GBR</td>
<td>1.9</td>
<td>Discrete automation (small packets)</td>
<td>27 (N.A.)</td>
</tr>
<tr>
<td>83</td>
<td>GBR</td>
<td>2.2</td>
<td>Discrete automation (large packets)</td>
<td>29 (N.A.)</td>
</tr>
<tr>
<td>84</td>
<td>GBR</td>
<td>2.4</td>
<td>Intelligent Transport Systems</td>
<td>31 (N.A.)</td>
</tr>
<tr>
<td>85</td>
<td>GBR</td>
<td>2.1</td>
<td>Electricity Distribution - High</td>
<td>25 (N.A.)</td>
</tr>
<tr>
<td>5</td>
<td>Non-GBR</td>
<td>1</td>
<td>Voltage</td>
<td>40 (CS5)</td>
</tr>
<tr>
<td>6</td>
<td>Non-GBR</td>
<td>6</td>
<td>Video (Buffered Streaming) TCP-based (e.g. www, email, chat, ftp, p2p file sharing, progressive video)</td>
<td>10 (AF11)</td>
</tr>
<tr>
<td>7</td>
<td>Non-GBR</td>
<td>7</td>
<td>Voice, Video (live streaming), interactive gaming</td>
<td>38 (AF43)</td>
</tr>
<tr>
<td>8</td>
<td>Non-GBR</td>
<td>8</td>
<td>Video (buffered streaming) TCP-based (e.g. www, email, chat, ftp, p2p file sharing, progressive video)</td>
<td>12 (AF12)</td>
</tr>
<tr>
<td>9</td>
<td>Non-GBR</td>
<td>9</td>
<td>Same as 8</td>
<td>14 (AF13)</td>
</tr>
<tr>
<td>69</td>
<td>Non-GBR</td>
<td>0.5</td>
<td>Mission Critical delay sensitive signalling (e.g., MC-PTT signalling, MC Video signalling)</td>
<td>41 (N.A.)</td>
</tr>
<tr>
<td>70</td>
<td>Non-GBR</td>
<td>5.5</td>
<td>Mission Critical Data (e.g. example services are the same as QCI 6/8/9)</td>
<td>20 (AF22)</td>
</tr>
<tr>
<td>79</td>
<td>Non-GBR</td>
<td>6.5</td>
<td>V2X messages</td>
<td>21 (N.A.)</td>
</tr>
<tr>
<td>80</td>
<td>Non-GBR</td>
<td>6.8</td>
<td>Low latency eMBB applications (TCP/UDP-based); augmented reality</td>
<td>32 (CS4)</td>
</tr>
</tbody>
</table>

6. IANA Considerations

This document allocates thirteen codepoints (17, 19, 21, 25, 27, 29, 31, 33, 35, 37, 41, 42 and 43), further detailed in Section 5, in Pool 1 of the code space defined by [RFC2474].
7. Specific Security Considerations

The recommendations in this document concern widely deployed wired and wireless network functionality, and, for that reason, do not present additional security concerns that do not already exist in these networks.

8. Security Recommendations for General QoS

It may be possible for a wired or wireless device (which could be either a host or a network device) to mark packets (or map packet markings) in a manner that interferes with or degrades existing QoS policies. Such marking or mapping may be done intentionally or unintentionally by developers and/or users and/or administrators of such devices.

To illustrate: A gaming application designed to run on a smartphone may request that all its packets be marked DSCP EF. Although the 3GPP infrastructure may only allocate a non-GBR default QCI (e.g. QCI 9) for this traffic, the translation point into the Internet domain may consider the DSCP marking instead of the allocated QCI, and forward this traffic with a marking of EF. This traffic may then interfere with QoS policies intended to provide priority services for business voice applications.

To mitigate such scenarios, it is RECOMMENDED to implement general QoS security measures, including:

- Setting a traffic conditioning policy reflective of business objectives and policy, such that traffic from authorized users and/or applications and/or endpoints will be accepted by the network; otherwise, packet markings will be "bleached" (i.e., re-marked to DSCP DF). Additionally, Section 5 made it clear that it is generally NOT RECOMMENDED to pass through DSCP markings from unauthorized, unidentified and/or unauthenticated devices, as these are typically considered untrusted sources. This is especially relevant for Internet of Things (IoT) deployments, where tens of billions of devices with little or no security capabilities are being connected to LTE and IP networks, leaving them vulnerable to be utilized as agents for DDoS attacks. These attacks can be amplified with preferential QoS treatments, should the packet markings of such devices be trusted.

- Policing EF marked packet flows, as detailed in [RFC2474] Section 7 and [RFC3246] Section 3.

Finally, it should be noted that the recommendations put forward in this document are not intended to address all attack vectors.
leverage QoS marking abuse. Mechanisms that may further help mitigate security risks of both wired and wireless networks deploying QoS include strong device- and/or user-authentication, access-control, rate-limiting, control-plane policing, encryption, and other techniques; however, the implementation recommendations for such mechanisms are beyond the scope of this document to address in detail. Suffice it to say that the security of the devices and networks implementing QoS, including QoS mapping between wired and wireless networks, merits consideration in actual deployments.

9. References

9.1. Normative References


9.2. Informative References


Authors’ Addresses

Jerome Henry
Cisco

Email: jerhenryl@cisco.com
Tim Szigeti
Cisco

Email: szigeti@cisco.com
DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput
(L4S)
draft-ietf-tsvwg-aqm-dualq-coupled-10

Abstract

The Low Latency Low Loss Scalable Throughput (L4S) architecture allows data flows over the public Internet to achieve consistent ultra-low queuing latency, generally zero congestion loss and scaling of per-flow throughput without the scaling problems of traditional TCP. To achieve this, L4S data flows have to use one of the family of ‘Scalable’ congestion controls (Data Centre TCP and TCP Prague are examples) and a form of Explicit Congestion Notification (ECN) with modified behaviour. However, until now, Scalable congestion controls did not co-exist with existing TCP Reno/Cubic traffic --- Scalable controls are so aggressive that ‘Classic’ TCP algorithms 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 specification defines ‘DualQ Coupled Active Queue Management (AQM)’, which enables these Scalable congestion controls to safely co-exist with Classic Internet traffic.

Analytical study and implementation testing of the Coupled AQM have shown that Scalable and Classic flows competing under similar conditions run at roughly the same rate. It achieves this indirectly, without having to inspect transport layer flow identifiers. When tested in a residential broadband setting, DCTCP also achieves sub-millisecond average queuing delay and zero congestion loss under a wide range of mixes of DCTCP and ‘Classic’ broadband Internet traffic, without compromising the performance of the Classic traffic. The solution also reduces network 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 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
   1.1. Outline of the Problem ................................. 3
   1.2. Scope .................................................. 5
   1.3. Terminology ............................................ 7
   1.4. Features ................................................. 8
2. DualQ Coupled AQM ............................................ 9
   2.1. Coupled AQM ............................................. 9
   2.2. Dual Queue ............................................... 10
   2.3. Traffic Classification ................................. 11
   2.4. Overall DualQ Coupled AQM Structure .................... 11
   2.5. Normative Requirements for a DualQ Coupled AQM ....... 14
       2.5.1. Functional Requirements ......................... 14
          2.5.1.1. Requirements in Unexpected Cases ............ 15
          2.5.2. Management Requirements ....................... 16
          2.5.2.1. Configuration .................................. 16
          2.5.2.2. Monitoring .................................... 18
          2.5.2.3. Anomaly Detection ............................. 18
          2.5.2.4. Deployment, Coexistence and Scaling .......... 19
   3. IANA Considerations ....................................... 19

4. Security Considerations ........................................ 19
4.1. Overload Handling ............................................. 19
  4.1.1. Avoiding Classic Starvation: Sacrifice L4S Throughput
         or Delay? .................................................. 20
  4.1.2. Congestion Signal Saturation: Introduce L4S Drop or
         Delay? ....................................................... 21
  4.1.3. Protecting against Unresponsive ECN-Capable Traffic . 22
5. Acknowledgements ............................................... 22
6. Contributors ..................................................... 23
7. References ....................................................... 23
  7.1. Normative References ....................................... 23
  7.2. Informative References .................................... 24
Appendix A. Example DualQ Coupled PI2 Algorithm ................ 27
  A.1. Pass #1: Core Concepts .................................... 28
  A.2. Pass #2: Overload Details ................................. 36
Appendix B. Example DualQ Coupled Curvy RED Algorithm ........... 40
  B.1. Curvy RED in Pseudocode .................................. 40
  B.2. Efficient Implementation of Curvy RED ..................... 46
Appendix C. Guidance on Controlling Throughput Equivalence ....... 48
Authors’ Addresses ................................................... 49

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 ultra-low queuing latency 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 ultra-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 (sawtoothing) 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 TCP-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. BBRv1 [BBRv1]) seems to have hit a similar lower limit to queuing delay of about 20ms on average (and any additional BBRv1 flow adds another 20ms of queuing) 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 TCP-Friendly congestion controls;

2. A shift of smoothing and hence smoothing delay from network to sender.

Without the former, a ‘Classic’ 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). 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 need 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 fine.

However, until now, Scalable congestion controls (like DCTCP) did not co-exist with existing ECN-capable TCP Reno [RFC5681] or Cubic [RFC8312] traffic --- Scalable controls are so aggressive that these ‘Classic’ TCP algorithms 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, L4S addresses the root cause of the latency problem --- it is an enabler for the smooth low latency scalable behaviour of Scalable congestion controls, so that every packet in every flow can enjoy very low latency, then there is 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);

End-system: A Scalable congestion control (defined in Section 2.1.
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 that has been deployed for some time in Linux, Windows and FreeBSD operating systems and Relentless TCP [Mathis09] is another example. During the progress of this document through the IETF a number of other Scalable congestion controls were implemented, e.g. TCP Prague [PragueLinux], QUIC Prague and the L4S variant of SCREAM for real-time media [RFC8298]. (Note: after the v3.19 Linux kernel, bugs were introduced into DCTCP’s scalable behaviour and not all the patches applied for L4S evaluation had been applied to the mainline Linux kernel, which was at v5.2 at the time of writing).

The focus of this specification is to get the network part of the L4S service in place. 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 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. Here, the term ‘site’ is used loosely 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 [PI2] and [DCttH15] give the full rationale for the AQM’s design, both discursively and in more precise mathematical form.

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 (denoted by subscript C): The ‘Classic’ service is intended for all the behaviours that currently co-exist with TCP Reno (TCP Cubic, Compound, SCTP, etc).

Low-Latency, Low-Loss and Scalable (L4S, denoted by subscript L): The ‘L4S’ service is intended for a set of congestion controls with scalable properties, such as TCP Prague and DCTCP. For the public Internet an L4S transport has to comply with the requirements in Section 4 of [I-D.ietf-tsvwg-ecn-l4s-id] (aka. the ‘Prague L4S requirements’).

Either service can cope with a proportion of unresponsive or less-responsive traffic as well, as long (e.g. DNS, VoIP, game sync datagrams, etc), just as a single queue AQM can if this traffic makes minimal contribution to queuing. The DualQ Coupled AQM behaviour below is defined to be similar to a single FIFO queue with respect to unresponsive and overload traffic.
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 stunningly 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 [PI2] [DCttH15].

Subjective testing was also 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 watching 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 touchpad 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 ultra-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, and the latency of
Classic traffic does not suffer 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
- while Scalable flows do not need a queue to keep utilization high, but they cannot keep latency predictably low if they are mixed with legacy TCP flows,

The L4S queue has latency priority, but 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 TCP's congestion window, cwnd, and its drop probability, p. To a first order approximation, cwnd of TCP 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 700Mb/s. In such cases it can be assumed that Cubic traffic behaves similarly to Reno (but with a slightly different constant of proportionality). The term 'Classic' will be used for the collection of Reno-friendly traffic including Cubic in Reno mode.

The 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 also 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, $p_L$. [RFC8311] updates the original ECN specification [RFC3168] to allow these probabilities to be distinct, because RFC 3168 required them to be the same.

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, and allows 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 Classic ‘TCP formula’.

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:

$$ p_C = \left( \frac{p_{CL}}{k} \right)^2 $$

(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.

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 L and 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 sender to use the ECT(1) and CE codepoints of the ECN field as this identifier, having assessed various alternatives. 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, [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 (see [I-D.ietf-tsvwg-ecn-l4s-id]), specific Diffserv codepoints such as EF ( Expedited Forwarding) and Voice-Admit service classes (see [I-D.briscoe-tsvwg-l4s-diffserv]) or certain protocols (e.g. ARP, DNS). Note that the mechanism only reads these identifiers. [I-D.ietf-tsvwg-ecn-l4s-id] says it "MUST NOT alter these non-ECN identifiers".

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.

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 \cite{PI2} that it is preferable to control load with a linear controller, then square the output before applying it as a drop probability to TCP (because TCP 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

$$p_C = (p')^2. \tag{2}$$

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:

$$p_{CL} = k*p', \tag{3}$$

where $k$ is the constant coupling factor (see Appendix C).

It can be seen that these two transformations of $p'$ implement the required coupling given in equation (1) earlier.

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 strict priority over the C queue, whenever the L queue grows, the AQM should apply marking probability $p'_L$, but $p_L$ should not fall below $p_{CL}$. This suggests:

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

which has also been found to work very well in practice.
Legend: ===> traffic flow; ---> control dependency.

Figure 1: DualQ Coupled AQM Schematic

After the AQMs have applied their dropping or marking, the scheduler forwards their packets to the link, giving priority to L4S traffic. Priority has to be conditional in some way (see Section 4.1). Simple strict priority is inappropriate otherwise it could lead the L4S queue to starve the Classic queue. For example, consider the case where a continually busy L4S queue blocks a DNS request in the Classic queue, arbitrarily delaying the start of a new Classic flow.

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], but its configuration parameters are insensitive to link rate and it requires less operations per packet. However, DualPI2 is more responsive and stable over a wider range of RTTs than Curvy RED. As a consequence, DualPI2 has attracted more development and evaluation attention than Curvy RED, leaving the Curvy RED design incomplete and 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.

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 utilize two queues, each with an AQM algorithm. The two queues can be part of a larger queuing hierarchy [I-D.briscoe-tsvwg-l4s-diffserv].

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.

[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). [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 congestion control (Reno) [RFC5681] and other so-called TCP-friendly controls, such as TCP Cubic in its TCP-friendly mode.

The choice of \( k \) is a matter of operator policy, and operators MAY choose a different value using Table 1 and the guidelines in Appendix C.

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 TCP 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 recommended that the AQM in each queue inspects the ECN field to determine what sort of congestion notification to signal, then decides 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 or latency policing):

  + 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)

  + 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

o 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 signal congestion solely using drop, irrespective of the ECN field. Switching to drop if ECN marking is persistently high is required by Section 7 of [RFC3168] and Section 4.2.1 of [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 TCP flows from under-utilizing capacity. For example:
  * for the PI2 algorithm (Appendix A) the queuing delay target is set to 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 TCP 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;

- 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 4.1.1) 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 switching over to 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.

### 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 TCP-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

This specification contains no IANA considerations.

4. Security Considerations

4.1. Overload Handling

Where the interests of users or flows might conflict, it could be necessary to police traffic to isolate any harm to the performance of individual flows. However it is hard to avoid unintended side-effects with policing, and in a trusted environment policing is not necessary. Therefore per-flow policing needs to be separable from a basic AQM, as an option under policy control.

However, a basic DualQ AQM does at least need to handle overload. A useful objective would be for the overload behaviour of the DualQ AQM to be at least no worse than a single queue AQM. However, a trade-off needs to be made between complexity and the risk of either traffic class harming the other. In each of the following three subsections, an overload issue specific to the DualQ is described, followed by proposed solution(s).

Under overload the higher priority L4S service will have to sacrifice some aspect of its performance. Alternative solutions are provided
below that each 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.

4.1.1. Avoiding Classic Starvation: Sacrifice L4S Throughput or Delay?

Priority of L4S is required to be conditional to avoid total starvation of Classic by heavy L4S traffic. This raises the question of whether to sacrifice L4S throughput or L4S delay (or some other policy) to mitigate starvation of Classic:

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.

The scheduling weight of the Classic queue should be small (e.g. 1/16). Then, in most traffic scenarios the scheduler will not interfere and it will not need to - the coupling mechanism and the end-systems will share out the capacity across both queues as if it were a single pool. However, because the congestion coupling only applies in one direction (from C to L), 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 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: To control milder overload of responsive traffic, particularly when close to the maximum congestion signal, 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 of [MEDF]). This scheduler adds tshift 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 (tshift) defines the maximum extra queuing delay of Classic packets relative to L4S.

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

4.1.2. Congestion Signal Saturation: Introduce L4S Drop or Delay?

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 one AQM first saturates to a probability of 100% leaving no room to push back the load any harder. If k>1, L4S will saturate first, even though saturation could be caused by unresponsive traffic in either queue.

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 TCP-like flow that is normally responsive, but above a certain congestion level it will not be able to reduce its congestion window below the minimum of 2 segments [RFC5681], effectively becoming unresponsive. (Note that L4S traffic ought to remain responsive below a window of 2 segments (see [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 tail drop too):

Drop on Saturation: Saturation can be avoided by setting a maximum threshold for 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 queuing delay can be kept at the target in any overload situation, including with unresponsive traffic, and no further measures are required [DualQ-Test].
Delay on Saturation: When L4S marking saturates, instead of switching to drop, the drop and marking probabilities 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.

4.1.3. Protecting against Unresponsive ECN-Capable Traffic

Unresponsive traffic has a greater advantage if it is also ECN-capable. The advantage is undetectable at normal low levels of drop/marking, but it becomes significant with the higher levels of drop/marking typical during overload. This is an issue whether the ECN-capable traffic is L4S or Classic.

This raises the question of whether and when to switch off ECN marking and use solely drop instead, as required by both Section 7 of [RFC3168] and Section 4.2.1 of [RFC7567].

Experiments with the DualPI2 AQM (Appendix A) have shown that introducing 'drop on saturation' at 100% L4S marking addresses this problem with unresponsive ECN as well as addressing the saturation problem. It leaves only a small range of congestion levels where unresponsive traffic gains any advantage from using the ECN capability, and the advantage is hardly detectable [DualQ-Test].

5. Acknowledgements

Thanks to Anil Agarwal, Sowmini Varadhan’s, Gabi Bracha, Nicolas Kuhn, Tom Henderson and David Pullen 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). Bob Briscoe’s contribution was also part-funded by 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.

Tom Henderson <tomh@tomh.org> of CableLabs, US implemented various Coupled DualQ AQMs for ns3, including DualPI2 and DualPIE over point to point and DOCSIS 3.1 link models and conducted extensive evaluations.

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

7. References

7.1. Normative References

[I-D.ietf-tsvwg-ecn-l4s-id]


7.2. Informative References

[Alizadeh-stability]

[AQMmetrics]

[ARED01]

[BBRv1]

[CoDel]

[CRED_Insights]

[DCttH15]

[DOCSIS3.1]


Briscoe, B., "Interactions between Low Latency, Low Loss, Scalable Throughput (L4S) and Differentiated Services", draft-briscoe-.tsvwg-l4s-diffserv-02 (work in progress), November 2018.


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 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, M, G, u, m, %, ... are assumed to be converted to their appropriate multiple or fraction. 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 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)
- recur(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.len()` or `lq.len()` 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);
- `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.
1: dualpi2_params_init(...) {  % Set input parameter defaults
2:   % DualQ Coupled framework parameters
3:   k = 2                                          % Coupling factor
4:   % NOT SHOWN % scheduler-dependent weight or equival’t parameter
5:    limit = MAX_LINK_RATE * 250 ms               % Dual buffer size
6:  }
7:  % PI2 AQM parameters
8:  RTT_max = 100 ms                          % Worst case RTT expected
9:  RTT_typ = 15 ms                           % Typical RTT
10: p_Cmax = min(1/k^2, 1)                 % Max Classic drop/mark prob
11: target = RTT_typ                           % PI AQM Classic queue delay target
12: Tupdate = min(RTT_typ, 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: p_Lmax = 1                               % Max L4S marking prob
16: floor = Th_len / MIN_LINK_RATE
17: if (minTh < floor) {
18:   % L4S ramp AQM parameters
19:   minTh = 475 us                      % L4S min marking threshold in time units
20:   range = 525 us                       % Range of L4S ramp in time units
21:   Th_len = 2 * MTU                      % Min L4S marking threshold in bytes
22:   p_Lmax = 1                               % Max L4S marking prob
23:       floor = Th_len / MIN_LINK_RATE
24:   if (minTh < floor) {
25:     % Shift ramp so minTh >= serialization time of 2 MTU
26:       minTh = floor
27:   }
28:   maxTh = minTh+range                     % L4S max marking threshold in time units
29: }

Figure 2: Example Header Pseudocode for DualQ Coupled PI2 AQM

The overall goal of the code is to maintain the base probability (p’, p-prime as in Section 2.4), which is an internal variable from which the marking and dropping probabilities for L4S and Classic traffic (p_L and p_C) are derived, with p_L in turn being derived from p_CL. 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.len() + cq.len() + 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.len() + cq.len() > 0 )
3:     if ( scheduler() == lq ) {
4:       lq.dequeue(pkt)                      % Scheduler chooses lq
5:       p'_L = laqm(lq.time())                     % Native L4S AQM
6:       p_L = max(p'_L, p_CL)                 % Combining function
7:       if ( recur(p_L) )                          % Linear marking
8:         mark(pkt)
9:     } else {
10:       cq.dequeue(pkt)                      % Scheduler chooses cq
11:         if ( p_C > rand() ) {              % probability p_C = p'^2
12:           if ( ecn(pkt) == 0 ) {           % if ECN field = not-ECT
13:             drop(pkt)                                % squared drop
14:           }
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(likelihood) {      % Returns TRUE with a certain likelihood
24:   count += likelihood
25:   if (count > 1) {
26:     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 enqueuing 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 [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 $T_{update}$ ($T_{update}$ is set in line 13 of Figure 2. It defaults to 16 ms in the reference Linux implementation because it has to be rounded to a multiple of 4 ms).

- 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 queuing 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 a 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 18 & 19 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 $0.17*RTT_{typ}$, 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 a default minimum ECN marking threshold of about $RTT_{typ}/30$ is recommended.

A minimum marking threshold parameter ($Th_{len}$) in transmission units (default 2 MTU) is also necessary to ensure that the ramp does not trigger excessive marking on slow links. The code in lines 24-27 of the initialization function (Figure 2) converts 2 MTU into time units...
and shifts the ramp so that the min threshold is no shallower than this floor.

1:  laqm(qdelay) {
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) {
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

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’ - see [RFC8033]); 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 two ‘gain factors’ in line 3, alpha and beta, respectively weight how strongly each of these elements ((a) and (b)) alters p’. They are in units of ‘per second of delay’ or Hz, because they transform
alpha and beta determine how much $p'$ ought to change after each update interval ($T_{update}$). For smaller $T_{update}$, $p'$ should change by the same amount per second, but in finer more frequent steps. So alpha depends on $T_{update}$ (see line 14 of the initialization function in Figure 2). It is best to update $p'$ as frequently as possible, but $T_{update}$ will probably be constrained by hardware performance. As shown in line 13, the update interval should be at least as frequent as once per the RTT of a typical flow ($RTT_{typ}$) as long as it does not exceed roughly $RTT_{max}/3$. For link rates from 4 - 200 Mb/s, a target RTT of 15ms and a maximum RTT of 100ms, it has been verified through extensive testing that $T_{update}=16$ms (as recommended in [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 ($RTT_{max}$).

Recommended derivations of the gain constants alpha and beta can be approximated for Reno over a PI2 AQM as: $alpha = 0.1 * T_{update} / RTT_{max}^2$; $beta = 0.3 / RTT_{max}$, as shown in lines 14 & 15 of Figure 2. These are derived from the stability analysis in [PI2]. For the default values of $T_{update}=16$ 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 $p_{CL} = k * p'$ and $p_C = p'^2$.

Because the coupled L4S marking probability ($p_{CL}$) 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 Hz and $k=2$, effective L4S alpha = 0.32 Hz).

Unlike in PIE [RFC8033], alpha and beta do not need to be tuned every $T_{update}$ 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.

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 must be measured in time, not bytes or packets [AQMmetrics] [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 [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 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 that 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 4 of Figure 5 because, if minTh is equal to maxTh, the condition for this ramp calculation cannot arise.

A.2. Pass #2: Overload Details

Figure 7 repeats the dequeue function of Figure 4, but with overload details added. Similarly Figure 8 repeats the core PI algorithm of
Figure 6 with overload details added. The initialization, enqueue, L4S AQM and recur functions are unchanged.

In line 10 of the initialization function (Figure 2), the maximum Classic drop probability $p_{C_{\text{max}}} = \min(1/k^2, 1)$ or 1/4 for the default coupling factor $k=2$. $p_{C_{\text{max}}}$ is the point at which it is deemed that the Classic queue has become persistently overloaded, so it switches to using drop, even for ECN-capable packets. ECT packets that are not dropped can still be ECN-marked.

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 $p_{C_{\text{max}}}$ 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 $p_{L_{\text{max}}}$ 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’ TCP. So it is correct that, when the L4S queue drops packets, it drops them proportional to $p'^2$, as if they are Classic packets.

Both these switch-overs are triggered by the tests for overload introduced 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 $p_{L_{\text{max}}}$ = 1, all remaining packets will be marked because, to have reached the else block at line 8b, $p_{CL} >= 1$.

Lines 2c to 2d in the core PI algorithm (Figure 8) deal with overload of the L4S queue when there is 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 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, the test at line 2a of the full PI update function in Figure 8 keeps delay on target using drop. If the test at line 2a of finds that the Classic queue is empty, line 2d measures the current
queue delay using the L4S queue instead. While the L4S queue is not overloaded, its delay will always be tiny compared to the target Classic queue delay. So p_CL will be driven to zero, and the L4S queue will naturally be governed solely by p'_L from the native L4S AQM (lines 5 and 6 of the dequeue algorithm in Figure 7). But, if unresponsive L4S source(s) cause overload, the DualQ transitions smoothly to L4S marking based on the PI algorithm. If overload increases further, it naturally transitions from marking to dropping by the switch-over mechanism already described.

```c
1:  dualpi2_dequeue(lq, cq, pkt) {     % Couples L4S & Classic queues
2:    while ( lq.len() + cq.len() > 0 ) {
3:      if ( scheduler() == lq ) {
4a:        lq.dequeue(pkt)                             % L4S scheduled
4b:        if ( p_CL < p_Lmax ) { % Check for overload saturation
5:          p'_L = laqm(lq.time())                   % Native L4S AQM
6:          p_L = max(p'_L, p_CL)                % Combining function
7:          if ( recur(p_L) )                        % Linear marking
8a:            mark(pkt)                           % overload saturation
8b:          } else {
8c:            if ( p_C > rand() ) { % probability p_C = p'^2
8d:              drop(pkt) % revert to Classic drop due to overload
8e:              continue        % continue to the top of the while loop
8g:          }    
8h:            if ( p_CL > rand() )           % probability p_CL = k * p'
8i:              mark(pkt)         % linear marking of remaining packets
8j:          }
9:      } else {
10:       cq.dequeue(pkt)                         % Classic scheduled
11:       if ( p_C > rand() ) { % 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:  }                                  % no packet to dequeue
21: return(NULL)                             % no packet to dequeue
22: }
```

Figure 7: Example Dequeue Pseudocode for DualQ Coupled PI2 AQM
(Including Overload Code)
1: dualpi2_update(lq, cq) { % Update p’ every Tupdate
2a:   if ( cq.len() > 0 )
2b:     curq = cq.time() %use queuing time of first-in Classic packet
2c:   else                                      % Classic queue empty
2d:     curq = lq.time()    % use queuing time of first-in L4S packet
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 4.1).

- 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. 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,
based 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 will only measure the delay

of the burst once the burst is over, even though the queue knew about it from the start. At the cost of more operations and more storage, a ‘scaled sojourn time’ metric of queue delay can be used, 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, because it would starve Classic if L4S was overloaded.

**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.len() or lq.len() 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
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: ii) marking depends on queuing time, not bytes, in order to scale for any link rate without being reconfigured; ii) marking of the L4S queue does not depend on itself, it depends on the queuing time of the _other_ (Classic) queue, where cq.time() is the queuing time of the packet at the head of the Classic queue (zero if empty); iii) marking depends on the instantaneous queuing time (of the other Classic queue), not a smoothed average; 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 excess of the Classic queueing delay qc.time() above 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 marking probability will linearly increase with queuing 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 in the Classic queue, where qc.time() is the current instantaneous queueing time of the Classic queue 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 queuing time \( Q_C \) is divided by the Classic scaling parameter \( \text{range}_C \), in the same way that queuing time was scaled for L4S marking. This scaled queuing 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 queuing 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 queuing delay to a fixed target seemed misguided [CRED_Insights]. As the number of flows increases, if the AQM does not allow TCP to increase queuing delay, it has to introduce abnormally high levels of loss. Then loss rather than queuing 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 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 required.

From our limited experiments with Curvy RED so far, recommended values of these parameters are: \( S_C = -1 \); \( g_C = 5 \); \( T = 5 \times \text{MTU} \) 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 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.

```c
1:  cred_dequeue(lq, cq, pkt) { % Couples L4S & Classic queues
2:    while ( lq.len() + cq.len() > 0 ) {
3:      if ( scheduler() == lq ) {
4:        lq.dequeue(pkt) % L4S scheduled
5:        if ((lq.time() > T) OR (cq.ns() >> (S_L-2) > maxrand(U)))
6:          mark(pkt)
7:      } else {
8:        cq.dequeue(pkt) % Classic scheduled
9:        Q_C += (cq.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: } % 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 <= 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 lines 5 and 10 queuing times cq.ns() and pkt.ns() are returned in
integer nanoseconds, making the values about $2^{30}$ times larger than when the units were seconds, ii) then in lines 3 and 9 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 ($\gg g_C$).

Appendix C. Guidance on Controlling Throughput Equivalence

+---------------+------+-------+
| RTT_C / RTT_L | Reno | Cubic |
+---------------+------+-------+
|             1 | k'=1 | k'=0  |
|             2 | k'=2 | k'=1  |
|             3 | k'=2 | k'=2  |
|             4 | k'=3 | k'=2  |
|             5 | k'=3 | k'=3  |
+---------------+------+-------+

Table 1: Value of $k'$ for which DCTCP throughput is roughly the same as Reno or Cubic, for some example RTT ratios

$k'$ is related to $k$ in Equation (1) (Section 2.1) by $k=2^k'$.

To determine the appropriate policy, the operator first has to judge whether it wants DCTCP flows to have roughly equal throughput with Reno or with Cubic (because, even in its Reno-compatibility mode, Cubic is about 1.4 times more aggressive than Reno). Then the operator needs to decide at what ratio of RTTs it wants DCTCP and Classic flows to have roughly equal throughput. For example choosing $k'=0$ (equivalent to $k=1$) will make DCTCP throughput roughly the same as Cubic, __if their RTTs are the same__. However, even if the base RTTs are the same, the actual RTTs are unlikely to be the same, because Classic (Cubic or Reno) traffic needs a large queue to avoid under-utilization and excess drop, whereas L4S (DCTCP) does not. The operator might still choose this policy if it judges that DCTCP throughput should be rewarded for keeping its own queue short.

On the other hand, the operator will choose one of the higher values for $k'$, if it wants to slow DCTCP down to roughly the same throughput as Classic flows, to compensate for Classic flows slowing themselves down by causing themselves extra queuing delay.
The values for \( k' \) in the table are derived from the formulae, which was developed in [DCttH15]:

\[
2^{k'} = 1.64 \left( \frac{\text{RTT}_\text{reno}}{ \text{RTT}_\text{dc}} \right) \quad (2)
\]

\[
2^{k'} = 1.19 \left( \frac{\text{RTT}_\text{cubic}}{ \text{RTT}_\text{dc}} \right) \quad (3)
\]

For localized traffic from a particular ISP’s data centre, using the measured RTTs, it was calculated that a value of \( k' = 3 \) (equivalent to \( k = 8 \)) would achieve throughput equivalence, and experiments verified the formula very closely.

For a typical mix of RTTs from local data centres and across the general Internet, a value of \( k' = 1 \) (equivalent to \( k = 2 \)) is recommended as a good workable compromise.

Authors’ Addresses

Koen De Schepper  
Nokia Bell Labs  
Antwerp  
Belgium  

Email: koen.de_schepper@nokia.com  
URI: https://www.bell-labs.com/usr/koen.de_schepper

Bob Briscoe (editor)  
CableLabs  
UK  

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

Greg White  
CableLabs  
Louisville, CO  
US  

Email: G.White@cablelabs.com
Packetization Layer Path MTU Discovery for Datagram Transports
draft-ietf-tsvwg-datagram-plpmtud-08

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 network 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 functionally for datagram transports that is equivalent to the Packetization Layer PMTUD specification for TCP, specified 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 4821.

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 7 December 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. Classical Path MTU Discovery ...................... 4
   1.2. Packetization Layer Path MTU Discovery .......... 6
   1.3. Path MTU Discovery for Datagram Services .......... 7
2. Terminology ............................................. 7
3. Features Required to Provide Datagram PLPMTUD ........ 10
4. DPLPMTUD Mechanisms ..................................... 12
   4.1. PLPMTU Probe Packets .............................. 12
   4.2. Confirmation of Probed Packet Size ................ 14
   4.3. Detection of Unsupported PLPMTU Size, aka Black Hole Detection .......................... 14
   4.4. Response to PTB Messages ........................... 15
   4.4.1. Validation of PTB Messages ....................... 15
   4.4.2. Use of PTB Messages .............................. 16
5. Datagram Packetization Layer PMTUD .................... 17
   5.1. DPLPMTUD Components ............................... 18
   5.1.1. Timers ........................................... 18
   5.1.2. Constants ......................................... 19
   5.1.3. Variables ......................................... 20
   5.1.4. Overview of DPLPMTUD Phases .................... 21
   5.2. State Machine ....................................... 23
   5.3. Search to Increase the PLPMTU ..................... 26
   5.3.1. Probing for a larger PLPMTU .................... 26
   5.3.2. Selection of Probe Sizes ....................... 27
   5.3.3. Resilience to Inconsistent Path Information .... 27
   5.4. Robustness to Inconsistent Paths .................. 28
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 may 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, and a method that 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 are sometimes 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 sent with this size, or larger, 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 stateful 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 needs to also 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 the 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 utilising 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.

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, 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] 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 PMTU discovery with TCP.

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 with a progressively 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 no response is received to a probe packet, the method reduces the probe size. The result of probing with the PLPMTU is used to set the application MPS.

PLPMTUD introduces flexibility in the implementation of PMTU discovery. At one extreme, it can be configured to only perform ICMP black Hole Detection and recovery to increase the robustness of Classical PMTUD, or at the other extreme, all PTB processing can be disabled and PLPMTUD can completely replace 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 described relies on 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 UDP Usage Guidelines [RFC8085] 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. Prior to this document, PLPMTUD had not been specified for UDP.

Section 10.2 of [RFC4821] recommends 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], but RFC4821 does not provide a complete specification. The present document provides the details to complete that 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 could be used with DCCP.

Section 6 specifies the method for a set of transports, and provides information to enable the implementation of PLPMTUD with other datagram transports and applications that use datagram transports.

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.

Other terminology is directly copied from [RFC4821], and the definitions in [RFC1122].
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 endpoint. Two types of Black Hole are relevant to DPLPMTUD:

Packet Black Hole: 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).

ICMP Black Hole: 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.

Black holed: Traffic is black-holed when the sender is unaware that packets are not being delivered. This could be due to a Packet Black Hole or an ICMP Black Hole.

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 datagram 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.

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 by EMTU_R (Effective MTU to receive).

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 and 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.

MAX_PMTU: The MAX_PMTU is the largest size of PLPMTU that DPLPMTUD will attempt to use.

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. In DPLPMTUD this quantity is derived from the PLPMTU by taking into consideration the size of the lower protocol layer headers. Probe packets generated by DPLPMTUD can have a size larger than the MPS.

MIN_PMTU: The MIN_PMTU is the smallest size of PLPMTU that DPLPMTUD will attempt to use.

Packet: A Packet is the IP header plus the IP payload.

Packetization Layer (PL): The Packetization Layer (PL) is the layer of the network stack that places data into packets and performs transport protocol functions.

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.

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.

PLPMTU: The Packetization Layer PMTU is an estimate of the actual PMTU provided by the DPLPMTUD algorithm.

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.

3. Features Required to Provide Datagram PLPMTUD

TCP PLPMTUD has been defined using standard TCP protocol mechanisms. All of the requirements in [RFC4821] also apply to the use of the technique with a datagram PL. Unlike TCP, some datagram PLs require additional mechanisms to implement PLPMTUD.

There are eight requirements for performing the datagram PLPMTUD method described in this specification:

1. PMTU parameters: A DPLPMTUD sender is RECOMMENDED to provide information about the maximum size of packet that can be transmitted by the sender on the local link (the local Link MTU). It MAY utilize similar information about the receiver when this is supplied (note this could be less than EMTU_R). This avoids implementations trying to send probe packets that can not be transmitted 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).

2. PLPMTU: A datagram application using a PL not supporting fragmentation is REQUIRED to be able to choose the size of datagrams sent to the network, up to the PLPMTU, or a smaller value (such as the MPS) derived from this. This value is managed by the DPLPMTUD method. The PLPMTU (specified as the effective PMTU in Section 1 of [RFC1191]) is equivalent to the EMTU_S (specified in [RFC1122]).

3. Probe packets: On request, a DPLPMTUD sender is REQUIRED to be able to transmit a packet larger than the PLPMTU. This is used to send a probe packet. 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]).
4. 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].

5. 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. The mechanism needs to be robust to the possibility that packets could be significantly delayed along a network path. The local PL endpoint at the sending node is REQUIRED to pass this feedback to the sender DPLPMTUD method.

6. Probe loss recovery: It is RECOMMENDED to use probe packets that do not carry any user data. 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. DPLPMTUD is REQUIRED to be robust in the case where probe packets are lost due to other reasons (including link transmission error, congestion).

7. Probing and congestion control: The DPLPMTUD sender treats isolated loss of a probe packet (with or without a corresponding PTB message) as a potential indication of a PMTU limit for the path. Loss of a probe packet SHOULD NOT be treated as an indication of congestion and the loss SHOULD NOT directly trigger a congestion control reaction [RFC4821].

8. Shared PLPMTU state: The PLPMTU value could also be stored with the corresponding entry in the destination 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, PLPMTU adjustments based on shared PLPMTU values should be incorporated in the search algorithms. 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:

* MPS: A method is REQUIRED to signal an appropriate MPS to the higher layer using the PL. The value of the MPS can change following a change to the path. It is RECOMMENDED that methods avoid forcing an application to use an arbitrary small MPS (PLPMTU) for transmission while the method is searching for the currently supported PLPMTU. Datagram PLs do not necessarily support fragmentation of PDUs larger than the PLPMTU. A reduced MPS can adversely impact the performance of a datagram application.

* 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.

* 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 needs to construct 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) could alternatively prefer to generate a probe packet by extending a control message with padding data.

A receiver needs 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 listed in order of preference:

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 required for 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 required for the probe packet. If the application/transport needs protection from the loss of this probe packet, the application/transport 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).

Probing using application data: A probe packet that contains a data block supplied by an application that matches the size required for the probe packet. This method requests the application to issue a data block of the desired probe size. If the application/transport needs protection from the loss of an unsuccessful probe packet, the application/transport needs then to perform transport-layer retransmission/repair of the data block (e.g., by retransmission after loss is detected).

A PL that uses a probe packet carrying an application data block, could need to retransmit this application data block if the probe fails. This could need the PL to re-fragment the data block to a smaller packet size that is expected to traverse the end-to-end path (which could utilize endpoint network-layer or PL fragmentation when these are available).

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, 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 and SCTP provide keep-alive/heartbeat features). When supported, this mechanism SHOULD 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 either rely on an application protocol to detect this loss, or make use of an additional transport method such as UDP-Options [I-D.ietf-tsvwg-udp-options].

Section 6 specifies this function for a set of IETF-specified protocols.

4.3. Detection of Unsupported PLPMTU Size, aka Black Hole Detection

A PL sender needs to reduce the PLPMTU when it discovers the actual PMTU supported by a network path is less than the PLPMTU. This can be triggered when a validated PTB message is received, or by another event that indicates the network path no longer sustains the current packet size, such as a loss report from the PL, or repeated lack of response to probe packets sent to confirm the PLPMTU. Detection is followed by a reduction of the PLPMTU.

This is performed by sending packet probes of size PLPMTU to verify that a network path still supports the last acknowledged PLPMTU size. There are two alternative mechanisms:

* A PL can rely upon 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 PMTU (as in PLPMTUD for TCP [RFC4821]).

* 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 acknowledgement, PROBE_COUNT, becomes greater than MAX_PROBES).
A PL 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-data PLPMTU once user data is sent again, MAY choose to continue PLPMTU discovery for each path. However, this may result in additional packets being sent.

When the method detects the current PLPMTU is not supported, DPLPMTUD sets a lower MPS. The PL then confirms that the updated PLPMTU can be successfully used across the path. The PL could need to send a probe packet with a size less than the size of the data block generated by an application. In this case, the PL could provide a way to fragment a datagram at the PL, or use a control packet as the packet probe.

4.4. 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 PTB_SIZE indicated in the PTB message, the state of the PLPMTUD state machine, and the IP protocol being used.

Section 4.4.1 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.

4.4.1. Validation of PTB Messages

This section specifies utilization 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.

* An implementation that supports PTB messages MUST validate messages before they are further processed.

A PL that receives a PTB message from a router or middlebox, performs ICMP validation as specified in Section 5.2 of [RFC8085] [RFC8201]. 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...
The validation SHOULD utilize information that it is not simple for an off-path attacker to determine [RFC8085]. For example, by checking 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.

PTB messages that have been validated MAY be utilized by the DPLPMTUD algorithm, but MUST NOT be used directly to set the PLPMTU. A method that utilizes these PTB messages can improve the speed at which the algorithm detects an appropriate PLPMTU, compared to one that relies solely on probing. Section 4.4.2 describes this processing.

### 4.4.2. Use of PTB Messages

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 PTB_SIZE is less than the size used by probe packets and larger than minimum size accepted.

This section provides a summary of how PTB messages can be utilized. This processing depends on the PTB_SIZE and the current value of a set of variables:

- **MIN_PMTU < PTB_SIZE < BASE_PMTU**
  - A robust PL MAY enter an error state (see Section 5.2) for an IPv4 path when the PTB_SIZE reported in the PTB message is larger than or equal to 68 bytes and when this is less than the BASE_PMTU.
  - A robust PL MAY enter an error state (see Section 5.2) for an IPv6 path when the PTB_SIZE reported in the PTB message is larger than or equal to 1280 bytes and when this is less than the BASE_PMTU.

- **PTB_SIZE = PLPMTU**
  - Completes the search for a larger PLPMTU.

- **PTB_SIZE > PROBED_SIZE**
  - Inconsistent network signal.
* PTB message ought to be discarded without further processing
  (e. g. PLPMTU not modified).

* The information could be utilized as an input to trigger
  enabling a resilience mode.

BASE_PMTU <= PTB_SIZE < PLPMTU
* Black Hole Detection is triggered and the PLPMTU ought to be
  set to BASE_PMTU.

* The PL could use the PTB_SIZE reported in the PTB message to
  initialize a search algorithm.

PLPMTU < PTB_SIZE < PROBED_SIZE
* The PLPMTU continues to be valid, but the last PROBED_SIZE
  searched was larger than the actual PMTU.

* The PLPMTU is not updated.

* The PL can use the reported PTB_SIZE from the PTB message as
  the next search point when it resumes the search algorithm.

xxx Author Note: Do we want to specify how to handle PTB Message with
PTB_SIZE = 0? xxx

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 NOT be used by an application if it is already
used in a lower layer.
The central idea of DPLPMTUD is probing by a sender. Probe packets are sent to find the maximum size of a 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 [RFC8085].

If the PL has a path Round Trip Time (RTT) estimate and timely acknowledgements the
PROBE_TIMER can be derived from the PL RTT estimate.

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 secs, 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-data PMTU once user data is sent again, can choose to continue PMTU discovery for each path. However, this may 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 [RFC8085].

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-data PMTU once user data is sent again, can choose to continue PMTU discovery for each path. However, this may result in sending additional packets.

An implementation could implement the various timers 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). The default value of MAX_PROBES is 10.

**MIN_PMTU:** The MIN_PMTU is the smallest allowed probe packet size. For IPv6, this value is 1280 bytes, as specified in [RFC2460]. For IPv4, the minimum value is 68 bytes.

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_PMTU:** The MAX_PMTU is the largest size of PLPMTU. This has to be less than or equal to the minimum of the local MTU of the outgoing interface and the destination PMTU for receiving. An application, or PL, MAY reduce the MAX_PMTU when there is no need to send packets larger than a specific size.

**BASE_PMTU:** The BASE_PMTU is a configured size expected to work for most paths. The size is equal to or larger than the MIN_PMTU and smaller than the MAX_PMTU. In the case of IPv6, this value is 1280 bytes [RFC2460]. When using IPv4, a size 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. 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 unsuccessful probe packets that have been sent with a size of PROBED_SIZE. The value is initialized to zero when a particular size of PROBED_SIZE is first attempted.

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 towards the actual PMTU.

```
MIN_PMTU   MAX_PMTU
<------------------------->
 |                         |
 v                         v
 BASE_PMTU                Actual PMTU
 v                         v
 PLPMTU

Figure 2: 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.

```
+------+
| Base |----------------+ Connectivity or BASE_PMTU confirmation failed
+------+
 | Connectivity and BASE_PMTU confirmed +-------+
 | Error |
 | Consistent connectivity and BASE_PMTU confirmed +-------+
v PLPMTU confirmation failed +---------+
 | Search |----------------+ confirmed
 | Search algorithm completed |
 | Search Complete |
```

Figure 3: DPLPMTUD Phases
BASE_PMTU Confirmation Phase
* The BASE_PMTU Confirmation Phase confirms connectivity to the remote peer. This phase is implicit for a connection-oriented PL (where it can be performed in a PL connection handshake). A connectionless PL needs to send an acknowledged probe packet to confirm that the remote peer is reachable.
* The sender also confirms that BASE_PMTU is supported across the network path.
* A PL that does not wish to support a path with a PLPMTU less than BASE_PMTU can simplify the phase into a single step by performing the connectivity checks with a probe of the BASE_PMTU size.
* Once confirmed, DPLPMTUD enters the Search Phase. If this phase fails to confirm, DPLPMTUD enters the Error Phase.

Search Phase
* 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 4.4.
* Black Hole Detection can also terminate the search by entering the BASE_PMTU Confirmation phase.

Search Complete Phase
* 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, DPLPMTUD enters the BASE_PMTU Confirmation phase.
* The PMTU_RAISE_TIMER is used to periodically resume the search phase to discover if the PLPMTU can be raised.
* Black Hole Detection or receipt of a validated PTB message (Section 4.4.1) can cause the sender to enter the BASE_PMTU Confirmation Phase.
Error Phase

* The Error Phase is entered when there is conflicting or invalid PLPMTU information for the path (e.g., a failure to support the BASE_PMTU) 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_PMTU (or DPLPMTUD is suspended).

* Note: MIN_PMTU may be identical to BASE_PMTU, simplifying the actions in this phase.

An implementation that only reduces the PLPMTU to a suitable size would be sufficient to ensure reliable operation, but can be very inefficient when the actual MTU 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 4. If multipath or multihoming is supported, a state machine is needed for each path.

Note: Some state changes are not shown to simplify the diagram.
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.

Figure 4: State machine for Datagram PLPMTUD
connectivity. This state is left, once the PL indicates connectivity to the remote PL.

**BASE:**

The BASE state is used to confirm that the BASE_PMTU 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 where traffic is black holed while searching for a larger PLPMTU.

On entry, the PROBED_SIZE is set to the BASE_PMTU 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_PMTU was successful.

The PROBE_COUNT is set to zero when the first probe packet is sent for each probe size. Each time a probe packet is acknowledged, the PLPMTU is set to the PROBED_SIZE, and then the PROBED_SIZE is increased using the search algorithm.

When a probe packet is sent and not acknowledged within the period of the PROBE_TIMER, the PROBE_COUNT is incremented and the probe packet is retransmitted. The state is exited when the PROBE_COUNT reaches MAX_PROBES, a received PTB message is validated, a probe of size MAX_PMTU is acknowledged, or a black hole is detected.

**SEARCH_COMPLETE:**

The SEARCH_COMPLETE state indicates a successful end to the SEARCHING state. DPLPMTUD remains in this state until either the PMTU_RAISE_TIMER expires, a received PTB message is validated, 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 the probe packet fails to be acknowledged after MAX_PROBES attempts, 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_PMTU size or when there is contradictory information about the network path that would otherwise result in excessive variation in the MPS signalled 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 or when the PL indicates that connectivity has been lost.

Implementations are permitted to enable endpoint fragmentation if the DPLPMTUD is unable to validate MIN_PMTU within PROBE_COUNT probes. If DPLPMTUD is unable to validate MIN_PMTU the implementation should transition to the DISABLED 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_PMTU. MAX_PMTU is the minimum of the local MTU and EMTU_R (learned from the remote endpoint). The MAX_PMTU 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 a probe packet is first sent with a particular size. A timer is used by the search algorithm
to trigger the sending of probe packets of size PROBED_SIZE, larger than the PLPMTU. Each probe packet successfully sent to the remote peer is confirmed by acknowledgement 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 probe packet of the same size is retransmitted (the replicated probe improve the resilience to loss). The maximum number of retransmissions for a particular size 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 needs to determine 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 and has the undesirable effect of slowing the time to reach a more optimal MPS. 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 implementor 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 it sends 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 due to MTU limitation.

5.4. Robustness to Inconsistent Paths

Some paths could be unable to sustain packets of the BASE_PMTU size. To be robust to these paths an implementation could implement the Error State. 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_PMTU.


This section specifies protocol-specific details for datagram PLPMTUD for IETF-specified transports.

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.

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 layer 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 [RFC8085].

Some primitives used by DPLPMTUD might not be available via the Datagram API (e.g., the ability to access the PLPMTU cache, or interpret received PTB messages).

In addition, it is desirable that PMTU discovery is not performed by multiple protocol layers. An application SHOULD avoid using DPLPMTUD when the underlying transport system provides this capability. To
use common method for managing the PLPMTU has benefits, both in the ability to share state between different processes and opportunities to coordinate probing.

6.1.1. Application Request

An application needs an application-layer protocol mechanism (such as a message acknowledgement 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 [RFC8085], 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 may 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 that may carry an application data block, but the successful transmission of this data is at risk when used for probing. Some applications may prefer to use a probe packet that does not carry an application data block to avoid disruption to data transfer.

6.1.4. 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.

6.1.5. 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 [RFC8085]. This requires that the application to check each received PTB messages to validate it is received in response to transmitted traffic and that the reported PTB_SIZE is less than the current probed size (see Section 4.4.2). A validated PTB message MAY be used as input to the DPLPMUT algorithm, but MUST NOT be used directly to set the PLPMTU.
6.2. DPLPMTUD for SCTP

Section 10.2 of [RFC4821] specifies a recommended PLPMTUD probing method for SCTP. It recommends the use of the PAD chunk, defined in [RFC4820] to be attached to a minimum length HEARTBEAT chunk to build a probe packet. This enables probing without affecting the transfer of user messages and without interfering with congestion control. This is preferred to using DATA chunks (with padding as required) as path probes.

XXX Author Note: Future versions of this document might define a parameter contained in the INIT and INIT ACK chunk to indicate the remote peer MTU to the local peer. However, multihoming makes this a bit complex, so it might not be worth doing. XXX

6.2.1. SCTP/IPv4 and SCTP/IPv6

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.

6.2.1.1. 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.

The HEARTBEAT chunk carries a Heartbeat Information parameter which should include, besides the information suggested in [RFC4960], the probe size, which is the size of the complete datagram. The size of the PAD chunk is therefore computed by reducing the probing size by the IPv4 or IPv6 header size, the SCTP common header, the HEARTBEAT request and the PAD chunk header. The payload of the PAD chunk contains arbitrary data.

To avoid fragmentation of retransmitted data, probing starts right after the PL handshake, before data is sent. Assuming this behavior (i.e., the PMTU is smaller than or equal to the interface MTU), this process will take a few round trip time periods depending on the number of PMTU sizes probed. The Heartbeat timer can be used to implement the PROBE_TIMER.

6.2.1.2. 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.
6.2.1.3. 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 PTB_SIZE reported in the PTB message SHOULD be used with the DPLPMTUD algorithm, providing that the reported PTB_SIZE is less than the current probe size (see Section 4.4).

6.2.2. DPLPMTUD for SCTP/UDP

The UDP encapsulation of SCTP is specified in [RFC6951].

6.2.2.1. Sending SCTP/UDP Probe Packets

Packet probing can be performed as specified in Section 6.2.1.1. The maximum payload is reduced by 8 bytes, which has to be considered when filling the PAD chunk.

6.2.2.2. Validating the Path with SCTP/UDP

Since SCTP provides an acknowledged PL, a sender MUST NOT implement the CONFIRMATION_TIMER while in the SEARCH_COMPLETE state.

6.2.2.3. 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 PTB_SIZE indicated in the PTB message SHOULD be used with the DPLPMTUD providing that the reported 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]. It is used for data channels in WebRTC implementations.

6.2.3.1. Sending SCTP/DTLS Probe Packets

Packet probing can be done as specified in Section 6.2.1.1.
6.2.3.2. 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.

6.2.3.3. Handling of PTB Messages by SCTP/DTLS

It is not possible to perform ICMP validation as specified in [RFC4960], since even if the ICMP message payload contains sufficient information, the reflected SCTP common header would be encrypted. Therefore it is not possible to process PTB messages at the PL.

6.3. DPLPMTUD for QUIC

Quick UDP Internet Connection (QUIC) [I-D.ietf-quic-transport] is a UDP-based transport that provides reception feedback. The UDP payload includes the QUIC packet header, protected payload, and any authentication fields. QUIC depends on a PMTU of at least 1280 bytes.

Section 14.1 of [I-D.ietf-quic-transport] describes the path considerations when sending QUIC packets. It recommends the use of PADDING frames to build the probe packet. Pure probe-only packets are constructed with PADDING frames and PING frames to create a padding only packet that will elicit an acknowledgement. Such padding only packets enable probing without affecting the transfer of other QUIC frames.

The recommendation for QUIC endpoints implementing DPLPMTUD is that a MPS is maintained for each combination of local and remote IP addresses [I-D.ietf-quic-transport]. If a QUIC endpoint determines that the PMTU between any pair of local and remote IP addresses has fallen below an acceptable MPS, it needs to immediately cease sending QUIC packets on the affected path. This could result in termination of the connection if an alternative path cannot be found [I-D.ietf-quic-transport].

6.3.1. Sending QUIC Probe Packets

A probe packet consists of a QUIC Header and a payload containing PADDING Frames and a PING Frame. PADDING Frames are a single octet (0x00) and several of these can be used to create a probe packet of size PROBED_SIZE. QUIC provides an acknowledged PL, a sender can therefore enter the BASE state as soon as connectivity has been confirmed.

The current specification of QUIC sets the following:
* BASE_PMTU: 1200. A QUIC sender needs to pad initial packets to 1200 bytes to confirm the path can support packets of a useful size.

* MIN_PMTU: 1200 bytes. A QUIC sender that determines the PMTU has fallen below 1200 bytes MUST immediately stop sending on the affected path.

6.3.2. Validating the Path with QUIC

QUIC provides an acknowledged PL. A sender therefore MUST NOT implement the CONFIRMATION_TIMER while in the SEARCH_COMPLETE state.

6.3.3. Handling of PTB Messages by QUIC

QUIC operates over the UDP transport, and the guidelines on ICMP validation as specified in Section 5.2 of [RFC8085] therefore apply. In addition to UDP Port validation QUIC can validate an ICMP message by looking for valid Connection IDs in the quoted packet.

6.4. DPLPMTUD for UDP-Options

UDP Options [I-D.ietf-tsvwg-udp-options] provides a way to extend UDP to provide new transport mechanisms.

Support for using DPLPMTUD with UDP-Options is defined in the UDP-Options specification [I-D.ietf-tsvwg-udp-options].

7. Acknowledgements

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).

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 references RFCs. Security guidance for applications using UDP is provided in the UDP Usage Guidelines [RFC8085], specifically the generation of probe packets is regarded as a "Low Data-Volume Application", described in section 3.1.3 of this document. This
recommends that sender limits generation of probe packets to an average rate lower than one probe per 3 seconds.

A PL sender needs to ensure that the method used to confirm reception of probe packets offers protection from off-path attackers injecting packets into the path. This protection if 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 [RFC8085]).

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 4.4.1).

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. 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).

Parallel forwarding paths SHOULD be considered. Section 5.4 identifies the need for robustness in the method when the path information may be inconsistent.

A node performing DPLPMTUD could experience conflicting information about the size of supported probe packets. This could occur when there are multiple paths are concurrently in use and these exhibit a different PMTU. If not considered, this could result in data being black holed when the PLPMTU is larger than the smallest PMTU across the current paths.

10. References

10.1. Normative References


10.2. Informative References

[I-D.ietf-intarea-tunnels]

[I-D.ietf-tsvwg-udp-options]


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
  evlauated 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 PLPMTIU.

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.

Authors’ Addresses

Godred Fairhurst  
University of Aberdeen  
School of Engineering, Fraser Noble Building  
Aberdeen  
AB24 3UE  
United Kingdom

Email: 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

Michael Tuexen  
Muenster University of Applied Sciences  
Stegerwaldstrasse 39  
48565 Steinfurt  
Germany

Email: tuexen@fh-muenster.de
Irene Ruengeler  
Muenster University of Applied Sciences  
Stegerwaldstrasse 39  
48565 Steinfurt  
Germany  
Email: i.ruengeler@fh-muenster.de

Timo Voelker  
Muenster University of Applied Sciences  
Stegerwaldstrasse 39  
48565 Steinfurt  
Germany  
Email: timo.voelker@fh-muenster.de
Guidelines for Adding Congestion Notification to Protocols that Encapsulate IP
draft-ietf-tsvwg-ecn-encap-guidelines-13

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 21, 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. 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 (to be removed by RFC Editor) .......... 25
8. Security Considerations ...................................... 26
9. Conclusions .................................................. 26
10. Acknowledgements ............................................ 27
11. Comments Solicited .......................................... 27
12. References .................................................. 27
   12.1. Normative References .................................. 27
   12.2. Informative References ................................ 28
Appendix A. Changes in This Version (to be removed by RFC Editor) ........................................ 33
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:

- 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];

- 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 [RFC1323] (now [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 [RFCXXXX], 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 [RFCXXXX] as an informative reference. (RFC Editor: Please
replace both instances of XXXX above with the number of this 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:

- 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];
- 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 descapsulation, 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 RFC 2119 [RFC2119] 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).

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.
Note that the FECN (forward ECN) bit in Frame Relay [Buck00] and the explicit forward congestion indication (EFCI [ITU-T.I.371]) bit in ATM user data cells follow a feed-forward pattern. However, in ATM, this arrangement is only part of a feed-forward-and-backward pattern at the lower layer, not feed-forward-and-up out of the lower layer--the intention was never to interface to IP ECN at the subnet egress. To our knowledge, Frame Relay FECN is solely used to detect where more capacity should be provisioned.

3.2. Feed-Up-and-Forward Mode

Ethernet is particularly difficult to extend incrementally to support explicit congestion notification. One way to support ECN in such cases has been to use so called 'layer-3 switches'. These are Ethernet switches that dig into the Ethernet payload to find an IP header and manipulate or act on certain IP fields (specifically DiffServ & ECN). For instance, in Data Center TCP [RFC8257], layer-3 switches are configured to mark the ECN field of the IP header within the Ethernet payload when their output buffer becomes congested.

With respect to switching, a layer-3 switch acts solely on the addresses in the Ethernet header; it does not use IP addresses, and it does not decrement the TTL field in the IP header.

![Diagram of Feed-Up-and-Forward Mode](image)

By comparing Figure 2 with Figure 1, it can be seen that subnet E (perhaps a subnet of layer-3 Ethernet switches) works in feed-up-and-forward mode by notifying congestion directly into L3 at the point of congestion, even though the congested switch does not otherwise act at L3. In this example, the technology in subnet F (e.g. MPLS) does...
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.
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:

- 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.

- The subsequent sections give guidelines for adding ECN to a subnet technology that uses feed-forward-and-up mode like IP, but it is
Internet-Draft  ECN Encapsulation Guidelines  May 2019

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. Generic UDP Encapsulation (GUE) [I-D.ietf-intarea-gue] and Geneve [I-D.ietf-nvo3-geneve] 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.
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 RBridges 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 802.1p classes of service 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 802.1p classes.

### 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 severity and the packet’s marking is not the most severe, the packet MAY be forwarded, but it 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, packets or fragments.

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). Where framing boundaries do not necessarily align with packet boundaries, the following guidance will be needed. It explains how to propagate ECN markings from layer-2 frame headers when they are stripped off and IP PDUs with different boundaries are reassembled for forwarding.
Congestion indications SHOULD be propagated on the basis that a congestion indication on a PDU applies to all the octets in the PDU. On average, an encapsulator or decapsulator SHOULD approximately preserve the number of marked octets arriving and leaving (counting the size of inner headers, but not encapsulating headers that are being added or stripped).

The next departing frame SHOULD be immediately marked even if only enough incoming marked octets have arrived for part of the departing frame. This ensures that any outstanding congestion marked octets are propagated immediately, rather than held back waiting for a frame no bigger than the outstanding marked octets—which might involve a long wait.

For instance, an algorithm for marking departing frames could maintain a counter representing the balance of arriving marked octets minus departing marked octets. It adds the size of every marked frame that arrives and if the counter is positive it marks the next frame to depart and subtracts its size from the counter. This will often leave a negative remainder in the counter, which is deliberate.

5. Feed-Up-and-Forward Mode: Guidelines for Adding Congestion Notification

The guidance in this section is applicable, for example, when IP packets:

- are encapsulated in Ethernet headers, which have no support for ECN;
- 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:

- IPv4 and IPv6 are not the only layer-3 protocols that might be encapsulated by lower layer protocols
- Link-layer encryption might be in use, making the layer-2 payload inaccessible
- Many Ethernet switches do not have ‘layer-3 switch’ capabilities so they cannot read or modify an IP payload
- 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 (to be removed by RFC Editor)

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 take 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.:

- 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].

- 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-forward mode.

9. Conclusions

Following the guidance in the 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:

- 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;

- 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 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. 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. References

12.1. Normative References

12.2. Informative References


[I-D.ietf-intarea-gue]

[I-D.ietf-nvo3-geneve]

[I-D.ietf-trill-ecn-support]

[I-D.ietf-tsvwg-ecn-l4s-id]

[I-D.ietf-tsvwg-rfc6040update-shim]

[IEEE802.1Q]

[ITU-T.I.371]

[Leiserson85]


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 feedback 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-rfc6040-update-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
URI:   http://bobbriscoe.net/

John Kaippallimalil
Huawei
5340 Legacy Drive, Suite 175
Plano, Texas  75024
USA
EMail: john.kaippallimalil@huawei.com

Pat Thaler
Broadcom Corporation (retired)
CA
USA
Identifying Modified Explicit Congestion Notification (ECN) Semantics for Ultra-Low Queuing Delay (L4S)
draft-ietf-tsvwg-ecn-l4s-id-07

Abstract

This specification defines the identifier to be used on IP packets for a new network service called low latency, low loss and scalable throughput (L4S). It is similar to the original (or 'Classic') Explicit Congestion Notification (ECN). ‘Classic’ ECN marking was required to be equivalent to a drop, both when applied in the network and when responded to by a transport. Unlike ‘Classic’ ECN marking, for packets carrying the L4S identifier, the network applies 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 ‘Classic’ flow under the same conditions. However, the much more frequent control signals and the finer responses to them result in ultra-low queuing delay without compromising link utilization, and low delay is maintained during high load. Examples of new active queue management (AQM) marking algorithms and examples of new transports (whether TCP-like or real-time) are specified separately. The new L4S identifier is the key piece that enables them to interwork and distinguishes them from ‘Classic’ traffic. It gives an incremental migration path so that existing ‘Classic’ TCP traffic will be no worse off, but it can be prevented from degrading the ultra-low delay and loss of the new scalable transports.

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
   1.1. Problem .................................................. 4
   1.2. Terminology .............................................. 6
   1.3. Scope .................................................... 6
2. Consensus Choice of L4S Packet Identifier: Requirements ........ 7
3. L4S Packet Identification at Run-Time .......................... 8
4. Prerequisite Transport Layer Behaviour .......................... 8
   4.1. Prerequisite Codepoint Setting .......................... 8
   4.2. Prerequisite Transport Feedback .......................... 8
   4.3. Prerequisite Congestion Response .......................... 9
5. Prerequisite Network Node Behaviour .............................. 11
   5.1. Prerequisite Classification and Re-Marking Behaviour .... 11
   5.2. The Meaning of L4S CE Relative to Drop .................. 11
   5.3. Exception for L4S Packet Identification by Network Nodes with Transport-Layer Awareness ................. 12
   5.4. Interaction of the L4S Identifier with other Identifiers 13
       5.4.1. Examples of Other Identifiers Complementing L4S Identifiers .............................................. 13
           5.4.1.1. Inclusion of Additional Traffic with L4S .... 13
           5.4.1.2. Exclusion of Traffic From L4S Treatment .... 14
       5.4.2. Generalized Combination of L4S and Other Identifiers 15
6. L4S Experiments ................................................. 16
7. IANA Considerations ............................................. 16
8. Security Considerations .......................................... 16
9. Acknowledgements ............................................... 17
10. References .................................................... 17
1. Introduction

This specification defines the identifier to be used on IP packets for a new network service called low latency, low loss and scalable throughput (L4S). It is similar to the original (or ‘Classic’) Explicit Congestion Notification (ECN [RFC3168]). ‘Classic’ ECN marking was required 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 ‘Classic’ flow under the same conditions. Nonetheless, the much more frequent control signals and the finer responses to them result in ultra-low queuing delay without compromising link utilization, and low delay is maintained during high load.
An example of a scalable congestion control that would enable the L4S service is Data Center TCP (DCTCP), which until now has been applicable solely to controlled environments like data centres [RFC8257], because it is too aggressive to co-exist with existing TCP. 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 like DCTCP to co-exist with existing traffic, each getting roughly the same flow rate when they compete under similar conditions. Note that a transport such as DCTCP is still not safe to deploy on the Internet unless it satisfies the requirements listed in Section 4. Also note that 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 new L4S identifier is the key piece that enables L4S hosts and L4S network nodes to interwork and distinguishes their traffic from ‘Classic’ traffic. It gives an incremental migration path so that existing ‘Classic’ TCP traffic will be no worse off, but it can be prevented from degrading the ultra-low delay and loss of the new scalable congestion controls. The performance improvement is so great that it is motivating initial deployment of the separate parts of this system.

1.1. 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.

The Diffserv architecture provides Expedited Forwarding [RFC3246], so that low latency traffic can jump the queue of other traffic. However, on access links dedicated to individual sites (homes, small enterprises or mobile devices), often all traffic at any one time will be latency-sensitive. Then Diffserv is of little use. Instead, we need to remove the 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, AQMs 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, AQM was not widely deployed.

More recent state-of-the-art AQMs, 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 sawtoothing rate of TCP will either cause queuing delay to vary or cause the link to be under-utilized. 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. Flow-queuing can isolate one flow from another, but it cannot isolate a TCP flow from the delay variations it inflicts on itself, and it has other problems — it overrides the flow rate decisions of variable rate video applications, it does not recognise the flows within IPSec VPN tunnels and it is relatively expensive to implement.

Latency is not our only concern: It was known when TCP was first developed that it would not scale to high bandwidth-delay products [TCP-CA]. Given regular broadband bit-rates over WAN distances are already [RFC3649] beyond the scaling range of ‘Classic’ TCP Reno, ‘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] cause ‘Classic’ TCP to starve itself, which is why they have been confined to private data centres or research testbeds (until now).

It turns out that a TCP algorithm like DCTCP that solves the latency problem also solves TCP’s scalability problem. The finer sawteeth have low amplitude, so they cause very little queuing delay variation and the number of sawteeth per round trip remains invariant, which maintains constant tight control as flow-rate scales. A supporting paper [DCttH15] 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 the L4S architecture document [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.

Classic service: The ‘Classic’ service is intended for all the behaviours that currently co-exist with TCP Reno (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 Center TCP. But it is also more general--it allows the set of congestion controls with similar scaling properties to DCTCP to evolve (e.g. Relentless TCP [Mathis09] and the L4S variant of SCREAM for real-time media [RFC8298]).

Both Classic and L4S services can cope with a proportion of unresponsive or less-responsive traffic as well, as long as it does not build a queue (e.g. DNS, VoIP, game sync datagrams, etc).

Classic ECN: The original Explicit Congestion Notification (ECN) protocol [RFC3168].

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, both when applied by the network, and when responded to by the sender;

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].

2. Consensus Choice of L4S Packet Identifier: Requirements

This subsection briefly records the process that led to a consensus choice of L4S identifier, selected from all the alternatives in Appendix B.

Ideally, the identifier for packets using the Low Latency, Low Loss, Scalable throughput (L4S) service ought to meet the following requirements:

- it SHOULD survive end-to-end between source and destination applications: across the boundary between host and network, between interconnected networks, and through middleboxes;
- 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 the chosen identifier is unlikely to satisfy all these requirements, particularly given the limited space left in the IP header. Therefore a compromise will be necessary, which is why all the requirements are expressed with the word ‘SHOULD’ not ‘MUST’. Appendix B discusses the pros and cons of the compromises made in various competing identification schemes against the above requirements.

On the basis of this analysis, "ECT(1) and CE codepoints" is the best compromise. Therefore this scheme is defined in detail in the following sections, while Appendix B records the rationale for this decision.

3. L4S Packet Identification at Run-Time

The L4S treatment is an experimental track alternative packet marking treatment [RFC4774] to the classic ECN treatment [RFC3168], which has been updated by [RFC8311] to allow this experiment (amongst others). 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 normally classifies arriving ECT(1) and CE packets for L4S treatment. 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. Prerequisite Transport Layer Behaviour

4.1. Prerequisite 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

In general, a scalable congestion control needs 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
Some transport protocols derived from TCP mimic this behaviour while others report the accurate extent of TCP 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 accurate ECN feedback (AccECN [I-D.ietf-tcpm-accurate-ecn]) by both ends is a prerequisite for scalable congestion control. Therefore, the presence of ECT(1) in the IP headers even in one direction of a TCP connection will imply that both ends support AccECN. 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 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 [I-D.ietf-avtcore-cc-feedback-message]. The presence of ECT(1) implies that both (all) ends of that 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 first IETF QUIC transport [I-D.ietf-quic-transport].

**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 the flow rate is inversely proportional to the proportion of bytes in packets marked with the CE codepoint. This is termed a scalable congestion control, because the average number of control signals (ECN marks) per round trip remains roughly constant for any flow rate. As with all transport behaviours, a detailed specification will need to be defined for each type of transport or application, including the timescale over which the proportionality
is averaged, and control of burstiness. The inverse proportionality requirement above is worded as a ‘SHOULD’ rather than a ‘MUST’ to allow reasonable flexibility when defining these specifications.

Data Center TCP (DCTCP [RFC8257]) and the L4S variant of SCReAM [RFC8298] are examples of a scalable congestion controls.

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.

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. The specification of a particular scalable congestion control MUST describe in detail how it satisfies each requirement:

- A scalable congestion control MUST react to packet loss in a way that will coexist safely with a TCP Reno congestion control [RFC5681] (see Appendix A.1.3 for rationale).
- A scalable congestion control MUST react to ECN marking from a non-L4S but ECN-capable bottleneck in a way that will coexist with a TCP Reno congestion control [RFC5681] (see Appendix A.1.4 for rationale).

Note that a scalable congestion control is not expected to change to setting ECT(0) while it temporarily falls back to coexist with Reno. However an implementer who believes this would be beneficial if fall-back persists, can choose to do so.

- A scalable congestion control MUST reduce or eliminate RTT bias over as wide a range of RTTs as possible, or at least over the typical range of RTTs that will interact in the intended deployment scenario (see Appendix A.1.5 for rationale).
- A scalable congestion control MUST remain responsive to congestion when the RTT is significantly smaller than in the current public Internet (see Appendix A.1.6 for rationale).
- A scalable congestion control MUST detect loss by counting in time-based units, which is scalable, as opposed to counting in units of packets (as in the 3 DupACK rule of traditional TCP), which is not scalable (see Appendix A.1.7 for rationale).

As well as traffic controlled by a scalable congestion control, a reasonable level of smooth unresponsive traffic at a low rate
relative to typical broadband capacities is likely to be acceptable
(see "Safe Unresponsive Traffic" in Section 5.4.1.1.1).

5. Prerequisite Network Node Behaviour

5.1. Prerequisite Classification and Re-Marking Behaviour

A network node that implements the L4S service MUST classify arriving
ECT(1) packets for L4S treatment and, other than in the exceptional
case referred to next, it MUST classify arriving CE packets for L4S
treatment as well. 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.1 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 SHOULD turn off ECN marking, using drop as a congestion
signal until the overload episode has subsided, as recommended for
all AQMs in [RFC7567] (Section 4.2.1), which follows the similar
advice in RFC 3168 (Section 7).

For backward compatibility in uncontrolled environments, a network
node that implements the L4S treatment MUST also implement a classic
AQM treatment. It MUST classify arriving ECT(0) and Not-ECT packets
for treatment by the Classic AQM (see the discussion of the
classifier for the dual-queue coupled AQM in
[I-D.ietf-tsvwg-aqm-dualq-coupled]). Classic treatment means that
the AQM will mark ECT(0) packets under the same conditions as it
would drop Not-ECT packets [RFC3168].

5.2. The Meaning of L4S CE Relative to Drop

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). 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.

This formula ensures that Scalable and Classic flows will converge to roughly equal congestion windows. 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 all TCP-friendly flows.

[I-D.ietf-tsvwg-aqm-dualq-coupled] specifies the essential aspects of an L4S AQM, as well as recommending other aspects. It gives example implementations in appendices.

The term ‘likelihood’ is used above to allow for marking and dropping to be either probabilistic or deterministic. The example AQMs in [I-D.ietf-tsvwg-aqm-dualq-coupled] drop and mark probabilistically, so the drop probability is arranged to be the square of the marking probability. Nonetheless, an alternative AQM that dropped and marked deterministically would be valid, as long as the dropping frequency was proportional to the square of the marking frequency.

Note that, contrary to RFC 3168, a Dual 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, it does mark an ECT(0) packet under the same conditions that it would have dropped a Not-ECT packet.

5.3. Exception for L4S Packet Identification by Network Nodes with Transport-Layer Awareness

To implement the L4S treatment, a network node does not need to identify transport-layer flows. Nonetheless, if an implementer is willing to identify transport-layer flows at a network node, and if the most recent ECT packet in the same flow was ECT(0), the node MAY classify CE packets for classic ECN [RFC3168] treatment. In all other cases, a network node MUST classify all CE packets for L4S treatment. 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 the most recent ECT packet in a flow was ECT(1).

If an implementer uses flow-awareness to classify CE packets, to determine whether the flow is using ECT(0) or ECT(1) it only uses the most recent ECT packet of a flow (this advice will need to be verified as part of L4S experiments). This is because a sender might switch from sending ECT(1) (L4S) packets to sending ECT(0) (Classic)
packets, or back again, in the middle of a transport-layer flow (e.g.
it might manually switch its congestion control module mid-
connection, or it might be deliberately attempting to confuse the
network).

5.4. Interaction of the L4S Identifier with other Identifiers

5.4.1. 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 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. Such non-ECN-based packet
types MUST be safe to mix with L4S traffic without harming the low
latency service, where 'safe' is explained in Section 5.4.1.1.1
below.

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. Therefore, a network element that
implements L4S MAY classify additional packets into the L queue if
they carry certain non-ECN identifiers. For instance:

- addresses of specific applications or hosts configured to be safe
  (but for example cannot set the ECN field for some temporary
  reason);

- certain protocols that are usually lightweight (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.white-tsvwg-nqb]) service classes or equivalent local-use
  DSCPs (see [I-D.briscoe-tsvwg-l4s-diffserv]).
Of course, a packet that carried both the ECT(1) codepoint and a relevant non-ECN identifier would also be classified into the L queue.

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 be). Specifically ECT(1) indicates that the host claims its behaviour satisfies the per-requisite transport requirements in Section 4.

To include additional traffic with L4S, a network element only reads identifiers such as those itemized above. It MUST NOT alter these non-ECN identifiers.

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 data 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 data rate from packet to packet stays well below a typical broadband access rate.

This is a vague but useful definition, because it encompasses many low latency applications of interest, such as DNS, voice, game sync packets, RPC, ACKs, keep-alives, etc.

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 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 operator 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.
The operator MUST NOT alter the end-to-end L4S ECN identifier, because its 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.

5.4.2. Generalized Combination of L4S and Other Identifiers

L4S concerns low latency, which it can provide for all traffic without differentiation and without 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 examples above were 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 by far 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 with a low latency and a classic variant.

In the first case, if we assume that there are no other PHBs except the DualQ, if a packet carries ECT(1) or CE, a 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 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 for all L4S traffic 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 any 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 (Recommended-standard-use or Local-use)
   B. Excluding Certain Traffic from the L Queue (Local-use only)

2. Identifiers to place L4S classification in a PHB Hierarchy (Recommended-standard-use or Local-use)
   A. PHBs Before L4S ECN Classification
   B. PHBs After L4S ECN Classification

6. L4S Experiments

[I-D.ietf-tsvwg-aqm-dualq-coupled] sets operational and management requirements for experiments with DualQ Coupled AQMs. 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 [RFC5706] provides a helpful checklist.

7. IANA Considerations

This specification contains no IANA considerations.

8. Security Considerations

Approaches to assure the integrity of signals using the new identifier are introduced in Appendix C.1.

The requirement to detect loss in time units prevents the ACK-splitting attacks described in [Savage-TCP].
9. Acknowledgements

Thanks to Richard Scheffenegger, John Leslie, David Taeh, Jonathan Morton, Gorry 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, Gorry Fairhurst and Brian Carpenter for providing help and reviewing this draft and to Ingemar Johansson for reviewing and providing substantial text. 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). Bob Briscoe was also part-funded by the Research Council of Norway through the TimeIn project. The views expressed here are solely those of the authors.

10. References

10.1. Normative References


10.2. Informative References


[I-D.briscoe-tsvwg-l4s-diffserv] Briscoe, B., "Interactions between Low Latency, Low Loss, Scalable Throughput (L4S) and Differentiated Services", draft-briscoe-tsvwg-l4s-diffserv-02 (work in progress), November 2018.


[I-D.ietf-tcpm-rack]

[I-D.ietf-tsvwg-aqm-dualq-coupled]
Schepper, K., Briscoe, B., and G. White, "DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)", draft-ietf-tsvwg-aqm-dualq-coupled-09 (work in progress), July 2019.

[I-D.ietf-tsvwg-ecn-encap-guidelines]

[I-D.ietf-tsvwg-l4s-arch]

[I-D.sridharan-tcpm-ctcp]

[I-D.stewart-tsvwg-sctpecn]

[I-D.white-tsvwg-nqb]

[Mathis09]
[Paced-Chirping]
Missund, J., "Rapid Acceleration in TCP Prague", Masters Thesis, May 2018,

[QV] Briscoe, B. and P. Hurtig, "Up to Speed with Queue View", RITE Technical Report D2.3; Appendix C.2, August 2015,


<https://www.rfc-editor.org/info/rfc2983>.


[Savage-TCP]


Appendix A. 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 necessary safety improvements (requirements) this appendix also includes preferable performance improvements (optimizations).
These recommendations have become known as the Prague L4S Requirements, because they were originally identified at an ad hoc meeting during IETF-94 in Prague [TCPPrague]. The wording has been generalized to apply to all scalable congestion controls, not just TCP congestion control specifically. They were originally called the ‘TCP Prague Requirements’, but they are not solely applicable to TCP, so the name has been generalized, and TCP Prague is now used for a specific implementation of the requirements.

DCTCP [RFC8257] is currently 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 it is likely that most of these requirements equally apply to other scalable congestion controls.

A.1. 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.

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.

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 an experimental change to the TCP wire protocol to satisfy these requirements. Most other transport protocols already satisfy this requirement.
A.1.3. Fall back to Reno-friendly congestion control on packet loss

Description: A scalable congestion control needs to react to packet loss in a way that will coexist safely with a TCP Reno congestion control [RFC5681].

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 with this requirement 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 bulk transport like TCP. Therefore, the particular classic congestion control behaviour to fall back on will need to be part of the congestion control specification of the relevant transport. In the particular case of DCTCP, the current DCTCP specification states that "It is RECOMMENDED that an implementation deal with loss episodes in the same way as conventional TCP." For safe deployment of a scalable congestion control in the public Internet, the above requirement would need to be defined as a "MUST".

Packet loss might (rarely) occur in the case that the bottleneck is L4S capable. In this case, the sender may receive a high number of packets marked with the CE bit set and also experience a loss. Current DCTCP implementations 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). 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.1.4. Fall back to Reno-friendly congestion control on classic ECN bottlenecks

Description: A scalable congestion control needs to react to ECN marking from a non-L4S but ECN-capable bottleneck in a way that will coexist with a TCP Reno congestion control [RFC5681].

Motivation: Similarly to the requirement in Appendix A.1.3, this requirement is a safety condition to ensure a scalable congestion
control behaves properly when it builds a queue at a network bottleneck that has not been upgraded to support L4S. On detecting classic ECN marking (see below), a scalable congestion control will need to fall back to classic congestion control behaviour. If it does not comply with this requirement it could starve classic traffic.

It would take time for endpoints to distinguish classic and L4S ECN marking. An increase in queuing delay or in delay variation would be a tell-tale sign, but it is not yet clear where a line would be drawn between the two behaviours. It might be possible to cache what was learned about the path to help subsequent attempts to detect the type of marking.

A.1.5. Reduce RTT dependence

Description: A scalable congestion control needs to reduce or eliminate RTT bias over as wide a range of RTTs as possible, or at least over the typical range of RTTs that will interact in the intended deployment scenario.

Motivation: Classic TCP’s throughput 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 TCP has never allowed a very low RTT path to exist because it induces a large queue. For instance, consider two paths with base RTT 1ms and 100ms. If Classic TCP induces a 100ms queue, it turns these RTTs into 101ms and 200ms leading to a throughput ratio of about 2:1. Whereas if a Scalable TCP induces only a 1ms 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.

A.1.6. Scaling down to fractional congestion windows

Description: A scalable congestion control needs to remain responsive to congestion when RTTs are significantly smaller than in the current public Internet.

Motivation: As currently specified, the minimum required congestion window of TCP (and its derivatives) is set to 2 maximum segment sizes (MSS) (see equation (4) in [RFC5681]). Once the congestion window reaches this minimum, all current TCP algorithms become unresponsive to congestion signals. No matter how much drop or ECN marking, the congestion window no longer reduces. Instead, TCP forces the queue to grow, overriding any AQM and increasing queuing delay.
L4S mechanisms significantly reduce queueing delay so, over the same path, the RTT becomes lower. Then this problem becomes surprisingly common [TCP-sub-mss-w]. 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 MSS if the RTT is 6ms or less, which is quite common when accessing a nearby data centre. So, any more than two such parallel TCP flows will become unresponsive and increase queuing delay.

Unless scalable congestion controls are required to comply with this requirement from the start, they will frequently become unresponsive, negating the low latency benefit of L4S, for themselves and for others. One possible sub-MSS window mechanism is described in [TCP-sub-mss-w], and other approaches are likely to be feasible.

A.1.7. Measuring Reordering Tolerance in Time Units

Description: A scalable congestion control needs to detect loss by counting in time-based units, which is scalable, rather than counting in units of packets, which is not.

Motivation: If it is known that all L4S senders using a link obey this rule, then link technologies that support L4S can remove the head-of-line blocking delay they have to introduce while trying to keep packets in tight order to avoid triggering loss detection based on counting packets.

End-systems cannot know whether a missing packet is due to loss or reordering, except in hindsight - if it appears later. If senders deem that loss has occurred by counting reordered packets (e.g. the 3 Duplicate ACK rule of Classic TCP), the time over which the network has to keep packets in order scales down as packet rates scale up over the years. In contrast, if senders allow a reordering window in time-based units before they deem there has been a loss, the time over which the network has to keep packets in order stays constant.

The potential benefit for links comes in two parts: i) switching the unit from packet count to time-based; ii) potentially relaxing the amount of time available for re-ordering. The initial switch to time-based units offers no immediate benefit, but as the years progress it stops the reordering requirement becoming tighter. The secondary relaxation might be possible where some transport protocols find they can tolerate more re-ordering (e.g. more than the 3 DupACK rule, perhaps because it was reasonable when packet rates were low, but it is now far too tight).
Tolerance of reordering over a small duration could allow parallel (e.g. bonded-channel) link technologies to relax their need to deliver packets strictly in order. Such links typically give arriving packets a link-level sequence number and introduce delay while buffering packets at the receiving end until they can be delivered in the same order. For radio links, this delay usually includes the time allowed for link-layer retransmissions.

Note, however, that a link technology will only be able to relax its ordering requirement if it is certain that it will not degrade the transport /most/ sensitive to reordering that might use the link. Also note that in some controlled environments no reordering is tolerated by some transports (e.g. RoCEv2 used for RDMA), therefore a switch to time-based units could not be exploited to relax reordering.

For receivers that need their packets in order, it would seem that relaxing network ordering would simply shift this reordering delay from the network to the receiver. However, that is not true in the general case because links generally do not recognize transport layer flows and often cannot even see application layer streams within the flows (as in SCTP, HTTP/2 or QUIC). So a link will often be holding back packets from one flow or stream while waiting for those from another. Relaxing strict ordering in the network will remove this head-of-line blocking delay. (ToDo: this is being quantified experimentally - will need to add the figures here.)

Classic TCP implementations are switching over to the time-based approach of RACK (Recent ACKnowledgements [I-D.ietf-tcpm-rack]). However, it will be many years (decades?) before networks no longer have to allow for the presence of traditional TCP senders still using the 3 DupACK rule. This specification (Section 4.3) says that senders are not entitled to identify packets as L4S in the IP/ECN field unless they use the time-based approach. Then networks that identify L4S traffic separately (e.g. using [I-D.ietf-tsvwg-aqm-dualq-coupled]) can know for certain that all L4S traffic is using the scalable time-based approach.

This will allow networks to remove the head-of-line blocking delay of resequencing straight away, but only for L4S traffic. Classic traffic will have to wait for many years until incremental deployment of RACK has become near-universal. Nonetheless, experience with RACK will determine how much reordering tolerance networks will be reasonable for L4S traffic.

Performance Optimization as well as Safety Improvement: The delay benefit would be lost if any L4S sender did not follow the time-based approach. Therefore, the time-based approach is made a normative
requirement (a necessary safety improvement). Nonetheless, the time-based approach also enables a throughput benefit that a flow can enjoy independently of others (a performance optimization), explained next.

Given the requirement for a scalable congestion control to fall-back to Reno or Cubic on a loss (see Appendix A.1.3), it is important that a scalable congestion control does not deem that a loss has occurred too soon. If, later within the same round trip, an out-of-order acknowledgement fills the gap, the sender would have halved its rate spuriously (as well as retransmitting spuriously). With a RACK-like approach, allowing longer before a loss is deemed to have occurred maintains higher throughput in the presence of reordering {ToDo: Quantify this statement}.

On the other hand, it is also important not to wait too long before deeming that a gap is due to a loss (termed a long reordering window), otherwise loss recovery would be slow.

The speed of loss recovery is much more significant for short flows than long, therefore a good compromise would 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 the approach adopted by TCP RACK (Recent ACKnowledgements) [I-D.ietf-tcpm-rack] and recommended for all L4S senders, whether using TCP or another transport protocol.

The requirement to detect 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 TCP Control Packets and Retransmissions

Description: This item only concerns TCP and its derivatives (e.g. SCTP), because the original specification of ECN for TCP precluded the use of ECN on control packets and retransmissions. 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 derivatives of TCP, e.g. SCTP.

Motivation: 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. [I-D.ietf-tcpm-generalized-ecn] counters each argument in RFC 3168 in turn, showing it was over-cautious. Instead it 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 is itself a
prerequisite for using a scalable congestion control).

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
traditional additive increase of 1 MSS 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, DCTCP uses the traditional TCP 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. In the steady state, DCTCP induces
about 2 ECN marks per round trip, so it should be possible to quickly
detect when these signals have disappeared and seek available
capacity more rapidly. It will of course be necessary to minimize
the impact on other flows (classic and scalable).

TCP Cubic was designed to solve this problem, but as flow rates have
continued to increase, the delay accelerating into available capacity
has become prohibitive. For instance, with RTT=20 ms, to increase
flow rate from 100Mb/s to 200Mb/s Cubic takes between 50 and 100
round trips. Every 8x increase in flow rate leads to 2x more
acceleration delay.

A.2.3. Faster Convergence at Flow Start

Description: Particularly when a flow starts, scalable congestion
controls need to converge (reach their steady-state share of the
capacity) at least as fast as classic TCP and preferably faster.
This does not just affect TCP Prague, but also 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 classic
TCP 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 regular TCP is
even less favourable [Paced-Chirping]). It is exacerbated by the
typically greater mismatch between the link rate of the sending host
and typical Internet access bottlenecks, in combination with the
shallow ECN marking threshold needed for L4S. This problem is
detrimental in general, but would particularly harm the performance
of short flows relative to classic TCP.

Appendix B. Alternative Identifiers

This appendix is informative, not normative. It records the pros and
cons of various alternative ways to identify L4S packets to record
the rationale for the choice of ECT(1) (Appendix B.1) as the L4S
identifier. At the end, Appendix B.6 summarises the distinguishing
features of the leading alternatives. It is intended to supplement,
not replace the detailed text.

The leading solutions all use the ECN field, sometimes in combination
with the Diffserv field. Both the ECN and Diffserv fields have the
additional advantage that they are no different in either IPv4 or
IPv6. A couple of alternatives that use other fields are mentioned
at the end, but it is quickly explained why they are not serious
contenders.

B.1. ECT(1) and CE codepoints

Definition:

Packets with ECT(1) and conditionally packets with CE would
signify L4S semantics as an alternative to the semantics of
classic ECN [RFC3168], specifically:

* The ECT(1) codepoint would signify that the packet was sent by
  an L4S-capable sender;

* Given shortage of codepoints, both L4S and classic ECN sides of
  an AQM would have to use the same CE codepoint to indicate that
  a packet had 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 would be 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 is intended to 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 ECN in an AQM acting in a buffer below the IP layer [I-D.ietf-tsvwg-ecn-encap-guidelines]. In such cases, the L4S service would have to drop rather than mark frames even though they might contain an ECN-capable packet. However, such cases would be unusual.

Risk of reordering classic CE packets: 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 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 a non-L4S AQM 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 50 out of sequence would trigger a spurious retransmission, but there is no spurious retransmission here, 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 recommended TCP ($N=3$) spurious retransmissions will be unlikely for all the above reasons. As RACK [I-D.ietf-tcpm-rack] 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. Admittedly TCP reduces its congestion window when it deems there has been a loss, but even this can be recovered once the sender detects that the retransmission was spurious.

Non-L4S service for control packets: The classic ECN RFCs [RFC3168] and [RFC5562] require a sender to clear the ECN field to Not-ECT for retransmissions and certain control packets specifically pure ACKs, window probes and SYNs. When L4S packets are classified by the ECN field alone, these 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 re-ordering (because retransmissions are already out of order, and the control packets carry no data). However, it would make critical control packets more vulnerable to loss and delay. To address this problem, [I-D.ietf-tcpm-generalized-ecn] proposes an experiment in which all TCP control packets and retransmissions are ECN-capable as long as ECN feedback is available.
Pros:

Should work e2e: The ECN field generally works end-to-end across the Internet. Unlike the DSCP, the setting of the ECN field is at least forwarded unchanged by networks that do not support ECN, and networks rarely clear it to zero;

Should work in tunnels: Unlike Diffserv, ECN is defined to always work across tunnels. However, tunnels do not always implement ECN processing as they should do, particularly because IPsec tunnels were defined differently for a few years.

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.

B.2. ECN Plus a Diffserv Codepoint (DSCP)

Definition:

For packets with a defined DSCP, all codepoints of the ECN field (except Not-ECT) would signify alternative L4S semantics to those for classic ECN [RFC3168], specifically:

* The L4S DSCP would signify that the packet came from an L4S-capable sender;

* ECT(0) and ECT(1) would both signify that the packet was travelling between transport endpoints that were both ECN-capable;

* CE would signify that the packet had been marked by an AQM implementing the L4S service.

Use of a DSCP is the only approach for alternative ECN semantics given as an example in [RFC4774]. However, it was perhaps considered more for controlled environments than new end-to-end services;

Cons:

Consumes DSCP pairs: A DSCP is obviously not orthogonal to Diffserv. Therefore, wherever the L4S service is applied to multiple Diffserv scheduling behaviours, it would be necessary to replace each DSCP with a pair of DSCPs.
Uses critical lower-layer header space: The resulting increased number of DSCPs might be hard to support for some lower layer technologies, e.g. 802.1p and MPLS both offer only 3-bits for a maximum of 8 traffic class identifiers. Although L4S should reduce and possibly remove the need for some DSCPs intended for differentiated queuing delay, it will not remove the need for Diffserv entirely, because DiffServ is also used to allocate bandwidth, e.g. by prioritising some classes of traffic over others when traffic exceeds available capacity.

Not end-to-end (host-network): Very few networks honour a DSCP set by a host. Typically a network will zero (bleach) the DiffServ field from all hosts. Sometimes networks will attempt to identify applications by some form of packet inspection and, based on network policy, they will set the DSCP considered appropriate for the identified application. Network-based application identification might use some combination of protocol ID, port numbers(s), application layer protocol headers, IP address(es), VLAN ID(s) and even packet timing.

Not end-to-end (network-network): Very few networks honour a DSCP received from a neighbouring network. Typically a network will zero (bleach) the DiffServ field from all neighbouring networks at an interconnection point. Sometimes bilateral arrangements are made between networks, such that the receiving network remarks some DSCPs to those it uses for roughly equivalent services. The likelihood that a DSCP will be bleached or ignored depends on the type of DSCP:

Local-use DSCP: These tend to be used to implement application-specific network policies, but a bilateral arrangement to remark certain DSCPs is often applied to DSCPs in the local-use range simply because it is easier not to change all of a network’s internal configurations when a new arrangement is made with a neighbour;

Recommended standard DSCP: These do not tend to be honoured across network interconnections more than local-use DSCPs. However, if two networks decide to honour certain of each other’s DSCPs, the reconfiguration is a little easier if both of their globally recognised services are already represented by the relevant recommended standard DSCPs.

Note that today a recommended standard DSCP gives little more assurance of end-to-end service than a local-use DSCP. In future the range recommended as standard might give more assurance of end-to-end service than local-use, but it is
unlikely that either assurance will be high, particularly given the hosts are included in the end-to-end path.

Not all tunnels: Diffserv codepoints are often not propagated to the outer header when a packet is encapsulated by a tunnel header. DSCPs are propagated to the outer of uniform mode tunnels, but not pipe mode [RFC2983], and pipe mode is fairly common.

ECN hard in some lower layers: Because this approach uses both the Diffserv and ECN fields, an AQM will only work at a lower layer if both can be supported. If individual network operators wished to deploy an AQM at a lower layer, they would usually propagate an IP Diffserv codepoint to the lower layer, using for example IEEE 802.1p. However, the ECN capability is harder to propagate down to lower layers because few lower layers support it.

Pros:

Could migrate to e2e: If all usage of classic ECN migrates to usage of L4S, the DSCP would become redundant, and the ECN capability alone could eventually identify L4S packets without the interconnection problems of Diffserv detailed above, and without having permanently consumed more than one codepoint in the IP header. Although the DSCP does not generally function as an end-to-end identifier (see above), it could be used initially by individual ISPs to introduce the L4S service for their own locally generated traffic;

B.3. ECN capability alone

Definition:

This approach uses ECN capability alone as the L4S identifier. It is only feasible if classic ECN is not widely deployed. The specific definition of codepoints would be:

* Any ECN codepoint other than Not-ECT would signify an L4S-capable sender;

* ECN codepoints would not be used for classic [RFC3168] ECN, and the classic network service would only be used for Not-ECT packets.

This approach would only be feasible if

A. it was generally agreed that there was little chance of any classic [RFC3168] ECN deployment in any network nodes;
it was generally agreed that there was little chance of any
client devices being deployed with classic [RFC3168] TCP-ECN
on by default (note that classic TCP-ECN is already on-by-
default on many servers);

for TCP connections, developers of client OSs would all have
to agree not to encourage further deployment of classic ECN.
Specifically, at the start of a TCP connection classic ECN
could be disabled during negotiation of the ECN capability:

+ an L4S-capable host would have to disable ECN if the
  corresponding host did not support accurate ECN feedback
  [RFC7560], which is a prerequisite for the L4S service;

+ developers of operating systems for user devices would only
  enable ECN by default for TCP once the stack implemented
  L4S and accurate ECN feedback [RFC7560] including
  requesting accurate ECN feedback by default.

Cons:

Near-infeasible deployment constraints: The constraints for
deployment above represent a highly unlikely, but not completely
impossible, set of circumstances. If, despite the above measures,
a pair of hosts did negotiate to use classic ECN, their packets
would be classified into the same queue as L4S traffic, and if
they had to compete with a long-running L4S flow they would get a
very small capacity share;

ECN hard in some lower layers: See the same issue with "ECT(1) and
CE codepoints" (Appendix B.1);

Non-L4S service for control packets: See the same issue with "ECT(1)
and CE codepoints" (Appendix B.1).

Pros:

Consumes no additional codepoints: The ECT(1) codepoint and all
spare DiffServ codepoints would remain available for future use;

Should work e2e: As with "ECT(1) and CE codepoints" (Appendix B.1);

Should work in tunnels: As with "ECT(1) and CE codepoints"
(Appendix B.1).
B.4. Protocol ID

It has been suggested that a new ID in the IPv4 Protocol field or the IPv6 Next Header field could identify L4S packets. However this approach is ruled out by numerous problems:

- A new protocol ID would need to be paired with the old one for each transport (TCP, SCTP, UDP, etc.);
- In IPv6, there can be a sequence of Next Header fields, and it would not be obvious which one would be expected to identify a network service like L4S;
- A new protocol ID would rarely provide an end-to-end service, because it is well-known that new protocol IDs are often blocked by numerous types of middlebox;
- The approach is not a solution for AQMs below the IP layer;

B.5. Source or destination addressing

Locally, a network operator could arrange for L4S service to be applied based on source or destination addressing, e.g. packets from its own data centre and/or CDN hosts, packets to its business customers, etc. It could use addressing at any layer, e.g. IP addresses, MAC addresses, VLAN IDs, etc. Although addressing might be a useful tactical approach for a single ISP, it would not be a feasible approach to identify an end-to-end service like L4S. Even for a single ISP, it would require packet classifiers in buffers to be dependent on changing topology and address allocation decisions elsewhere in the network. Therefore this approach is not a feasible solution.

B.6. Summary: Merits of Alternative Identifiers

Table 1 provides a very high level summary of the pros and cons detailed against the schemes described respectively in Appendix B.2, Appendix B.3 and Appendix B.1, for six issues that set them apart.
Note 1: Only feasible if classic ECN is obsolete.

Table 1: Comparison of the Merits of Three Alternative Identifiers

The schemes are scored based on both their capabilities now ('initial') and in the long term ('eventual'). The 'ECN' scheme shares the 'eventual' scores of the 'ECT(1) + CE' scheme. The scores are one of 'N, O, Y', meaning 'Poor', 'Ordinary', 'Good' respectively. The same scores are aligned vertically to aid the eye. A score of '?' in one of the positions means that this approach might optimistically become this good, given sufficient effort. The table summarises the text and is not meant to be understandable without having read the text.

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] as been reclassified as historic, partly because other ways have been developed to protect TCP feedback integrity [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 [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 any of the L4S service identifiers proposed in Appendix B.

Authors’ Addresses

Koen De Schepper
Nokia Bell Labs
Antwerp
Belgium

Email: koen.de_schepper@nokia.com
URI: https://www.bell-labs.com/usr/koen.de_schepper

Bob Briscoe (editor)
CableLabs
UK

Email: ietf@bobbriscoe.net
URI: http://bobbriscoe.net/
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.
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)
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:

- 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;
- each sliding window FEC Scheme is associated to a specific FEC Encoding ID subject to IANA registration, just like block FEC Schemes;
- 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 < code rate <= 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:

- 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;

- 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:

- [RFC6363] section 8 does not detail any congestion control mechanism, but only provides high level normative requirements;
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);

- 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
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
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:

- MUST define the relationships between ADUs and the associated source symbols (mapping);
- 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;
- 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:

- Organisation: Inria

- 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.

- Maturity: the basic FECFRAME maturity is "production", the FECFRAME extension maturity is "under progress".

- 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.

- Licensing: proprietary.

- Implementation experience: maximum.

- Information update date: March 2018.

- 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]:

- 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, Gorry 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


14.2. Informative References


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:

- 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;

- 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:

- at the source flows level: real-time constraints can limit the total time source symbols can remain in the encoding window;

- 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;

- 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

Ali Begen
Networked Media
Konya
Turkey

EMail: ali.begen@networked.media
Abstract

This document describes the L4S architecture for the provision of a new Internet service that could eventually replace best efforts for all traffic: Low Latency, Low Loss, Scalable throughput (L4S). It is becoming common for _all_ (or most) applications being run by a user at any one time to require low latency. However, the only solution the IETF can offer for ultra-low queuing delay is Diffserv, which only favours a minority of packets at the expense of others. In extensive testing the new L4S service keeps average queuing delay under a millisecond for _all_ applications even under very heavy load, without sacrificing utilization; and it keeps congestion loss to zero. It is becoming widely recognized that adding more access capacity gives diminishing returns, because latency is becoming the critical problem. Even with a high capacity broadband access, the reduced latency of L4S remarkably and consistently improves performance under load for applications such as interactive video, conversational video, voice, Web, gaming, instant messaging, remote desktop and cloud-based apps (even when all being used at once over the same access link). The insight is that the root cause of queuing delay is in TCP, not in the queue. By fixing the sending TCP (and other transports) queuing latency becomes so much better than today that operators will want to deploy the network part of L4S to enable new products and services. Further, the network part is simple to deploy - incrementally with zero-config. Both parts, sender and network, ensure coexistence with other legacy traffic. At the same time L4S solves the long-recognized problem with the future scalability of TCP throughput.

This document describes the L4S architecture, briefly describing the different components and how the work together to provide the aforementioned enhanced Internet service.
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. L4S Architecture Overview ............................... 4
3. Terminology .............................................. 6
4. L4S Architecture Components ............................. 7
5. Rationale ................................................. 10
   5.1. Why These Primary Components? ..................... 10
   5.2. Why Not Alternative Approaches? ................... 12
6. Applicability ............................................. 15
   6.1. Applications ...................................... 15
   6.2. Use Cases ......................................... 16
   6.3. Deployment Considerations ......................... 17
      6.3.1. Deployment Topology ............................ 18
      6.3.2. Deployment Sequences ........................... 19
      6.3.3. L4S Flow but Non-L4S Bottleneck ............... 21

Briscoe, et al. Expires January 9, 2020
1. Introduction

It is increasingly common for _all_ of a user’s applications at any one time to require 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 common. During a long-running flow, even with state-of-the-art active queue management (AQM), the base speed-of-light path delay roughly doubles. 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. Differentiated services (DiffServ) offers Expedited Forwarding (EF [RFC3246]) for some packets at the expense of others, but this is not sufficient when all (or most) of a user’s applications require low latency.

Therefore, the goal is an Internet service with ultra-Low queueing Latency, ultra-Low Loss and Scalable throughput (L4S) — for _all_ traffic. A service for all traffic will need none of the configuration or management baggage (traffic policing, traffic contracts) associated with favouring some packets over others. This document describes the L4S architecture for achieving that goal.

It must be said that queuing delay only degrades performance infrequently [Hohlfeld14]. It only 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 flow (e.g. interactive video). At these times, the performance improvement from L4S must be so remarkable 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 call this family of congestion controls ‘Classic’ TCP. It has been demonstrated that if the sending host replaces Classic TCP 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 stunningly improved. For instance, queuing delay under heavy load with the example DCTCP/DualQ solution cited below is roughly 1 millisecond (1 to 2 ms) at the 99th percentile without losing link utilization. This compares with 5 to 20 ms on _average_ with a Classic TCP and current state-of-the-art AQMs such as fq_CoDel [RFC8290] or PIE [RFC8033] and about 20-30 ms at the 99th percentile. Also, with a Classic TCP, 5 ms of queuing is usually only possible by losing some utilization.

It has been convincingly demonstrated [DCTtH15] that it is possible to deploy such an L4S service alongside the existing best efforts service so that all of a user’s applications can shift to it when their 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 a single network node should give nearly all the benefit. The L4S approach also requires component mechanisms at the endpoints to fulfill its goal. This document presents the L4S architecture, by describing the different components and how they interact to provide the scalable low-latency, low-loss, Internet service.

2. L4S Architecture Overview

There are three main components to the L4S architecture (illustrated in Figure 1):

1) Network: L4S traffic needs to be isolated from the queuing latency of Classic traffic. However, the two should be able to freely share a common pool of capacity. This is because there is no way to predict how many flows at any one time might use each
service and capacity in access networks is too scarce to partition into two. The Dual Queue Coupled AQM [I-D.ietf-tsvwg-aqm-dualq-coupled] was developed as a minimal complexity solution to this problem. The two queues appear to be separated by a ‘semi-permeable’ membrane that partitions latency but not bandwidth (explained later).

Per-flow queuing such as in [RFC8290] could be used (see Section 4), but it partitions both latency and bandwidth between every end-to-end flow. So it is rather overkill, which brings disadvantages (see Section 5.2), not least that large number of queues are needed when two are sufficient.

2) Protocol: A host needs to distinguish L4S and Classic packets with an identifier so that the network can classify them into their separate treatments. [I-D.ietf-tsvwg-ecn-l4s-id] considers various alternative identifiers, and concludes that all alternatives involve compromises, but the ECT(1) and CE codepoints of the ECN field represent a workable solution.

3) Host: Scalable congestion controls already exist. They solve the scaling problem with TCP that was first pointed out in [RFC3649]. The one 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 ‘works’ well over the public Internet, most implementations lack certain safety features that will be necessary once it is used outside controlled environments like data centres (see later). A similar scalable congestion control will also need to be transplanted into protocols other than TCP (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 and an L4S variant of the RMCAT SCReAM controller [RFC8298].
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]. 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. COMMENT: Since this will be an information document, This should be removed.

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

Low-Latency, Low-Loss and Scalable (L4S) service: The ‘L4S’ service is intended for traffic from scalable TCP algorithms such as Data Center TCP. But it is also more general—it will allow a set of congestion controls with similar scaling properties to DCTCP (e.g. Relentless [Mathis09]) 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).

Scalable Congestion Control: A congestion control where the packet flow rate per round trip (the window) is inversely proportional to the level (probability) of congestion signals. Then, as flow rate scales, the number of congestion signals per round trip remains invariant, maintaining the same degree of control. For instance,
DCTCP averages 2 congestion signals per round-trip whatever the flow rate.

Classic Congestion Control: A congestion control with a flow rate that can co-exist with standard TCP Reno [RFC5681] without starvation. With Classic congestion controls, as capacity increases enabling higher flow rates, the number of round trips between congestion signals (losses or ECN marks) rises in proportion to the flow rate. So control of queuing and/or utilization becomes very slack. For instance, with 1500 B packets and an RTT of 18 ms, as TCP Reno flow rate increases from 2 to 100 Mb/s the number of round trips between congestion signals rises proportionately, from 2 to 100.

The default congestion control in Linux (TCP Cubic) is Reno-compatible for most Internet access scenarios expected for some years. For instance, with a typical domestic round-trip time (RTT) of 18ms, TCP Cubic only switches out of Reno-compatibility mode once the flow rate approaches 1 Gb/s. For a typical data centre RTT of 1 ms, the switch-over point is theoretically 1.3 Tb/s. However, with a less common transcontinental RTT of 100 ms, it only remains Reno-compatible up to 13 Mb/s. All examples assume 1,500 B packets.

Classic ECN: The original proposed standard 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.

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.

4. L4S Architecture Components

The L4S architecture is composed of the following elements.

Protocols: The L4S architecture encompasses the two identifier changes (an unassignment and an assignment) and optional further identifiers:

a. An essential aspect of a scalable congestion control is the use of explicit congestion signals rather than losses, because the signals need to be sent immediately and frequently. ‘Classic’ ECN [RFC3168] requires an ECN signal to be treated the same as a 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 different meaning for ECN:
* much more frequent signals--too often to use drops;
* immediately tracking every fluctuation of the queue--too soon to commit to dropping packets.

So the standards track [RFC3168] has had to be updated to allow L4S packets to depart from the 'same as 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] recommends ECT(1) is used as the identifier to classify L4S packets into a separate treatment from Classic packets. This satisfies the requirements 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. [I-D.ietf-tsvwg-ecn-l4s-id] explains why 5 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 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.white-tsvwg-nqb]), or operator-specific identifiers.

Network components: The L4S architecture encompasses either dual-queue or per-flow queue solutions:

a. The Dual Queue Coupled AQM has been specified as generically as possible [I-D.ietf-tsvwg-aqm-dualq-coupled] as a ‘semi-permeable’ membrane without specifying the particular AQMs to use in the two queues. Informational appendices of the draft are provided for pseudocode examples of different possible AQM approaches. The aim is for designers to be free to implement diverse ideas. So the brief normative body of the draft only specifies the minimum
constraints an AQM needs to comply with to ensure that the L4S and Classic services will coexist. The core idea is the tension between the scheduler’s prioritization of L4S over Classic and the coupling from the Classic to the L4S AQM. The L4S AQM derives its level of ECN marking from the maximum of the congestion levels in both queues. So L4S flows leave enough space between their packets for Classic flows, as if they were all the same type of TCP, all sharing one FIFO queue.

Initially a zero-config variant of RED called Curvy RED was implemented, tested and documented. Then, a variant of PIE called DualPI2 (pronounced Dual PI Squared) [PI2] was implemented and found to perform better than Curvy RED over a wide range of conditions, so it was documented in another appendix of [I-D.ietf-tsvwg-aqm-dualq-coupled].

b. A scheduler with per-flow queues can be used for L4S. It would be simple to modify an existing design such as FQ-CoDel or FQ-PIE, although this has not been implemented and evaluated because the goal of the original proponents of L4S was to avoid per-flow scheduling.

The idea would be to implement two AQMs (Classic and Scalable) and switch each per-flow queue to use an instance of the appropriate AQM for the flow, based on the ECN codepoints of the packets. Flows of non-ECN or ECT(0) packets would use a Classic AQM such as CoDel or PIE, while flows of ECT(1) packets without any ECT(0) packets would use a simple shallow threshold AQM with immediate (unsmoothed) marking. The FQ scheduler might work as is, because it is likely that L4S flows would be continually categorized as ‘new’ flows. However, this presumption has not been tested under a wide range of conditions. A variant of FQ-CoDel already exists that adapts to a shallower threshold AQM for ECN-capable packets.

Host mechanisms: The L4S architecture includes a number of mechanisms in the end host that we enumerate next:

a. Data Center TCP is the most widely used example of a scalable congestion control. It has been documented as an informational record of the protocol currently in use [RFC8257]. It will be necessary to define a number of safety features for a variant usable on the public Internet. A draft list of these, known as the TCP Prague requirements, has been drawn up (see Appendix A of [I-D.ietf-tsvwg-ecn-l4s-id]). The list also includes some optional performance improvements.
b. Transport protocols other than TCP use various congestion controls designed to be friendly with Classic TCP. Before they can use the L4S service, it will be necessary to implement scalable variants of each of these congestion control behaviours. The following standards track RFCs currently define these protocols: ECN in TCP [RFC3168], in SCTP [RFC4960], in RTP [RFC6679], and in DCCP [RFC4340]. Not all are in widespread use, but those that are will eventually need to be updated to allow a different congestion response, which they will have to indicate by using the ECT(1) codepoint. Scalable variants are under consideration for some new transport protocols that are themselves under development, e.g. QUIC [I-D.ietf-quic-transport] and certain real-time media congestion avoidance techniques (RMCAT) protocols.

c. ECN feedback is sufficient for L4S in some transport protocols (RTCP, DCCP) but not others:

* For the case of TCP, the feedback protocol for ECN embeds the assumption from Classic ECN that an ECN mark is the same as 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. A fuller specification has been proposed [I-D.stewart-tsvwg-sctpecn], which would need to be implemented and deployed before SCTCP could support L4S.

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 a useful signal (more would reduce delay) and an impairment (less would reduce delay):

* 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. Whereas state-of-the-art AQMs have to introduce very high packet drop at high load to keep the queue short. Further, when using ECN, TCP’s sawtooth reduction can be smaller and therefore return to the
operating point more often, without worrying that this causes more signals (one at the top of each smaller sawtooth). The consequent smaller amplitude sawteeth fit between a very shallow marking threshold and an empty queue, so delay variation can be very low, without risk of under-utilization.

* Explicit congestion signals can be sent immediately to track fluctuations of the queue. L4S shifts smoothing from the network (which doesn’t know the round trip times of all the flows) to the host (which knows its own round trip time). Previously, the network had to smooth to keep a worst-case round trip stable, delaying congestion signals by 100–200ms.

All the above makes it clear that explicit congestion signalling is only advantageous for latency if it does not have to be considered ‘the same as’ drop (as was required with Classic ECN [RFC3168]). Therefore, in a DualQ AQM, the L4S queue uses a new L4S variant of ECN that is not equivalent to drop [I-D.ietf-tsvwg-ecn-l4s-id], while the Classic queue uses either classic ECN [RFC3168] or drop, which are equivalent.

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 ‘the same as 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 with coupled congestion notification (network):
Using just two queues is not essential to L4S (more would be possible), but it is the simplest way to isolate all the L4S traffic that keeps latency low from all the legacy Classic traffic that does not.

Similarly, coupling the congestion notification between the queues is not necessarily essential, but it is a clever and simple way to allow senders to determine their rate, packet-by-packet, rather than be overridden by a network scheduler. Because otherwise a network scheduler would have to inspect at least transport layer headers, and it would have to continually assign a rate to each flow without any easy way to understand application intent.
L4S packet identifier (protocol): Once there are at least two separate treatments in the network, hosts need an identifier at the IP layer to distinguish which treatment they intend to use.

Scalable congestion notification (host): A scalable congestion control keeps the signalling frequency high so that rate variations can be small when signalling is 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 TCP-Friendly congestion control algorithms [RFC3649]. It was known when TCP was first developed that it would not scale to high bandwidth-delay products (see footnote 6 in [TCP-CA]). Today, regular broadband bit-rates over WAN distances are already beyond the scaling range of ‘classic’ TCP Reno. 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, at 800Mb/s with a 20ms round trip, Cubic induces a congestion signal only every 500 round trips or 10 seconds, which makes its dynamic control very sloppy. In contrast on average a scalable congestion control like DCTCP or TCP Prague induces 2 congestion signals per round trip, which remains invariant for any flow rate, keeping dynamic control very tight.

5.2. Why Not Alternative 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. L4S solely addresses the problem of queuing latency (as well as loss and throughput scaling). 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, if there are Diffserv classes for important traffic,
the L4S approach can provide low latency for _all_ traffic within each Diffserv class (including the case where there is only one Diffserv class).

Also, as already explained, Diffserv only works for a small subset of the traffic on a link. It is not applicable when all the applications in use at one time at a single site (home, small business or mobile device) require low latency. Also, because L4S is for all traffic, it needs none of the management baggage (traffic policing, traffic contracts) associated with favouring some packets over others. This baggage has held Diffserv back from widespread end-to-end deployment.

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. The L4S work is intended to complement these AQMs, and we definitely do not want to distract from the need to deploy them as widely as possible. Nonetheless, without addressing the large saw-toothing rate variations of Classic congestion controls, AQMs alone cannot reduce queuing delay too far without significantly reducing link utilization. The L4S approach resolves this tension by ensuring hosts can minimize the size of their sawteeth without appearing so aggressive to legacy flows that they starve them.

Per-flow queuing: Similarly per-flow queuing is not incompatible with the L4S approach. However, one queue for every flow can be thought of as overkill compared to the minimum of two queues for all traffic needed for the L4S approach. The overkill of per-flow queuing has side-effects:

A. fq makes high performance networking equipment costly (processing and memory) — in contrast dual queue code can be very simple;

B. fq requires packet inspection into the end-to-end transport layer, which doesn’t sit well alongside encryption for privacy — in contrast the use of ECN as the classifier for L4S requires no deeper inspection than the IP layer;

C. fq isolates the queuing of each flow from the others but not from itself so existing FQ implementations still needs to have support for scalable congestion control added.

It might seem that self-inflicted queuing delay should not count, 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 the interactive media applications described in Section 6, 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. They cannot shuffle packets once they have released them into the network.

D. fq prevents any one flow from consuming more than $1/N$ of the capacity at any instant, where $N$ is the number of flows. This is fine if all flows are elastic, but it does not sit well with a variable bit rate real-time multimedia flow, which requires wriggle room to sometimes take more and other times less than a $1/N$ share.

It might seem that an fq scheduler offers the benefit that it prevents individual flows from hogging all the bandwidth. However, L4S has been deliberately designed so that policing of individual flows can be added as a policy choice, rather than requiring one specific policy choice as the mechanism itself. A scheduler (like fq) has to decide packet-by-packet which flow to schedule without knowing application intent. Whereas a separate policing function can be configured less strictly, so that senders can still control the instantaneous rate of each flow dependent on the needs of each application (e.g. variable rate video), giving more wriggle-room before a flow is deemed non-compliant. Also policing of queuing and of flow-rates can be applied independently.

Alternative Back-off ECN (ABE): Yet 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).

BBRv1: v1 of Bottleneck Bandwidth and Round-trip propagation time (BBR [I-D.cardwell-iccrg-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. Setting some problems with capacity sharing aside, queuing delay is good with BBRv1, 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 its regular bandwidth probes and the aggressive flow start-up phase.

L4S is a complement to BBRv1. Indeed BBRv2 (not released at the time of writing) is likely to use L4S ECN and a TCP-Prague-like
behaviour if it discovers a compatible path. Otherwise it will use an evolution of BBRv1.

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 will immediately experience significantly better quality of experience under load:

- 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: DSL, cable, mobile, satellite
  - 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 often desirable to hold a buffer to utilise sudden increases of capacity;
  - cellular networks are further complicated by a perceived need to buffer in order to make hand-overs imperceptible;
  - Satellite networks generally have a very large base RTT, so even with minimal queuing, overall delay can never be extremely low;
* Nonetheless, it is certainly desirable not to hold a buffer purely because of the sawteeth of Classic TCP, when it is more than is needed for all the above reasons.

o 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)

o Different types of transport (or application) congestion control:
  * elastic (TCP/SCTP);
  * real-time (RTP, RMCAT);
  * query (DNS/LDAP).

o Where low delay quality of service is required, but without inspecting or intervening above the IP layer [I-D.smith-encrypted-traffic-management]:
  * 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 gives favourable queuing to all traffic.

o 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.

6.3. Deployment Considerations

The DualQ is, in itself, an incremental deployment framework for L4S AQMs so that L4S traffic can coexist with existing Classic "TCP-friendly" traffic. Section 6.3.1 explains why only deploying a DualQ AQM [I-D.ietf-tsvwg-aqm-dualq-coupled] 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 6.3.2 suggests some typical sequences to deploy each part, and why there will be an immediate and significant benefit after deploying just one part.

If an ECN-enabled DualQ AQM has not been deployed at a bottleneck, an L4S flow is required to include a fall-back strategy to Classic behaviour. Section 6.3.3 describes how an L4S flow detects this, and how to minimize the effect of false negative detection.

6.3.1. Deployment Topology

DualQ AQMs will not have to be deployed throughout the Internet before L4S will work for 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. 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, cellular, line-of-sight wireless or satellite.

Therefore, the full benefit of the L4S service should be available in the downstream direction when the DualQ AQM is deployed at the ingress to this bottleneck link (or links for multihomed sites). And similarly, the full upstream service will be available once the DualQ is deployed at the upstream ingress.
Deployment in mesh topologies depends on how over-booked 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 the DualQ 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).

The DualQ 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.

6.3.2. Deployment Sequences

For any one L4S flow to work, it requires 3 parts to have been deployed. This was the same deployment problem that ECN faced [RFC8170] so we have learned from this.

Firstly, L4S deployment exploits the fact that DCTCP already exists on many Internet hosts (Windows, FreeBSD and Linux); both servers and clients. Therefore, just deploying DualQ AQM at a network bottleneck immediately gives a working deployment of all the L4S parts. DCTCP needs some safety concerns to be fixed for general use over the public Internet (see Section 2.3 of [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 [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 they must use their own local identifier in combination with the IETF’s identifier. Then, if an operator enables the optional local-use approach, they only have to remove this extra rule to make the service work Internet-wide – it will already traverse middleboxes, peerings, etc.

<table>
<thead>
<tr>
<th>Servers or proxies</th>
<th>Access link</th>
<th>Clients</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 DCTCP (existing)</td>
<td>DualQ AQM downstream</td>
<td>DCTCP (existing)</td>
</tr>
<tr>
<td>WORKS DOWNSTREAM FOR CONTROLLED DEPLOYMENTS/_TRIALS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 2 TCP Prague | | AccECN (already in progress: DCTCP/BBR) |
| FULLY WORKS DOWNSTREAM |

| 3 | DualQ AQM upstream | TCP Prague |
| FULLY WORKS UPSTREAM AND DOWNSTREAM |

Figure 3: Example L4S Deployment Sequences

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, the DualQ 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 still be added).

2. In this stage, the name ‘TCP Prague’ is used to represent a variant of DCTCP that is safe to use in a production environment. If the application is primarily unidirectional, ‘TCP Prague’ at one end will provide all the benefit needed. 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 [I-D.cardwell-iccrg-bbr-congestion-control]. The two ends can be deployed in either order, because TCP Prague 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 [I-D.ietf-tsvwg-ecn-l4s-id]).

3. This is a two-move stage to enable L4S upstream. The DualQ 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. The DualQ AQM also greatly improves the upstream Classic service, assuming 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, RMCAT; a body such as the 3GPP might require L4S to be implemented in 5G user equipment, or other random acts of kindness.

6.3.3. L4S Flow but Non-L4S Bottleneck

If L4S is enabled between two hosts but there is no L4S AQM at the bottleneck, any drop from the bottleneck will trigger the L4S sender to fall back to a classic (‘TCP-Friendly’) behaviour (see Appendix A.1.3 of [I-D.ietf-tsvwg-ecn-l4s-id]).
Unfortunately, as well as protecting legacy traffic, this rule degrades the L4S service whenever there is a loss, even if the loss was not from a non-DualQ bottleneck (false negative). And unfortunately, prevalent drop can be due to other causes, 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 TCP Prague, ignore certain losses deemed unlikely to be due to congestion (using some ideas from BBR [I-D.cardwell-iccrg-bbr-congestion-control] but with no need to ignore nearly all losses). This could mask any of the above types of loss (requires consensus on how to safely interoperate with drop-based congestion controls).

- A combination of RACK, reconfigured link retransmission and L4S could address transmission errors [UnorderedLTE], [I-D.ietf-tsvwg-ecn-l4s-id];

- Hybrid ECN/drop 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.

Classic ECN support is starting to materialize on the Internet as an increased level of CE marking. Given some of this Classic ECN might be due to single-queue ECN deployment, an L4S sender will have to fall back to a classic ('TCP-Friendly') behaviour if it detects that ECN marking is accompanied by greater queuing delay or greater delay variation than would be expected with L4S (see Appendix A.1.4 of [I-D.ietf-tsvwg-ecn-l4s-id]). It is hard to detect whether this is all due to the addition of support for ECN in the Linux implementation of FQ-CoDel, which would not require fall-back to Classic behaviour, because FQ inherently forces the throughput of each flow to be equal irrespective of its aggressiveness.
6.3.4. Other Potential Deployment Issues

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-ecn-encap-guidelines].

7. IANA Considerations

This specification contains no IANA considerations.

8. Security Considerations

8.1. Traffic (Non-)Policing

Because the L4S service can serve all traffic that is using the capacity of a link, it should not be necessary to police access to the L4S service. In contrast, Diffserv only works if some packets get less favourable treatment than others. So Diffserv has to use traffic policers to limit how much traffic can be favoured. In turn, traffic policers require traffic contracts between users and networks as well as pairwise between networks. Because L4S will lack all 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. These networks do not need to police or remark L4S traffic - they just forward it unchanged as best efforts traffic, as they already forward traffic with ECT(1) 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 as if it is a Classic congestion control (as required in Section 2.3 of [I-D.ietf-tsvwg-ecn-l4s-id]). This will ensure safe interworking with other traffic at the 'legacy' bottleneck, but it will degrade the L4S service to no better (but never worse) than classic best efforts, whenever a legacy (non-L4S) bottleneck is encountered on a path.

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 ECN. 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. Clearly explaining how operators can use an additional local classifiers (see [I-D.ietf-tsvwg-ecn-l4s-id]) is intended to remove any tendency to bleach 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 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’

The L4S service does rely on self-constraint - not in terms of limiting rate, but in terms of limiting latency (burstiness). It is hoped that self-interest and standardisation of dynamic behaviour (cf. TCP slow-start) 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. However it may only be necessary to apply such policing reactively, e.g. punitively targeted at any deployments of new bursty malware.

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.

Such a queue protection function is not considered a necessary part of the L4S architecture, which works without it (in a similar way to how the Internet works without per-flow rate policing). Indeed, under normal circumstances, DOCSIS queue protection does not intervene, and if operators find it is not necessary they can disable it. Part of the L4S experiment will be to see whether such a function is necessary.

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 may be dropped elsewhere during contention). Whenever L4S traffic encounters one of these rate policers, it will experience drops and the source has to fall back to a Classic congestion control, thus losing the benefits of L4S. 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.

This is currently a research area. 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. This could be applied to various types of policer, e.g. [RFC2697], [RFC2698] or the ‘local’ (non-ConEx) variant of the ConEx 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 TCP 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 [RFC3168]).

- A network can enforce a congestion response to its ECN markings (or packet losses) by auditing congestion exposure (ConEx) [RFC7713].
Appendix C.1 of [I-D.ietf-tsvwg-ecn-l4s-id] gives more details of these techniques including their applicability and pros and cons.

9. Acknowledgements

Thanks to Richard Scheffenegger, Wes Eddy, Karen Nielsen and David Black 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). Bob Briscoe was also part-funded by the Research Council of Norway through the TimeIn project. The views expressed here are solely those of the authors.

10. References

10.1. Normative References


10.2. Informative References


Briscoe, B., "Interactions between Low Latency, Low Loss, Scalable Throughput (L4S) and Differentiated Services", draft-briscoe-tsvwg-l4s-diffserv-02 (work in progress), November 2018.


Schepper, K., Briscoe, B., and G. White, "DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)", draft-ietf-tsvwg-aqm-dualq-coupled-09 (work in progress), July 2019.

[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-06 (work in progress), March 2019.

[I-D.smith-encrypted-traffic-management]
Smith, K., "Network management of encrypted traffic",
draft-smith-encrypted-traffic-management-05 (work in
progress), May 2016.

[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", draft-sridharan-tcpm-ctcp-02
(work in progress), November 2008.

[I-D.stewart-tsvwg-sctpecn]
Stewart, R., Tuexen, M., and X. Dong, "ECN for Stream
Control Transmission Protocol (SCTP)", draft-stewart-
tsvwg-sctpecn-05 (work in progress), January 2014.

[I-D.white-tsvwg-nqb]
White, G. and T. Fossati, "Identifying and Handling Non
Queue Building Flows in a Bottleneck Link", draft-white-
tsvwg-nqb-02 (work in progress), June 2019.

[L4Sdemo16]
Bondarenko, O., De Schepper, K., Tsang, I., and B.
Briscoe, "orderedUltra-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 )>.

[Mathis09]
Mathis, M., "Relentless Congestion Control", PFLDNeT’09,
presentations/2010-07-06-questnet-tcp/reference-

[NewCC_Proc]
Eggert, L., "Experimental Specification of New Congestion
Control Algorithms", IETF Operational Note ion-tsv-alt-cc,


Appendix A.  Standardization items

The following table includes all the items that will need to be standardized to provide a full L4S architecture.

The table is too wide for the ASCII draft format, so it has been split into two, with a common column of row index numbers on the left.

The columns in the second part of the table have the following meanings:

WG:  The IETF WG most relevant to this requirement. The "tcpm/iccrq" combination refers to the procedure typically used for congestion control changes, where tcpm owns the approval decision, but uses the iccrq for expert review [NewCC_Proc];

TCP:  Applicable to all forms of TCP congestion control;

DCTCP: Applicable to Data Center TCP as currently used (in controlled environments);

DCTCP bis: Applicable to an future Data Center TCP congestion control intended for controlled environments;

XXX Prague: Applicable to a Scalable variant of XXX (TCP/SCTP/RMCAT) congestion control.
<table>
<thead>
<tr>
<th>Req #</th>
<th>Requirement</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ARCHITECTURE</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>L4S IDENTIFIER</td>
<td>[I-D.ietf-tsvwg-ecn-l4s-id]</td>
</tr>
<tr>
<td>2</td>
<td>DUAL QUEUE AQM</td>
<td>[I-D.ietf-tsvwg-aqm-dualq-coupled]</td>
</tr>
<tr>
<td>3</td>
<td>Suitable ECN Feedback</td>
<td>[I-D.ietf-tcpm-accurate-ecn], [I-D.stewart-tsvwg-sctpecn].</td>
</tr>
</tbody>
</table>

### SCALABLE TRANSPORT - SAFETY ADDITIONS

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-1 Fall back to Reno/Cubic on loss</td>
<td>[I-D.ietf-tsvwg-ecn-l4s-id] S.2.3, [RFC80257]</td>
</tr>
<tr>
<td>4-2 Fall back to Reno/Cubic if classic ECN bottleneck detected</td>
<td>[I-D.ietf-tsvwg-ecn-l4s-id] S.2.3</td>
</tr>
<tr>
<td>4-3 Reduce RTT-dependence</td>
<td>[I-D.ietf-tsvwg-ecn-l4s-id] S.2.3</td>
</tr>
<tr>
<td>4-4 Scaling TCP’s Congestion Window for Small Round Trip Times</td>
<td>[I-D.ietf-tsvwg-ecn-l4s-id] S.2.3, [TCP-sub-mss-w]</td>
</tr>
</tbody>
</table>

### SCALABLE TRANSPORT - PERFORMANCE ENHANCEMENTS

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-1 Setting ECT in TCP Control Packets and Retransmissions</td>
<td>[I-D.ietf-tcpm-generalized-ecn]</td>
</tr>
<tr>
<td>5-2 Faster-than-additive increase</td>
<td>[I-D.ietf-tsvwg-ecn-l4s-id] (Appx A.2.2)</td>
</tr>
<tr>
<td>5-3 Faster Convergence at Flow Start</td>
<td>[I-D.ietf-tsvwg-ecn-l4s-id] (Appx A.2.2)</td>
</tr>
<tr>
<td>#</td>
<td>WG</td>
</tr>
<tr>
<td>----</td>
<td>---------</td>
</tr>
<tr>
<td>0</td>
<td>tsvwg</td>
</tr>
<tr>
<td>1</td>
<td>tsvwg</td>
</tr>
<tr>
<td>2</td>
<td>tsvwg</td>
</tr>
<tr>
<td>3</td>
<td>tcpm</td>
</tr>
<tr>
<td>4-1</td>
<td>tcpm</td>
</tr>
<tr>
<td>4-2</td>
<td>tcpm/iccrg?</td>
</tr>
<tr>
<td>4-3</td>
<td>tcpm/iccrg?</td>
</tr>
<tr>
<td>4-4</td>
<td>tcpm</td>
</tr>
<tr>
<td>5-1</td>
<td>tcpm</td>
</tr>
<tr>
<td>5-2</td>
<td>tcpm/iccrg?</td>
</tr>
<tr>
<td>5-3</td>
<td>tcpm/iccrg?</td>
</tr>
</tbody>
</table>

Authors' Addresses

Bob Briscoe (editor)
CableLabs
UK

Email: ietf@bobbriscoe.net
URI: http://bobbriscoe.net/
Koen De Schepper  
Nokia Bell Labs  
Antwerp  
Belgium

Email: koen.de_schepper@nokia.com  
URI: 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  
URI: http://www.it.uc3m.es

Greg White  
CableLabs  
US

Email: G.White@CableLabs.com
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
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] 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:

- For traffic caused by world-wide web search engines while they gather information from web servers.
- For software updates or dissemination of new releases of operating systems.
- For reporting errors or telemetry data from operating systems or applications.
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

- A BA classifier (Behavior Aggregate classifier, see [RFC2475]) that classifies packets according to the LE DSCP
- A dedicated LE queue
- 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:

- 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.

- 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.

- 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. Remarking 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 remarking 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., forward 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:

- This update to RFC 4594 removes the following entry from figure 3:

|---------------+---------+-------------+--------------------------|
| Low-Priority  |  CS1    |   001000    | Any flow that has no BW  |
| Data          |         |             | assurance                |
|---------------+---------+-------------+--------------------------|

and replaces this by the following entry:
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’.”

This update to RFC 4594 removes the following entry from figure 4:

<table>
<thead>
<tr>
<th>Low-Priority Data</th>
<th>CS1</th>
<th>Not applicable</th>
<th>RFC3662</th>
<th>Rate</th>
<th>Yes</th>
</tr>
</thead>
</table>

and replaces this by the following entry:

<table>
<thead>
<tr>
<th>Low-Priority Data</th>
<th>LE</th>
<th>Not applicable</th>
<th>RFCXXXX</th>
<th>Rate</th>
<th>Yes</th>
</tr>
</thead>
</table>

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."
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:

<table>
<thead>
<tr>
<th>Low-Priority Data</th>
<th>CS1</th>
<th>RFC 3662</th>
<th>1</th>
<th>AC_BK (Background)</th>
</tr>
</thead>
</table>

by the following entries:

<table>
<thead>
<tr>
<th>Low-Priority Data</th>
<th>LE</th>
<th>RFCXXXX</th>
<th>1</th>
<th>AC_BK (Background)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Low-Priority Data (legacy)</th>
<th>CS1</th>
<th>RFC 3662</th>
<th>1</th>
<th>AC_BK (Background)</th>
</tr>
</thead>
</table>

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:

<table>
<thead>
<tr>
<th>Flow Type</th>
<th>Very Low</th>
<th>Low</th>
<th>Medium</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
<td>LE (1)</td>
<td>DF</td>
<td>EF (46)</td>
<td>EF (46)</td>
</tr>
<tr>
<td>Interactive Video with</td>
<td>LE (1)</td>
<td>DF</td>
<td>AF42, AF43</td>
<td>AF41, AF42</td>
</tr>
<tr>
<td>or without Audio</td>
<td></td>
<td></td>
<td>(36, 38)</td>
<td>(34, 36)</td>
</tr>
<tr>
<td>Non-Interactive Video</td>
<td>LE (1)</td>
<td>DF</td>
<td>AF32, AF33</td>
<td>AF31, AF32</td>
</tr>
<tr>
<td>with or without Audio</td>
<td></td>
<td></td>
<td>(28, 30)</td>
<td>(26, 29)</td>
</tr>
<tr>
<td>Data</td>
<td>LE (1)</td>
<td>DF</td>
<td>AF11</td>
<td>AF21</td>
</tr>
</tbody>
</table>

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:

- Name: LE
- Value (Binary): 000001
- Value (Decimal): 1
- 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


15.2. Informative References


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, Gorry 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)

- Appended "for Differentiated Services" to the title as suggested by Alexey.
- 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 Gorry.
- 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.
- Made reference to RFC8174 normative.
- Added hint for the RFC editor to apply changes from section 12 and to delete it afterwards.
Changes in Version 09:

- 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:

- revised two sentences as suggested by Spencer Dawkins

Changes in Version 07:

- revised some text for clarification according to comments from Spencer Dawkins

Changes in Version 06:

- added Multicast Considerations section with input from Toerless Eckert
- incorporated suggestions by David Black with respect to better reflect legacy CS1 handling

Changes in Version 05:

- added scavenger service class into abstract
- added some more history
- added reference for "Myth of Over-Provisioning" in RFC3439 and references to presentations w.r.t. codepoint choices
- added text to update draft-ietf-rtcweb-qos
- revised text on congestion control in case of remarking to BE
- added reference to DSCP measurement talk @IETF99
- small typo fixes
Changes in Version 04:
- Several editorial changes according to review from Gorry Fairhurst
- Changed the section structure a bit (moved subsections 1.1 and 1.2 into own sections 3 and 7 respectively)
- Updated section 2 on requirements language
- Added updates to RFC 8325
- Tried to be more explicit what changes are required to RFCs 4594 and 8325

Changes in Version 03:
- Changed recommended codepoint to 000001
- Added text to explain the reasons for the DSCP choice
- Removed LE-min,LE-strict discussion
- Added one more potential use case: reporting errors or telemetry data from OSs
- Added privacy considerations to the security section (not worth an own section I think)
- Changed IANA considerations section

Changes in Version 02:
- Applied many editorial suggestions from David Black
- Added Multicast traffic use case
- Clarified what is required for deployment in section 1.2 (Deployment Considerations)
- Added text about implementations using AQMs and ECN usage
- Updated IANA section according to David Black’s suggestions
- Revised text in the security section
- Changed copyright Notice to pre5378Trust200902

Changes in Version 01:
- Now obsoletes RFC 3662.

- 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.

- Change section 2 (PHB Description) according to R. Geib’s suggestions.

- Added RFC 2119 language to several sentences.

- Detailed the description of remarking implications and recommendations in Section 8.

- 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.

- Apply the suggested changes of section Section 12 and add a normative reference in draft-ietf-tsvwg-rtcweb-qos to this RFC.

- Delete Section 12.

- Please replace the occurrences of RFCXXXX in Section 10 and Section 11 with the assigned RFC number for this document.

- Delete Appendix C.

- 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
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 for TCP that allows multiple hosts to reside behind a NAT and yet use only a single globally unique IPv4 address, even when two hosts (behind a NAT) 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 required for the SCTP endpoints and the mechanisms for NAT devices necessary to provide similar features of NAPT in the single-point and multi-point traversal scenario.
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. Conventions .................................................. 5
3. Terminology .................................................. 5
4. Motivation ...................................................... 6
  4.1. SCTP NAT Traversal Scenarios ............................. 6
  4.1.1. Single Point Traversal ................................ 6
  4.1.2. Multi Point Traversal ................................ 7
  4.2. Limitations of Classical NAPT for SCTP ................. 8
  4.3. The SCTP Specific Variant of NAT ....................... 8
5. Data Formats ................................................. 12
  5.1. Modified Chunks ........................................... 12
  5.1.1. Extended ABORT Chunk ................................ 12
  5.1.2. Extended ERROR Chunk ................................ 13
  5.2. New Error Causes .......................................... 13
  5.2.1. VTag and Port Number Collision Error Cause .......... 13
  5.2.2. Missing State Error Cause ............................. 14
  5.2.3. Port Number Collision Error Cause ................... 15
  5.3. New Parameters ............................................. 15
  5.3.1. Disable Restart Parameter ............................. 16
  5.3.2. VTags Parameter ........................................ 16
6. Procedures for SCTP Endpoints and NAT Devices ............... 17
  6.1. Association Setup Considerations for Endpoints .......... 18
  6.2. Handling of Internal Port Number and Verification Tag
       Collisions ................................................. 18
  6.2.1. NAT Device Considerations ............................ 19
  6.2.2. Endpoint Considerations ............................... 19
  6.3. Handling of Internal Port Number Collisions ............ 19
  6.3.1. NAT Device Considerations ............................ 20
  6.3.2. Endpoint Considerations ............................... 20
  6.4. Handling of Missing State ............................... 21
  6.4.1. NAT Device Considerations ............................ 21
1. Introduction

Stream Control Transmission Protocol [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 for TCP that allows multiple hosts to reside behind a NAT using private addresses (see [RFC6890]) and yet use only a single globally unique IPv4 address, even when two hosts (behind a NAT) 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 maps multiple private addresses to a single public address. To date, specialized code for SCTP has not yet been added to most NAT devices so that only true NAT is available. The end result of this is that only one SCTP capable host can be behind a NAT and this host can only be single-homed. The only alternative for supporting legacy NAT devices is to use UDP encapsulation as specified in [RFC6951].

This document specifies procedures allowing a NAT to support SCTP by providing similar features to those provided by a NAPT for TCP and other supported protocols. The 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 will maintain the proper state without 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’s public IP address in the same way that a TCP session can use a NAT.

- If a NAT 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 required by SCTP.

- Not having to touch the IP payload makes the processing of ICMP messages in NAT devices easier.

An SCTP-aware NAT will need to follow these procedures for generating appropriate SCTP packet formats.

When considering this feature it is possible to have multiple levels of support. At each level, the Internal Host, External Host and NAT may or may not support the features described in this document. The following table illustrates the results of the various combinations of support and if communications can occur between two endpoints.

<table>
<thead>
<tr>
<th>Internal Host</th>
<th>NAT Device</th>
<th>External Host</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support</td>
<td>Support</td>
<td>Support</td>
<td>Yes</td>
</tr>
<tr>
<td>Support</td>
<td>No Support</td>
<td>No Support</td>
<td>Limited</td>
</tr>
<tr>
<td>Support</td>
<td>No Support</td>
<td>No Support</td>
<td>None</td>
</tr>
<tr>
<td>No Support</td>
<td>Support</td>
<td>Support</td>
<td>Limited</td>
</tr>
<tr>
<td>No Support</td>
<td>Support</td>
<td>No Support</td>
<td>Limited</td>
</tr>
<tr>
<td>No Support</td>
<td>No Support</td>
<td>Support</td>
<td>None</td>
</tr>
<tr>
<td>No Support</td>
<td>No Support</td>
<td>No Support</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 1: Communication possibilities

From the table it can be seen that when a NAT device does not support the extension no communication can occur. This assumes that the NAT device does not handle SCTP packets at all and all SCTP packets sent externally from behind a NAT device are discarded by the NAT. In some cases, where the NAT device supports the feature but one of the two hosts does not support the feature, communication may occur but
in a limited way. For example only one host may be able to have a connection when a collision case occurs.

2. Conventions

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. Terminology

This document uses the following terms, which are depicted in Figure 1. Familiarity with the terminology used in [RFC4960] and [RFC5061] is assumed.

Private-Address (Priv-Addr): The private address that is known to the internal host.

Internal-Port (Int-Port): The port number that is in use by the host holding the Private-Address.

Internal-VTag (Int-VTag): The SCTP Verification Tag (VTag) that the internal host has chosen for its communication. The VTag is a unique 32-bit tag that must accompany any incoming SCTP packet for this association to the Private-Address.

External-Address (Ext-Addr): The address that an internal host is attempting to contact.

External-Port (Ext-Port): The port number of the peer process at the External-Address.

External-VTag (Ext-VTag): The Verification Tag that the host holding the External-Address has chosen for its communication. The VTag is a unique 32-bit tag that must accompany any incoming SCTP packet for this association to the External-Address.

Public-Address (Pub-Addr): The public address assigned to the NAT device that it uses as a source address when sending packets towards the External-Address.
4. Motivation

4.1. SCTP NAT Traversal Scenarios

This section defines the notion of single and multi-point NAT traversal.

4.1.1. Single Point Traversal

In this case, all packets in the SCTP association go through a single NAT, as shown below:

```
Internal Network   |   External Network
SCTP endpoint A   +-----+   INTERNET +-----+   SCTP endpoint B
| Address           | NAT     | Address     |
Private            | Public  | External    |
VTag               |         | VTag        |
Internal Port      +-----+   External Port  +-----+
```

Single NAT scenario

A variation of this case is shown below, i.e., multiple NAT devices in a single path:

```
Internal Network   |   External Network
SCTP endpoint A   +-----+   INTERNET +-----+   SCTP endpoint B
| Address           | NAT     | Address     |
Private            | Public  | External    |
VTag               |         | VTag        |
Internal Port      +-----+   External Port  +-----+
```

Figure 1: Basic network setup
Serial NAT Devices scenario

Although one of the main benefits of SCTP multi-homing is redundant paths, in this single point traversal scenario the NAT function represents a single point of failure in the path of the SCTP multi-home association. However, the rest of the path may 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 device (or NAT devices) in this case sees all the packets of the SCTP association.

4.1.2. Multi Point Traversal

This case involves multiple NAT devices and each NAT device only sees some of the packets in the SCTP association. An example is shown below:

Parallel NAT devices 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 may result in changing one of the SCTP port numbers during the processing which requires the recomputation of the transport layer checksum. Whereas for UDP and TCP this can be done very efficiently, for SCTP the checksum (CRC32c) over the entire packet needs to be recomputed. This would considerably add to the NAT computational burden, however hardware support may mitigate this in some implementations.

An SCTP endpoint may have multiple addresses but only has a single port number. To make multipoint traversal work, all the NAT devices involved must 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 pre-defined table of ports and addresses configured within each NAT. Other mechanisms could make use of NAT to NAT communication. Such mechanisms are not to be deployable on a wide scale base and thus not a recommended solution. Therefore the SCTP variant of NAT has been developed.

4.3. The SCTP Specific Variant of NAT

In this section it is assumed that there are multiple SCTP capable hosts behind a NAT that has one Public-Address. Furthermore this section focuses on the single point traversal scenario.

The modification of SCTP packets sent to the public Internet is simple: the source address of the packet has to be replaced with the Public-Address. It may also be necessary to establish some state in the NAT device to later handle incoming packets.

For the SCTP NAT processing the NAT device has to maintain a table of Internal-VTag, Internal-Port, External-VTag, External-Port, Private-Address, and whether the restart procedure is disabled or not. An entry in that table is called a NAT state control block. The function Create() obtains the just mentioned parameters and returns a NAT-State control block.

For SCTP packets coming from the public Internet the destination address of the packets has to be replaced with the Private-Address of the host the packet has to be delivered to. The lookup of the Private-Address is based on the External-VTag, External-Port, Internal-VTag and the Internal-Port.

The entries in the table fulfill some uniqueness conditions. There must not be more than one entry with the same pair of Internal-Port and External-Port. This rule can be relaxed, if all entries with the same Internal-Port and External-Port have the support for the restart
procedure enabled. In this case there must be no more than one entry
with the same Internal-Port, External-Port and Ext-VTag and no more
than one entry with the same Internal-Port, External-Port and Int-
VTag.

The processing of outgoing SCTP packets containing an INIT-chunk is
described in the following figure. The scenario shown is valid for
all message flows in this section.

```
+--------+          +-----+           /        \
| Host A | <------> | NAT | <------> | Internet | <------> | Host B |
+--------+          +-----+           \
                 \--/
```

INIT[Initiate-Tag]
Priv-Addr:Int-Port ------> Ext-Addr:Ext-Port
Ext-VTag=0

Create(Initiate-Tag, Int-Port, 0, Ext-Port, Priv-Addr, RestartSupported)
Returns(NAT-State control block)

Translate To:

```
+--------+          +-----+           /        \
| Host A | <------> | NAT | <------> | Internet | <------> | Host B |
+--------+          +-----+           \
                 \--/
```

INIT[Initiate-Tag]
Pub-Addr:Int-Port ------> Ext-Addr:Ext-Port
Ext-VTag=0

Normally a NAT control block will be created. However, it is
possible that there is already a NAT control block with the same
External-Address, External-Port, Internal-Port, and Internal-VTag but
different Private-Address. In this case the INIT MUST be dropped by
the NAT and an ABORT MUST be sent back to the SCTP host with the
M-Bit set and an appropriate 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.

It is also possible that a connection to External-Address and
External-Port exists without an Internal-VTag conflict but the
External-Address does not support the DISABLE_RESTART feature (noted
in the NAT control block when the prior connection was established).
In such a case the INIT SHOULD be dropped by the NAT and an ABORT
SHOULD be sent back to the SCTP host with the M-Bit set and an appropriate error cause (see Section 5.1.1 for the format).

The processing of outgoing SCTP packets containing no INIT-chunk is described in the following figure.

\[\begin{align*}
\text{Host A} & \quad <\text{------}> \quad \text{NAT} \quad <\text{------}> \quad \text{Internet} \quad <\text{------}> \quad \text{Host B} \\
\text{Priv-Addr:Int-Port} & \quad \rightarrow \quad \text{Ext-Addr:Ext-Port} \\
\text{Ext-VTag} & \quad \rightarrow \quad \text{Pub-Addr:Ext-Port} \\
\end{align*}\]

The processing of incoming SCTP packets containing INIT-ACK chunks is described in the following figure. The Lookup() function getting as input the Internal-VTag, Internal-Port, External-VTag, and External-Port, returns the corresponding entry of the NAT table and updates the External-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.
In the case Lookup fails, the SCTP packet is dropped. The Update routine inserts the External-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 described in the following figure.
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 an External-VTag of zero is found, it is considered a match and the External-VTag is updated.

This allows the handling of INIT-collision through NAT.

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 devices and received by SCTP endpoints. Section 5.3 describes parameters sent by SCTP endpoints and used by NAT devices and SCTP endpoints.

5.1. Modified Chunks

This section presents existing chunks defined in [RFC4960] that are modified by this document.

5.1.1. Extended ABORT Chunk

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 6    | Reserved  |M|T|           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               
/                   zero or more Error Causes                   /  
\                                                               
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
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.

[NOTE to RFC-Editor:

Assignment of M-bit to be confirmed by IANA.
]

5.1.2. Extended ERROR Chunk

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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 9    | Reserved  |M|T|           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\ +
\ /                   zero or more Error Causes                   /
\ /
\ +
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

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

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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Cause Code = 0x00B0        |     Cause Length = Variable   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                              Chunk                              /
\ /                              +
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
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

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.

Incoming 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.
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.

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.]

This 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 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 ASCONF can use this same value in the ASCONF-ACK to find which request the response is for. Note that the receiver MUST NOT change this 32-bit value.

Internal Verification Tag: 4 bytes (unsigned integer)
    The Verification Tag that the internal host has chosen for its communication. The Verification Tag is a unique 32-bit tag that must accompany any incoming SCTP packet for this association to the Private-Address.

External Verification Tag: 4 bytes (unsigned integer)
    The Verification Tag that the host holding the External-Address has chosen for its communication. The VTag is a unique 32-bit tag that must accompany any incoming SCTP packet for this association to the External-Address.

[NOTE to RFC-Editor:

    Assignment of parameter type to be confirmed by IANA.

]

This parameter MAY appear in ASCONF chunks and MUST NOT appear in any other chunk.

6. Procedures for SCTP Endpoints and NAT Devices

When an SCTP endpoint is behind an SCTP-aware NAT a number of problems may arise as it tries to communicate with its peer:

- IP addresses cannot not be included in the SCTP packet. This is discussed in Section 6.1.
o More than one host behind a NAT device could select the same VTag and source port when talking to the same peer server. This creates a situation where the NAT will not be able to tell the two associations apart. This situation is discussed in Section 6.2.

o When an SCTP endpoint is a server communicating with multiple peers and the peers are behind the same NAT, then the two endpoints cannot be distinguished by the server. This case is discussed in Section 6.3.

o A restart of a NAT during a conversation could cause a loss of its state. This problem and its solution is discussed in Section 6.4.

o NAT devices need to deal with SCTP packets being fragmented at the IP layer. This is discussed in Section 6.5.

o An SCTP endpoint may be behind two NAT devices providing redundancy. The method to set up this scenario is discussed in Section 6.6.

Each of these mechanisms requires additional chunks and parameters, defined in this document, and possibly modified handling procedures from those specified in [RFC4960].

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 doesn't work when NAT devices are present.

Every association MUST initially be set up single-homed. There MUST NOT be any IPv4 Address parameter, IPv6 Address parameter, or Supported Address Types parameter in the INIT-chunk. The INIT-ACK chunk MUST NOT contain any IPv4 Address parameter or IPv6 Address parameter.

If the association should finally be 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 Private-Address space want to set up an SCTP association with the same service provided by some hosts in the Internet. This means that the External-Port is the
same. If they both choose the same Internal-Port and Internal-VTag, the NAT device cannot distinguish between incoming packets anymore. But this is very unlikely. The Internal-VTags are chosen at random and if the Internal-Ports are also chosen from the ephemeral port range at random this gives a 46-bit random number which has to match. In the TCP-like NAPT case the NAT device can control the 16-bit Natted Port and therefore avoid collisions deterministically.

The same can happen with the External-VTag when an INIT-ACK chunk or an ASCONF chunk is processed by the NAT.

6.2.1. NAT Device Considerations

However, in this unlikely event the NAT device MUST send an ABORT chunk with the M-bit set if the collision is triggered by an INIT or INIT-ACK chunk or send an ERROR chunk with the M-bit set if the collision is triggered by an ASCONF chunk. 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. In the other two cases, the source and destination address and port numbers MUST be swapped.

The sender of the 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 the INIT-ACK chunk, upon reception of an ABORT chunk with M-bit set and the appropriate error cause code for colliding NAT table state is included, MUST 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 ASCONF chunk, upon reception of 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 Private-Address space will want to set up an SCTP association with the same server running on the same host in the Internet. For the NAT, appropriate tracking may be performed by assuring that the VTags are unique between the two hosts.
6.3.1. NAT Device Considerations

The NAT, when processing the INIT-ACK, should note in its internal table that the association supports the Disable Restart extension. This note is used when establishing future associations (i.e., when processing an INIT from an internal host) to decide if the connection should be allowed. The NAT device does the following when processing an INIT:

- If the INIT is destined to an external address and port for which the NAT device has no outbound connection, it MUST allow the INIT creating an internal mapping table.

- If the INIT matches the external address and port of an already existing connection, it MUST validate that the external server supports the Disable Restart feature and, if it does, allow the INIT to be forwarded.

- If the external server does not support the Disable Restart extension the NAT device MUST send an ABORT with the M-bit set. The ‘Port Number Collision’ error cause (see Section 5.2.3) MUST be included in the ABORT chunk.

If the collision is triggered by an ASCONF chunk, a packet containing an ERROR chunk with the ‘Port Number Collision’ error cause MUST be sent back.

6.3.2. Endpoint Considerations

For the external SCTP server on the Internet this means that the External-Port and the External-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) differentiated only by the VTags.

6.4. Handling of Missing State

6.4.1. NAT Device Considerations

If the NAT device receives a packet from the internal network for which the lookup procedure does not find an entry in the NAT table, a packet containing an ERROR chunk is sent back with the M-bit set. The source address of the packet containing the ERROR chunk MUST be the destination address of the incoming SCTP packet. The verification tag is reflected and the T-bit is set. Please note that such a packet containing an ERROR chunk SHOULD NOT be sent if the received packet contains an ABORT, SHUTDOWN-COMPLETE or INIT-ACK chunk. An ERROR chunk MUST NOT be sent if the received packet contains an ERROR chunk with the M-bit set.

When sending the ERROR chunk, the new error cause ’Missing State’ (see Section 5.2.2) MUST be included and the new M-bit of the ERROR chunk MUST be set (see Section 5.1.2).

If the NAT device receives a packet for which it has no NAT table entry and the packet contains an ASCONF chunk with the VTags parameter, the NAT device MUST update its NAT table according to the verification tags in the VTags parameter and the optional Disable Restart parameter.

6.4.2. Endpoint Considerations

Upon reception of this 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 the outgoing packet.
- It SHOULD validate that the peer of the SCTP association supports the dynamic address extension, if it does not discard the incoming ERROR chunk.
- It SHOULD generate a new ASCONF chunk containing the VTags parameter (see Section 5.3.2) and the Disable Restart parameter if the association is using the disabled restart feature. By processing this packet the NAT device can recover the appropriate state. The procedures for generating an ASCONF chunk can be found in [RFC5061].
The peer SCTP endpoint receiving such an ASCONF chunk SHOULD either add the address and respond with an acknowledgment, if the address is new to the association (following all procedures defined in [RFC5061]). Or, if the address is already part of the association, the SCTP endpoint MUST NOT respond with an error, but instead should respond with an ASCONF-ACK chunk acknowledging the address but take no action (since the address is already in the association).

Note that it is possible that upon receiving an ASCONF chunk containing the VTags parameter the NAT will realize that it has an ‘Internal Port Number and Verification Tag collision’. In such a case the NAT MUST send 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 an ERROR 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 required. 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.
- It MUST validate that the peer of the SCTP association supports the dynamic address extension. If the peer does not support it, the NAT Device MUST discard the incoming 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 devie has a VTag collision and the association cannot easily create a new VTag (as it would if the error occurred when sending an INIT).
- If the endpoint has no other path, i.e. the procedure was executed due to missing a state in the NAT device, then the endpoint MUST abort the association. This would occur only if the local NAT device 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 device looses its state).

6.5. Handling of Fragmented SCTP Packets by NAT Devices

A NAT device MUST support IP reassembly of received fragmented SCTP packets. The fragments may arrive in any order.
When an SCTP packet has to be fragmented by the NAT device and the IP header forbids fragmentation a corresponding ICMP packet SHOULD be sent.

6.6. Multi-Point Traversal Considerations for Endpoints

If a multi-homed SCTP endpoint behind a NAT connects to a peer, it SHOULD first set up the association single-homed with only one address causing the first NAT to populate its state. Then it SHOULD add each IP address using ASCONF chunks sent via their respective NAT devices. The address to add is the wildcard address and the lookup address SHOULD also contain the VTags parameter and optionally the Disable Restart parameter as illustrated above.

7. 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.

7.1. Single-homed Client to Single-homed Server

The internal client starts the association with the external server via a four-way-handshake. Host A starts by sending an INIT chunk.

```
+----------+----------+----------+----------+
| Host A   |  NAT     |  Internet| Host B   |
| +--------+----------+----------+----------+
| [Initiate-Tag = 1234] | 10.0.0.1:1 | 203.0.113.1:2 |
```

A NAT entry is created, the source address is substituted and the packet is sent on:
NAT creates entry:

<table>
<thead>
<tr>
<th>NAT</th>
<th>Int VTag</th>
<th>Int Port</th>
<th>Ext VTag</th>
<th>Ext Port</th>
<th>Priv Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1234</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>10.0.0.1</td>
</tr>
</tbody>
</table>

INIT[Initiate-Tag = 1234]
192.0.2.1:1 ------------------------> 203.0.113.1:2
Ext-VTag = 0

Host B receives the INIT and sends an INIT-ACK with the NAT’s external address as destination address.

| Host A | <------> | NAT | <------> | Internet | <------> | Host B |
|--------|<------>|<------|<------|---------|<------|--------|
|        |       |      |       |         |       |        |

INIT-ACK[Initiate-Tag = 5678]
192.0.2.1:1 <----------------------- 203.0.113.1:2
Int-VTag = 1234

NAT updates entry:

<table>
<thead>
<tr>
<th>NAT</th>
<th>Int VTag</th>
<th>Int Port</th>
<th>Ext VTag</th>
<th>Ext Port</th>
<th>Priv Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1234</td>
<td>1</td>
<td>5678</td>
<td>2</td>
<td>10.0.0.1</td>
</tr>
</tbody>
</table>

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.
7.2. Single-homed Client to Multi-homed Server

The internal client is single-homed whereas the external server is multi-homed. The client (Host A) sends an INIT like in the single-homed case.
INIT[Initiate-Tag = 1234]
10.0.0.1:1 ----> 203.0.113.1:2
Ext-VTag = 0

NAT creates entry:

<table>
<thead>
<tr>
<th>Int VTag</th>
<th>Int Port</th>
<th>Ext VTag</th>
<th>Ext Port</th>
<th>Priv Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>10.0.0.1</td>
</tr>
</tbody>
</table>

INIT[Initiate-Tag = 1234]
192.0.2.1:1 --------------------------> 203.0.113.1:2
Ext-VTag = 0

The server (Host B) includes its two addresses in the INIT-ACK chunk, which results in two NAT entries.
The handshake finishes with a COOKIE-ECHO acknowledged by a COOKIE-ACK.
7.3. Multihomed Client and Server

The client (Host A) sends an INIT to the server (Host B), but does not include the second address.
Host A includes its second address in the INIT-ACK, which results in two NAT entries in NAT 1.
NAT 1 does not need to update the table for second address:

```
+--------+--------+----------+--------+-----------+
| VTag   | Port   | VTag     | Port   | 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.
Host A announces its second address in an ASCONF chunk. The address parameter contains an undefined address (0) to indicate that the source address should be added. The lookup address parameter within the ASCONF chunk will also contain the pair of VTags (external and internal) so that the NAT may populate its table completely with this single packet.

ASCONF [ADD-IP=0.0.0.0, INT-VTag=1234, Ext-VTag=5678]
10.1.0.1:1 --------> 203.0.113.129:2
Ext-VTag = 5678

NAT 2 creates complete entry:
Association is already established between Host A and Host B, when the NAT loses its state and obtains a new public address. Host A sends a DATA chunk to Host B.

The NAT device cannot find entry for the association. It sends ERROR message 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
Ext-VTag = 5678

On reception of the ERROR message, Host A sends 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, Ext-VTag = 5678]
10.0.0.1:1 --------> 203.0.113.129:2
Ext-VTag = 5678

NAT
+---------+--------+----------+--------+-----------+
| VTag   | Port   | VTag     | Port   | Priv Addr |
| 1234   | 1      | 5678     | 2      | 10.0.0.1  |
+---------+--------+----------+--------+-----------+

ASCONF [ADD-IP,DELETE-IP,Int-VTag=1234, Ext-VTag = 5678]
192.0.2.2:1 -----------------> 203.0.113.129:2
Ext-VTag = 5678

Host B adds the new source address and deletes all former entries.
7.5. Peer-to-Peer Communication

If two hosts are behind NAT devices, they have to get knowledge of the peer’s public address. This can be achieved with a so-called rendezvous server. Afterwards the destination addresses are public, and the association is set up with the help of the INIT collision. The NAT devices create their entries according to their internal peer’s point of view. Therefore, NAT A’s Internal-VTag and Internal-Port are NAT B’s External-VTag and External-Port, respectively. The naming of the verification tag in the packet flow is done from the sending peer’s point of view.
INIT[Initiate-Tag = 1234]
10.0.0.1:1 --> 203.0.113.1:2
Ext-VTag = 0

NAT A creates entry:

+---------+--------+----------+--------+-----------+
| VTag    | Port   | Ext VTag | Port   | Addr      |
+---------+--------+----------+--------+-----------+
| 1234    | 1      | 0        | 2      | 10.0.0.1  |
+---------+--------+----------+--------+-----------+

INIT[Initiate-Tag = 1234]
192.0.2.1:1 ----------------> 203.0.113.1:2
Ext-VTag = 0

NAT B processes INIT, but cannot find an entry. The SCTP packet is silently discarded and leaves the NAT table of NAT B unchanged.
Now Host B sends INIT, which is processed by NAT B. Its parameters are used to create an entry.

```
<table>
<thead>
<tr>
<th>Internal</th>
<th>External</th>
<th>External</th>
<th>Internal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host A</td>
<td>NAT A</td>
<td>Internet</td>
<td>NAT B</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
</tbody>
</table>
```

INIT[Initiate-Tag = 5678]

```
192.0.2.1:1 <-- 10.1.0.1:2
Ext-VTag = 0
```

NAT A processes INIT. As the outgoing INIT of Host A has already created an entry, the entry is found and updated:

```
NAT B                +---------+--------+-----------+----------+--------+
<table>
<thead>
<tr>
<th>VTag</th>
<th>Port</th>
<th>Priv Addr</th>
<th>Ext VTag</th>
<th>Ext Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>5678</td>
<td>2</td>
<td>10.1.0.1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
```

INIT[Initiate-Tag = 5678]

```
192.0.2.1:1 <--------------- 203.0.113.1:2
Ext-VTag = 0
```
VTag != Int-VTag, but Ext-VTag == 0, find entry.

NAT A  |  Int    |  Int   |   Ext    |   Ext  |    Priv   |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VTag</td>
<td>Port</td>
<td>VTag</td>
<td>Port</td>
<td>Addr</td>
<td></td>
</tr>
<tr>
<td>1234</td>
<td>1</td>
<td>5678</td>
<td>2</td>
<td>10.0.0.1</td>
<td></td>
</tr>
</tbody>
</table>

INIT[Initiate-tag = 5678]
10.0.0.1:1 <-- 203.0.113.1:2
Ext-VTag = 0

Host A send INIT-ACK, which can pass through NAT B:
INIT-ACK[Initiate-Tag = 1234]
10.0.0.1:1 --> 203.0.113.1:2
Ext-VTag = 5678

INIT-ACK[Initiate-Tag = 1234]
192.0.2.1:1 --> 203.0.113.1:2
Ext-VTag = 5678

NAT B updates entry:

<table>
<thead>
<tr>
<th>NAT B</th>
<th>Int VTag</th>
<th>Int Port</th>
<th>Ext VTag</th>
<th>Ext Port</th>
<th>Priv Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5678</td>
<td>2</td>
<td>1234</td>
<td>1</td>
<td>10.1.0.1</td>
</tr>
</tbody>
</table>

INIT-ACK[Initiate-Tag = 1234]
192.0.2.1:1 --> 10.1.0.1:2
Ext-VTag = 5678

The lookup for COOKIE-ECHO and COOKIE-ACK is successful.
8. 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.
8.1. 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.

```c
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 may 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.

9. 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.

9.1. 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

<table>
<thead>
<tr>
<th>Chunk Flag Value</th>
<th>Chunk Flag Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>T bit</td>
<td>[RFCXXXX]</td>
</tr>
<tr>
<td>0x02</td>
<td>M bit</td>
<td>[RFCXXXX]</td>
</tr>
<tr>
<td>0x04</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>0x08</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>0x10</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>0x20</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>0x40</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>0x80</td>
<td>Unassigned</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Chunk Flag Value</th>
<th>Chunk Flag Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>T bit</td>
<td>[RFC4960]</td>
</tr>
<tr>
<td>0x02</td>
<td>M bit</td>
<td>[RFCXXXX]</td>
</tr>
<tr>
<td>0x04</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>0x08</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>0x10</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>0x20</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>0x40</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>0x80</td>
<td>Unassigned</td>
<td></td>
</tr>
</tbody>
</table>

9.2. 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

<table>
<thead>
<tr>
<th>Value</th>
<th>Cause Code</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>176</td>
<td>VTag and Port Number Collision</td>
<td>[RFCXXXX]</td>
</tr>
<tr>
<td>177</td>
<td>Missing State</td>
<td>[RFCXXXX]</td>
</tr>
<tr>
<td>178</td>
<td>Port Number Collision</td>
<td>[RFCXXXX]</td>
</tr>
</tbody>
</table>

9.3. Two New Chunk Parameter Types

Two chunk parameter types have to be assigned by IANA. It is requested to use the values given below. IANA should assign these values from the pool of parameters with the upper two bits set to '11'.

This requires two additional lines in the "Chunk Parameter Types" registry for SCTP:

<table>
<thead>
<tr>
<th>ID Value</th>
<th>Chunk Parameter Type</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>49159</td>
<td>Disable Restart (0xC007)</td>
<td>[RFCXXXX]</td>
</tr>
<tr>
<td>49160</td>
<td>VTags (0xC008)</td>
<td>[RFCXXXX]</td>
</tr>
</tbody>
</table>

10. Security Considerations

State maintenance within a NAT is always a subject of possible Denial Of Service attacks. This document recommends that at a minimum a NAT runs a timer on any SCTP state so that old association state can be cleaned up.

For SCTP endpoints, this document does not add any additional security considerations to the ones given in [RFC4960], [RFC4895], and [RFC5061]. In particular, SCTP is protected by the verification tags and the usage of [RFC4895] against off-path attackers.

11. Acknowledgments

The authors wish to thank Gorry Fairhurst, Bryan Ford, David Hayes, Alfred Hines, Karen E. E. Nielsen, Henning Peters, Timo Voelker, 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.

12. References

12.1. Normative References


12.2. Informative References


Authors’ Addresses

Randall R. Stewart  
Netflix, Inc.  
Chapin, SC 29036  
US  
Email: randall@lakerest.net

Michael Tuexen  
Muenster University of Applied Sciences  
Stegerwaldstrasse 39  
48565 Steinfurt  
DE  
Email: tuexen@fh-muenster.de

Irene Ruengeler  
Muenster University of Applied Sciences  
Stegerwaldstrasse 39  
48565 Steinfurt  
DE  
Email: i.ruengeler@fh-muenster.de
Abstract

This document obsoletes RFC 4960, if approved. It describes the Stream Control Transmission Protocol (SCTP). SCTP is designed to transport Public Switched Telephone Network (PSTN) signaling messages over IP networks, but is capable of broader applications.

SCTP is a reliable transport protocol operating on top of a connectionless packet network such as IP. It offers the following services to its users:

- acknowledged error-free non-duplicated transfer of user data,
- data fragmentation to conform to discovered path MTU size,
- sequenced delivery of user messages within multiple streams, with an option for order-of-arrival delivery of individual user messages,
- optional bundling of multiple user messages into a single SCTP packet, and
- network-level fault tolerance through supporting of multi-homing at either or both ends of an association.

The design of SCTP includes appropriate congestion avoidance behavior and resistance to flooding and masquerade attacks.

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 27, 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.

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 .................................................... 5
  1.1. Motivation ................................................... 6
  1.2. Architectural View of SCTP ................................. 6
  1.3. Key Terms .................................................... 7
  1.4. Abbreviations ............................................... 11
  1.5. Functional View of SCTP .................................. 11
       1.5.1. Association Startup and Takedown .................. 12
       1.5.2. Sequenced Delivery within Streams ............... 13
       1.5.3. User Data Fragmentation .......................... 13
1.5.4. Acknowledgement and Congestion Avoidance .......... 13
1.5.5. Chunk Bundling ........................................ 14
1.5.6. Packet Validation ...................................... 14
1.5.7. Path Management ....................................... 14
1.6. Serial Number Arithmetic ................................. 15
1.7. Changes from RFC 4960 .................................... 16
2. Conventions .................................................. 16
3. SCTP Packet Format .......................................... 16
  3.1. SCTP Common Header Field Descriptions ................ 17
  3.2. Chunk Field Descriptions ................................ 18
    3.2.1. Optional/Variable-Length Parameter Format ........ 20
    3.2.2. Reporting of Unrecognized Parameters ............. 22
  3.3. SCTP Chunk Definitions .................................. 22
    3.3.1. Payload Data (DATA) (0) ............................. 22
    3.3.2. Initiation (INIT) (1) ............................... 25
      3.3.2.1. Optional/Variable-Length Parameters in INIT .... 28
    3.3.3. Initiation Acknowledgement (INIT ACK) (2) .......... 31
      3.3.3.1. Optional or Variable-Length Parameters .......... 34
    3.3.4. Selective Acknowledgement (SACK) (3) ................ 35
    3.3.5. Heartbeat Request (HEARTBEAT) (4) .................. 39
    3.3.6. Heartbeat Acknowledgement (HEARTBEAT ACK) (5) .... 40
    3.3.7. Abort Association (ABORT) (6) ....................... 41
    3.3.8. Shutdown Association (SHUTDOWN) (7) ............... 42
    3.3.9. Shutdown Acknowledgement (SHUTDOWN ACK) (8) ...... 42
      3.3.10. Operation Error (ERROR) (9) ...................... 43
        3.3.10.1. Invalid Stream Identifier (1) ................ 44
        3.3.10.2. Missing Mandatory Parameter (2) ............. 45
        3.3.10.3. Stale Cookie Error (3) ....................... 45
        3.3.10.4. Out of Resource (4) ........................... 46
        3.3.10.5. Unresolvable Address (5) ...................... 46
        3.3.10.6. Unrecognized Chunk Type (6) .................. 46
        3.3.10.7. Invalid Mandatory Parameter (7) ............. 47
        3.3.10.8. Unrecognized Parameters (8) .................. 47
        3.3.10.9. No User Data (9) .............................. 48
        3.3.10.10. Cookie Received While Shutting Down (10) .... 48
      3.3.10.11. Restart of an Association with New Addresses (11) .... 49
    3.3.10.12. User-Initiated Abort (12) ...................... 49
    3.3.10.13. Protocol Violation (13) ......................... 49
    3.3.11. Cookie Echo (COOKIE ECHO) (10) .................... 50
    3.3.12. Cookie Acknowledgement (COOKIE ACK) (11) .......... 51
    3.3.13. Shutdown Complete (SHUTDOWN COMPLETE) (14) ....... 51
4. SCTP Association State Diagram ............................ 52
5. Association Initialization .................................... 55
  5.1. Normal Establishment of an Association ................ 55
    5.1.1. Handle Stream Parameters ............................ 57
    5.1.2. Handle Address Parameters ........................... 58
    5.1.3. Generating State Cookie ............................. 59
5.1.4. State Cookie Processing ............................ 60
5.1.5. State Cookie Authentication .......................... 60
5.1.6. An Example of Normal Association Establishment ....... 61
5.2. Handle Duplicate or Unexpected INIT, INIT ACK, COOKIE
       ECHO, and COOKIE ACK ............................... 63
5.2.1. INIT Received in COOKIE-WAIT or COOKIE-ECHOED State
       (Item B) ........................................... 63
5.2.2. Unexpected INIT in States Other than CLOSED, COOKIE-
       ECHOED, COOKIE-WAIT, and SHUTDOWN-ACK-SENT ......... 64
5.2.3. Unexpected INIT ACK ................................ 65
5.2.4. Handle a COOKIE ECHO when a TCB Exists ............. 65
5.2.4.1. An Example of a Association Restart .............. 67
5.2.5. Handle Duplicate COOKIE ACK .......................... 69
5.2.6. Handle Stale COOKIE Error ........................... 69
5.3. Other Initialization Issues ............................ 69
5.3.1. Selection of Tag Value ............................... 69
5.4. Path Verification ..................................... 70
6. User Data Transfer ...................................... 71
6.1. Transmission of DATA Chunks ......................... 73
6.2. Acknowledgement on Reception of DATA Chunks .......... 76
6.2.1. Processing a Received SACK ........................ 79
6.3. Management of Retransmission Timer ................... 81
6.3.1. RTO Calculation .................................. 81
6.3.2. Retransmission Timer Rules ........................ 82
6.3.3. Handle T3-rtx Expiration .......................... 83
6.4. Multi-Homed SCTP Endpoints ........................... 84
6.4.1. Failover from an Inactive Destination Address ....... 86
6.5. Stream Identifier and Stream Sequence Number .......... 86
6.6. Ordered and Unordered Delivery ........................ 86
6.7. Report Gaps in Received DATA TSNs .................... 87
6.8. CRC32c Checksum Calculation .......................... 88
6.9. Fragmentation and Reassembly .......................... 89
6.10. Bundling ........................................... 90
7. Congestion Control ....................................... 91
7.1. SCTP Differences from TCP Congestion Control ......... 92
7.2. SCTP Slow-Start and Congestion Avoidance ............. 93
7.2.1. Slow-Start ...................................... 94
7.2.2. Congestion Avoidance .............................. 95
7.2.3. Congestion Control ................................ 96
7.2.4. Fast Retransmit on Gap Reports ........................ 96
7.2.5. Making Changes to Differentiated Services Code Points 97
7.3. Path MTU Discovery .................................... 98
8. Fault Management ........................................ 98
8.1. Endpoint Failure Detection ............................ 98
8.2. Path Failure Detection ................................. 99
8.3. Path Heartbeat ....................................... 100
8.4. Handle "Out of the Blue" Packets ...................... 102
8.5. Verification Tag ...................................... 103
1. Introduction

This section explains the reasoning behind the development of the Stream Control Transmission Protocol (SCTP), the services it offers, and the basic concepts needed to understand the detailed description of the protocol.
1.1. Motivation

TCP [RFC0793] has performed immense service as the primary means of reliable data transfer in IP networks. However, an increasing number of recent applications have found TCP too limiting, and have incorporated their own reliable data transfer protocol on top of UDP [RFC0768]. The limitations that users have wished to bypass include the following:

- TCP provides both reliable data transfer and strict order-of-transmission delivery of data. Some applications need reliable transfer without sequence maintenance, while others would be satisfied with partial ordering of the data. In both of these cases, the head-of-line blocking offered by TCP causes unnecessary delay.

- The stream-oriented nature of TCP is often an inconvenience. Applications add their own record marking to delineate their messages, and make explicit use of the push facility to ensure that a complete message is transferred in a reasonable time.

- The limited scope of TCP sockets complicates the task of providing highly-available data transfer capability using multi-homed hosts.

- TCP is relatively vulnerable to denial-of-service attacks, such as SYN attacks.

Transport of PSTN signaling across the IP network is an application for which all of these limitations of TCP are relevant. While this application directly motivated the development of SCTP, other applications might find SCTP a good match to their requirements.

1.2. Architectural View of SCTP

SCTP is viewed as a layer between the SCTP user application ("SCTP user" for short) and a connectionless packet network service such as IP. The remainder of this document assumes SCTP runs on top of IP. The basic service offered by SCTP is the reliable transfer of user messages between peer SCTP users. It performs this service within the context of an association between two SCTP endpoints. Section 10 of this document sketches the API that exists at the boundary between the SCTP and the SCTP user layers.

SCTP is connection-oriented in nature, but the SCTP association is a broader concept than the TCP connection. SCTP provides the means for each SCTP endpoint (Section 1.3) to provide the other endpoint...
during association startup) with a list of transport addresses (i.e., multiple IP addresses in combination with an SCTP port) through which that endpoint can be reached and from which it will originate SCTP packets. The association spans transfers over all of the possible source/destination combinations that can be generated from each endpoint’s lists.

![Figure 1: An SCTP Association](image)

1.3. Key Terms

Some of the language used to describe SCTP has been introduced in the previous sections. This section provides a consolidated list of the key terms and their definitions.

**Active destination transport address:** A transport address on a peer endpoint that a transmitting endpoint considers available for receiving user messages.

**Bundling:** An optional multiplexing operation, whereby more than one user message might be carried in the same SCTP packet. Each user message occupies its own DATA chunk.

**Chunk:** A unit of information within an SCTP packet, consisting of a chunk header and chunk-specific content.

**Congestion window (cwnd):** An SCTP variable that limits outstanding data, in number of bytes, that a sender can send to a particular destination transport address before receiving an acknowledgement.
Cumulative TSN Ack Point: The Transmission Sequence Number (TSN) of the last DATA chunk acknowledged via the Cumulative TSN Ack field of a SACK.

Flightsize: The amount of bytes of outstanding data to a particular destination transport address at any given time.

Idle destination address: An address that has not had user messages sent to it within some length of time, normally the HEARTBEAT interval or greater.

Inactive destination transport address: An address that is considered inactive due to errors and unavailable to transport user messages.

Message = user message: Data submitted to SCTP by the Upper Layer Protocol (ULP).

Message Authentication Code (MAC): An integrity check mechanism based on cryptographic hash functions using a secret key. Typically, message authentication codes are used between two parties that share a secret key in order to validate information transmitted between these parties. In SCTP, it is used by an endpoint to validate the State Cookie information that is returned from the peer in the COOKIE ECHO chunk. The term "MAC" has different meanings in different contexts. SCTP uses this term with the same meaning as in [RFC2104].

Network Byte Order: Most significant byte first, a.k.a., big endian.

Ordered Message: A user message that is delivered in order with respect to all previous user messages sent within the stream on which the message was sent.

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 that is classified as lost but that has not yet been retransmitted is not in outstanding data.

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.

Path: The route taken by the SCTP packets sent by one SCTP endpoint to a specific destination transport address of its peer SCTP endpoint. Sending to different destination transport addresses does not necessarily guarantee getting separate paths.
Primary Path: The primary path is the destination and source address that will be put into a packet outbound to the peer endpoint by default. The definition includes the source address since an implementation MAY wish to specify both destination and source address to better control the return path taken by reply chunks and on which interface the packet is transmitted when the data sender is multi-homed.

Receiver Window (rwnd): An SCTP variable a data sender uses to store the most recently calculated receiver window of its peer, in number of bytes. This gives the sender an indication of the space available in the receiver’s inbound buffer.

SCTP association: A protocol relationship between SCTP endpoints, composed of the two SCTP endpoints and protocol state information including Verification Tags and the currently active set of Transmission Sequence Numbers (TSNs), etc. An association can be uniquely identified by the transport addresses used by the endpoints in the association. Two SCTP endpoints MUST NOT have more than one SCTP association between them at any given time.

SCTP endpoint: The logical sender/receiver of SCTP packets. On a multi-homed host, an SCTP endpoint is represented to its peers as a combination of a set of eligible destination transport addresses to which SCTP packets can be sent and a set of eligible source transport addresses from which SCTP packets can be received. All transport addresses used by an SCTP endpoint MUST use the same port number, but can use multiple IP addresses. A transport address used by an SCTP endpoint MUST NOT be used by another SCTP endpoint. In other words, a transport address is unique to an SCTP endpoint.

SCTP packet (or packet): The unit of data delivery across the interface between SCTP and the connectionless packet network (e.g., IP). An SCTP packet includes the common SCTP header, possible SCTP control chunks, and user data encapsulated within SCTP DATA chunks.

SCTP user application (SCTP user): The logical higher-layer application entity which uses the services of SCTP, also called the Upper-Layer Protocol (ULP).

Slow-Start Threshold (ssthresh): An SCTP variable. This is the threshold that the endpoint will use to determine whether to perform slow start or congestion avoidance on a particular destination transport address. Ssthresh is in number of bytes.
Stream: A unidirectional logical channel established from one to another associated SCTP endpoint, within which all user messages are delivered in sequence except for those submitted to the unordered delivery service.

Note: The relationship between stream numbers in opposite directions is strictly a matter of how the applications use them. It is the responsibility of the SCTP user to create and manage these correlations if they are so desired.

Stream Sequence Number: A 16-bit sequence number used internally by SCTP to ensure sequenced delivery of the user messages within a given stream. One Stream Sequence Number is attached to each user message.

Tie-Tags: Two 32-bit random numbers that together make a 64-bit nonce. These tags are used within a State Cookie and TCB so that a newly restarting association can be linked to the original association within the endpoint that did not restart and yet not reveal the true Verification Tags of an existing association.

Transmission Control Block (TCB): An internal data structure created by an SCTP endpoint for each of its existing SCTP associations to other SCTP endpoints. TCB contains all the status and operational information for the endpoint to maintain and manage the corresponding association.

Transmission Sequence Number (TSN): A 32-bit sequence number used internally by SCTP. One TSN is attached to each chunk containing user data to permit the receiving SCTP endpoint to acknowledge its receipt and detect duplicate deliveries.

Transport address: A transport address is traditionally defined by a network-layer address, a transport-layer protocol, and a transport-layer port number. In the case of SCTP running over IP, a transport address is defined by the combination of an IP address and an SCTP port number (where SCTP is the transport protocol).

Unacknowledged TSN (at an SCTP endpoint): A TSN (and the associated DATA chunk) that has been received by the endpoint but for which an acknowledgement has not yet been sent. Or in the opposite case, for a packet that has been sent but no acknowledgement has been received.

Unordered Message: Unordered messages are "unordered" with respect to any other message; this includes both other unordered messages as well as other ordered messages. An unordered message might be
delivered prior to or later than ordered messages sent on the same stream.

User message: The unit of data delivery across the interface between SCTP and its user.

Verification Tag: A 32-bit unsigned integer that is randomly generated. The Verification Tag provides a key that allows a receiver to verify that the SCTP packet belongs to the current association and is not an old or stale packet from a previous association.

1.4. Abbreviations

MAC  Message Authentication Code [RFC2104]
RTO  Retransmission Timeout
RTT  Round-Trip Time
RTTVAR Round-Trip Time Variation
SCTP Stream Control Transmission Protocol
SRTT Smoothed RTT
TCB Transmission Control Block
TLV Type-Length-Value coding format
TSN Transmission Sequence Number
ULP Upper-Layer Protocol

1.5. Functional View of SCTP

The SCTP transport service can be decomposed into a number of functions. These are depicted in Figure 2 and explained in the remainder of this section.
Figure 2: Functional View of the SCTP Transport Service

1.5.1. Association Startup and Takedown

An association is initiated by a request from the SCTP user (see the
description of the ASSOCIATE (or SEND) primitive in Section 10).

A cookie mechanism, similar to one described by Karn and Simpson in
[RFC2522], is employed during the initialization to provide
protection against synchronization attacks. The cookie mechanism
uses a four-way handshake, the last two legs of which are allowed to
carry user data for fast setup. The startup sequence is described in
Section 5 of this document.

SCTP provides for graceful close (i.e., shutdown) of an active
association on request from the SCTP user. See the description of
the SHUTDOWN primitive in Section 10. SCTP also allows ungraceful
close (i.e., abort), either on request from the user (ABORT primitive) or as a result of an error condition detected within the SCTP layer. Section 9 describes both the graceful and the ungraceful close procedures.

SCTP does not support a half-open state (like TCP) wherein one side continues sending data while the other end is closed. When either endpoint performs a shutdown, the association on each peer will stop accepting new data from its user and only deliver data in queue at the time of the graceful close (see Section 9).

1.5.2. Sequenced Delivery within Streams

The term "stream" is used in SCTP to refer to a sequence of user messages that are to be delivered to the upper-layer protocol in order with respect to other messages within the same stream. This is in contrast to its usage in TCP, where it refers to a sequence of bytes (in this document, a byte is assumed to be 8 bits).

The SCTP user can specify at association startup time the number of streams to be supported by the association. This number is negotiated with the remote end (see Section 5.1.1). User messages are associated with stream numbers (SEND, RECEIVE primitives, Section 10). Internally, SCTP assigns a Stream Sequence Number to each message passed to it by the SCTP user. On the receiving side, SCTP ensures that messages are delivered to the SCTP user in sequence within a given stream. However, while one stream might be blocked waiting for the next in-sequence user message, delivery from other streams might proceed.

SCTP provides a mechanism for bypassing the sequenced delivery service. User messages sent using this mechanism are delivered to the SCTP user as soon as they are received.

1.5.3. User Data Fragmentation

When needed, SCTP fragments user messages to ensure that the SCTP packet passed to the lower layer conforms to the path MTU. On receipt, fragments are reassembled into complete messages before being passed to the SCTP user.

1.5.4. Acknowledgement and Congestion Avoidance

SCTP assigns a Transmission Sequence Number (TSN) to each user data fragment or unfragmented message. The TSN is independent of any Stream Sequence Number assigned at the stream level. The receiving end acknowledges all TSNs received, even if there are gaps in the
sequence. In this way, reliable delivery is kept functionally separate from sequenced stream delivery.

The acknowledgement and congestion avoidance function is responsible for packet retransmission when timely acknowledgement has not been received. Packet retransmission is conditioned by congestion avoidance procedures similar to those used for TCP. See Section 6 and Section 7 for a detailed description of the protocol procedures associated with this function.

1.5.5. Chunk Bundling

As described in Section 3, the SCTP packet as delivered to the lower layer consists of a common header followed by one or more chunks. Each chunk might contain either user data or SCTP control information. The SCTP user has the option to request bundling of more than one user message into a single SCTP packet. The chunk bundling function of SCTP is responsible for assembly of the complete SCTP packet and its disassembly at the receiving end.

During times of congestion, an SCTP implementation MAY still perform bundling even if the user has requested that SCTP not bundle. The user’s disabling of bundling only affects SCTP implementations that might delay a small period of time before transmission (to attempt to encourage bundling). When the user layer disables bundling, this small delay is prohibited but not bundling that is performed during congestion or retransmission.

1.5.6. Packet Validation

A mandatory Verification Tag field and a 32-bit checksum field (see Appendix B for a description of the CRC32c checksum) are included in the SCTP common header. The Verification Tag value is chosen by each end of the association during association startup. Packets received without the expected Verification Tag value are discarded, as a protection against blind masquerade attacks and against stale SCTP packets from a previous association. The CRC32c checksum can be set by the sender of each SCTP packet to provide additional protection against data corruption in the network. The receiver of an SCTP packet with an invalid CRC32c checksum silently discards the packet.

1.5.7. Path Management

The sending SCTP user is able to manipulate the set of transport addresses used as destinations for SCTP packets through the primitives described in Section 10. The SCTP path management function chooses the destination transport address for each outgoing SCTP packet based on the SCTP user’s instructions and the currently
perceived reachability status of the eligible destination set. The path management function monitors reachability through heartbeats when other packet traffic is inadequate to provide this information and advises the SCTP user when reachability of any far-end transport address changes. The path management function is also responsible for reporting the eligible set of local transport addresses to the far end during association startup, and for reporting the transport addresses returned from the far end to the SCTP user.

At association startup, a primary path is defined for each SCTP endpoint, and is used for normal sending of SCTP packets.

On the receiving end, the path management is responsible for verifying the existence of a valid SCTP association to which the inbound SCTP packet belongs before passing it for further processing.

Note: Path Management and Packet Validation are done at the same time, so although described separately above, in reality they cannot be performed as separate items.

1.6. Serial Number Arithmetic

It is essential to remember that the actual Transmission Sequence Number space is finite, though very large. This space ranges from 0 to 2**32 - 1. Since the space is finite, all arithmetic dealing with Transmission Sequence Numbers MUST be performed modulo 2**32. This unsigned arithmetic preserves the relationship of sequence numbers as they cycle from 2**32 - 1 to 0 again. There are some subtleties to computer modulo arithmetic, so great care has to be taken in programming the comparison of such values. When referring to TSNs, the symbol "=<" means "less than or equal" (modulo 2**32).

Comparisons and arithmetic on TSNs in this document SHOULD use Serial Number Arithmetic as defined in [RFC1982] where SERIAL_BITS = 32.

An endpoint SHOULD NOT transmit a DATA chunk with a TSN that is more than 2**31 - 1 above the beginning TSN of its current send window. Doing so will cause problems in comparing TSNs.

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.

Any arithmetic done on Stream Sequence Numbers SHOULD use Serial Number Arithmetic as defined in [RFC1982] where SERIAL_BITS = 16. All other arithmetic and comparisons in this document use normal arithmetic.
1.7. Changes from RFC 4960

SCTP was originally defined in [RFC4960], which this document obsoletes, if approved. Readers interested in the details of the various changes that this document incorporates are asked to consult [RFC8540].

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. SCTP Packet Format

An SCTP packet is composed of a common header and chunks. A chunk contains either control information or user data.

The SCTP packet format is shown below:

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Common Header                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Chunk #1                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           ...                                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Chunk #n                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Multiple chunks can be bundled into one SCTP packet up to the MTU size, except for the INIT, INIT ACK, and SHUTDOWN COMPLETE chunks. These chunks MUST NOT be bundled with any other chunk in a packet. See Section 6.10 for more details on chunk bundling.

If a user data message doesn’t fit into one SCTP packet it can be fragmented into multiple chunks using the procedure defined in Section 6.9.

All integer fields in an SCTP packet MUST be transmitted in network byte order, unless otherwise stated.
3.1. SCTP Common Header Field Descriptions

SCTP Common Header Format

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Source Port Number        |     Destination Port Number   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Verification Tag                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Checksum                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Source Port Number: 16 bits (unsigned integer)

This is the SCTP sender’s port number. It can be used by the receiver in combination with the source IP address, the SCTP destination port, and possibly the destination IP address to identify the association to which this packet belongs. The port number 0 MUST NOT be used.

Destination Port Number: 16 bits (unsigned integer)

This is the SCTP port number to which this packet is destined. The receiving host will use this port number to de-multiplex the SCTP packet to the correct receiving endpoint/application. The port number 0 MUST NOT be used.

Verification Tag: 32 bits (unsigned integer)

The receiver of this packet uses the Verification Tag to validate the sender of this SCTP packet. On transmit, the value of this Verification Tag MUST be set to the value of the Initiate Tag received from the peer endpoint during the association initialization, with the following exceptions:

* A packet containing an INIT chunk MUST have a zero Verification Tag.

* A packet containing a SHUTDOWN COMPLETE chunk with the T bit set MUST have the Verification Tag copied from the packet with the SHUTDOWN ACK chunk.

* A packet containing an ABORT chunk MAY have the verification tag copied from the packet that caused the ABORT to be sent. For details see Section 8.4 and Section 8.5.
Checksum: 32 bits (unsigned integer)

This field contains the checksum of this SCTP packet. Its calculation is discussed in Section 6.8. SCTP uses the CRC32c algorithm as described in Appendix B for calculating the checksum.

3.2. Chunk Field Descriptions

The figure below illustrates the field format for the chunks to be transmitted in the SCTP packet. Each chunk is formatted with a Chunk Type field, a chunk-specific Flag field, a Chunk Length field, and a Value field.

```
+----------------+----------------+----------------+----------------+----------------+----------------+----------------+----------------+----------------+----------------+----------------+
| 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 2 3 4 5 6 7 8 9 0 1 |
+----------------+----------------+----------------+----------------+----------------+----------------+----------------+----------------+----------------+----------------+
| Chunk Type     | Chunk Flags    | Chunk Length   | Chunk Value    |
+----------------+----------------+----------------+----------------+
```

Chunk Type: 8 bits (unsigned integer)

This field identifies the type of information contained in the Chunk Value field. It takes a value from 0 to 254. The value of 255 is reserved for future use as an extension field.

The values of Chunk Types are defined as follows:

<table>
<thead>
<tr>
<th>ID Value</th>
<th>Chunk Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Payload Data (DATA)</td>
</tr>
<tr>
<td>1</td>
<td>Initiation (INIT)</td>
</tr>
<tr>
<td>2</td>
<td>Initiation Acknowledgement (INIT ACK)</td>
</tr>
<tr>
<td>3</td>
<td>Selective Acknowledgement (SACK)</td>
</tr>
<tr>
<td>4</td>
<td>Heartbeat Request (HEARTBEAT)</td>
</tr>
<tr>
<td>5</td>
<td>Heartbeat Acknowledgement (HEARTBEAT ACK)</td>
</tr>
<tr>
<td>6</td>
<td>Abort (ABORT)</td>
</tr>
<tr>
<td>7</td>
<td>Shutdown (SHUTDOWN)</td>
</tr>
<tr>
<td>8</td>
<td>Shutdown Acknowledgement (SHUTDOWN ACK)</td>
</tr>
<tr>
<td>9</td>
<td>Operation Error (ERROR)</td>
</tr>
<tr>
<td>10</td>
<td>State Cookie (COOKIE ECHO)</td>
</tr>
<tr>
<td>11</td>
<td>Cookie Acknowledgement (COOKIE ACK)</td>
</tr>
<tr>
<td>12</td>
<td>Reserved for Explicit Congestion Notify</td>
</tr>
<tr>
<td></td>
<td>Echo (ECNE)</td>
</tr>
<tr>
<td>13</td>
<td>Reserved for Congestion Window Reduced (CWR)</td>
</tr>
</tbody>
</table>
Chunk Types are encoded such that the highest-order 2 bits specify the action that is taken if the processing endpoint does not recognize the Chunk Type.

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'.

10 - Skip this chunk and continue processing.

11 - Skip this chunk and continue processing, but report in an ERROR chunk using the 'Unrecognized Chunk Type' cause of error.

Note: The ECNE and CWR chunk types are reserved for future use of Explicit Congestion Notification (ECN); see Appendix A.

Chunk Flags: 8 bits

The usage of these bits depends on the Chunk type as given by the Chunk Type field. Unless otherwise specified, they are set to 0 on transmit and are ignored on receipt.

Chunk Length: 16 bits (unsigned integer)

This value represents the size of the chunk in bytes, including the Chunk Type, Chunk Flags, Chunk Length, and Chunk Value fields. Therefore, if the Chunk Value field is zero-length, the Length field will be set to 4. The Chunk Length field does not count any chunk padding.

Chunks (including Type, Length, and Value fields) are padded out by the sender with all zero bytes to be a multiple of 4 bytes long. This padding MUST NOT be more than 3 bytes in total. The Chunk Length value does not include terminating padding of the
chunk. However, it does include padding of any variable-length parameter except the last parameter in the chunk. The receiver MUST ignore the padding.

Note: A robust implementation is expected to accept the chunk whether or not the final padding has been included in the Chunk Length.

Chunk Value: variable length

The Chunk Value field contains the actual information to be transferred in the chunk. The usage and format of this field is dependent on the Chunk Type.

The total length of a chunk (including Type, Length, and Value fields) MUST be a multiple of 4 bytes. If the length of the chunk is not a multiple of 4 bytes, the sender MUST pad the chunk with all zero bytes, and this padding is not included in the Chunk Length field. The sender MUST NOT pad with more than 3 bytes. The receiver MUST ignore the padding bytes.

SCTP-defined chunks are described in detail in Section 3.3. The guidelines for IETF-defined chunk extensions can be found in Section 14.1 of this document.

3.2.1. Optional/Variable-Length Parameter Format

Chunk values of SCTP control chunks consist of a chunk-type-specific header of required fields, followed by zero or more parameters. The optional and variable-length parameters contained in a chunk are defined in a Type-Length-Value format as shown below.

```
 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
+-----------------------------------------------+
| Parameter Type | Parameter Length |
+-----------------------------------------------+
| Parameter Value                                      |
+-----------------------------------------------+
```

Chunk Parameter Type: 16 bits (unsigned integer)

The Type field is a 16-bit identifier of the type of parameter. It takes a value of 0 to 65534.
The value of 65535 is reserved for IETF-defined extensions. Values other than those defined in specific SCTP chunk descriptions are reserved for use by IETF.

Chunk Parameter Length: 16 bits (unsigned integer)

The Parameter Length field contains the size of the parameter in bytes, including the Parameter Type, Parameter Length, and Parameter Value fields. Thus, a parameter with a zero-length Parameter Value field would have a Length field of 4. The Parameter Length does not include any padding bytes.

Chunk Parameter Value: variable length

The Parameter Value field contains the actual information to be transferred in the parameter.

The total length of a parameter (including Type, Parameter Length, and Value fields) MUST be a multiple of 4 bytes. If the length of the parameter is not a multiple of 4 bytes, the sender pads the parameter at the end (i.e., after the Parameter Value field) with all zero bytes. The length of the padding is not included in the Parameter Length field. A sender MUST NOT pad with more than 3 bytes. The receiver MUST ignore the padding bytes.

The Parameter Types are encoded such that the highest-order 2 bits specify the action that is taken if the processing endpoint does not recognize the Parameter Type.

00 - Stop processing this parameter; do not process any further parameters within this chunk.

01 - Stop processing this parameter, do not process any further parameters within this chunk, and report the unrecognized parameter in an ‘Unrecognized Parameter’, as described in Section 3.2.2.

10 - Skip this parameter and continue processing.

11 - Skip this parameter and continue processing but report the unrecognized parameter in an ‘Unrecognized Parameter’, as described in Section 3.2.2.

Please note that in all four cases, an INIT ACK or COOKIE ECHO chunk is sent. In the 00 or 01 case, the processing of the parameters after the unknown parameter is canceled, but no processing already done is rolled back.
The actual SCTP parameters are defined in the specific SCTP chunk sections. The rules for IETF-defined parameter extensions are defined in Section 14.3. Note that a parameter type MUST be unique across all chunks. For example, the parameter type ‘5’ is used to represent an IPv4 address (see Section 3.3.2.1). The value ‘5’ then is reserved across all chunks to represent an IPv4 address and MUST NOT be reused with a different meaning in any other chunk.

3.2.2. Reporting of Unrecognized Parameters

If the receiver of an INIT chunk detects unrecognized parameters and has to report them according to Section 3.2.1, it MUST put the 'Unrecognized Parameter' parameter(s) in the INIT ACK chunk sent in response to the INIT chunk. Note that if the receiver of the INIT chunk is NOT going to establish an association (e.g., due to lack of resources), an 'Unrecognized Parameter' would NOT be included with any ABORT being sent to the sender of the INIT.

If the receiver of an INIT ACK chunk detects unrecognized parameters and has to report them according to Section 3.2.1, it SHOULD bundle the ERROR chunk containing the 'Unrecognized Parameters' error cause with the COOKIE ECHO chunk sent in response to the INIT ACK chunk. If the receiver of the INIT ACK cannot bundle the COOKIE ECHO chunk with the ERROR chunk, the ERROR chunk MAY be sent separately but not before the COOKIE ACK has been received.

Note: Any time a COOKIE ECHO is sent in a packet, it MUST be the first chunk.

3.3. SCTP Chunk Definitions

This section defines the format of the different SCTP chunk types.

3.3.1. Payload Data (DATA) (0)

The following format MUST be used for the DATA chunk:
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 Section 4 of [RFC7053] for a
discussion of the benefits.

U bit: 1 bit

The (U)nordered bit, if set to ‘1’, indicates that this is an
unordered DATA chunk, and there is no Stream Sequence Number
assigned to this DATA chunk. Therefore, the receiver MUST ignore
the Stream Sequence Number field.

After reassembly (if necessary), unordered DATA chunks MUST be
dispatched to the upper layer by the receiver without any attempt
to reorder.

If an unordered user message is fragmented, each fragment of the
message MUST have its U bit set to ‘1’.

B bit: 1 bit

The (B)eginning fragment bit, if set, indicates the first fragment
of a user message.

E bit: 1 bit
The (E)nding fragment bit, if set, indicates the last fragment of a user message.

An unfragmented user message MUST have both the B and E bits set to ‘1’. Setting both B and E bits to ‘0’ indicates a middle fragment of a multi-fragment user message, as summarized in the following table:

<table>
<thead>
<tr>
<th>B</th>
<th>E</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>First piece of a fragmented user message</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Middle piece of a fragmented user message</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Last piece of a fragmented user message</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Unfragmented message</td>
</tr>
</tbody>
</table>

Table 1: Fragment Description Flags

When a user message is fragmented into multiple chunks, the TSNs are used by the receiver to reassemble the message. This means that the TSNs for each fragment of a fragmented user message MUST be strictly sequential.

Length: 16 bits (unsigned integer)

This field indicates the length of the DATA chunk in bytes from the beginning of the type field to the end of the User Data field excluding any padding. A DATA chunk with one byte of user data will have Length set to 17 (indicating 17 bytes).

A DATA chunk with a User Data field of length L will have the Length field set to (16 + L) (indicating 16 + L bytes) where L MUST be greater than 0.

TSN: 32 bits (unsigned integer)

This value represents the TSN for this DATA chunk. The valid range of TSN is from 0 to 4294967295 (2**32 - 1). TSN wraps back to 0 after reaching 4294967295.

Stream Identifier S: 16 bits (unsigned integer)

Identifies the stream to which the following user data belongs.

Stream Sequence Number n: 16 bits (unsigned integer)
This value represents the Stream Sequence Number of the following user data within the stream S. Valid range is 0 to 65535.

When a user message is fragmented by SCTP for transport, the same Stream Sequence Number MUST be carried in each of the fragments of the message.

Payload Protocol Identifier: 32 bits (unsigned integer)

This value represents an application (or upper layer) specified protocol identifier. This value is passed to SCTP by its upper layer and sent to its peer. This identifier is not used by SCTP but can be used by certain network entities, as well as by the peer application, to identify the type of information being carried in this DATA chunk. This field MUST be sent even in fragmented DATA chunks (to make sure it is available for agents in the middle of the network). Note that this field is NOT touched by an SCTP implementation; therefore, its byte order is NOT necessarily big endian. The upper layer is responsible for any byte order conversions to this field.

The value 0 indicates that no application identifier is specified by the upper layer for this payload data.

User Data: variable length

This is the payload user data. The implementation MUST pad the end of the data to a 4-byte boundary with all-zero bytes. Any padding MUST NOT be included in the Length field. A sender MUST never add more than 3 bytes of padding.

3.3.2. Initiation (INIT) (1)

This chunk is used to initiate an SCTP association between two endpoints. The format of the INIT chunk is shown below:
The INIT chunk contains the following parameters. Unless otherwise noted, each parameter MUST only be included once in the INIT chunk.

<table>
<thead>
<tr>
<th>Fixed Parameters</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initiate Tag</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Advertised Receiver Window Credit</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Number of Outbound Streams</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Number of Inbound Streams</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Initial TSN</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable Parameters</th>
<th>Status</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv4 Address (Note 1)</td>
<td>Optional</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>IPv6 Address (Note 1)</td>
<td>Optional</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Cookie Preservative</td>
<td>Optional</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Reserved for ECN Capable (Note 2)</td>
<td>Optional</td>
<td>32768 (0x8000)</td>
<td></td>
</tr>
<tr>
<td>Host Name Address (Note 3)</td>
<td>Optional</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>Supported Address Types (Note 4)</td>
<td>Optional</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

Note 1: The INIT chunks can contain multiple addresses that can be IPv4 and/or IPv6 in any combination.

Note 2: The ECN Capable field is reserved for future use of Explicit Congestion Notification.

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 "Unresolvable Address" error cause.
Note 4: This parameter, when present, specifies all the address types the sending endpoint can support. The absence of this parameter indicates that the sending endpoint can support any address type.

IMPLEMENTATION NOTE: If an INIT chunk is received with known parameters that are not optional parameters of the INIT chunk, then the receiver SHOULD process the INIT chunk and send back an INIT ACK. The receiver of the INIT chunk MAY bundle an ERROR chunk with the COOKIE ACK chunk later. However, restrictive implementations MAY send back an ABORT chunk in response to the INIT chunk.

The Chunk Flags field in INIT is reserved, and all bits in it SHOULD be set to 0 by the sender and ignored by the receiver. The sequence of parameters within an INIT can be processed in any order.

Initiate Tag: 32 bits (unsigned integer)

The receiver of the INIT (the responding end) records the value of the Initiate Tag parameter. This value MUST be placed into the Verification Tag field of every SCTP packet that the receiver of the INIT transmits within this association.

The Initiate Tag is allowed to have any value except 0. See Section 5.3.1 for more on the selection of the tag value.

If the value of the Initiate Tag in a received INIT chunk is found to be 0, the receiver MUST treat it as an error and close the association by transmitting an ABORT.

Advertised Receiver Window Credit (a_rwnd): 32 bits (unsigned integer)

This value represents the dedicated buffer space, in number of bytes, the sender of the INIT has reserved in association with this window. During the life of the association, this buffer space SHOULD NOT be lessened (i.e., dedicated buffers taken away from this association); however, an endpoint MAY change the value of a_rwnd it sends in SACK chunks.

Number of Outbound Streams (OS): 16 bits (unsigned integer)

Defines the number of outbound streams the sender of this INIT chunk wishes to create in this association. The value of 0 MUST NOT be used.

Note: A receiver of an INIT with the OS value set to 0 SHOULD abort the association.
Number of Inbound Streams (MIS): 16 bits (unsigned integer)

Defines the maximum number of streams the sender of this INIT chunk allows the peer end to create in this association. The value 0 MUST NOT be used.

Note: There is no negotiation of the actual number of streams but instead the two endpoints will use the min(requested, offered). See Section 5.1.1 for details.

Note: A receiver of an INIT with the MIS value of 0 SHOULD abort the association.

Initial TSN (I-TSN): 32 bits (unsigned integer)

Defines the initial TSN that the sender will use. The valid range is from 0 to 4294967295. This field MAY be set to the value of the Initiate Tag field.

3.3.2.1. Optional/Variable-Length Parameters in INIT

The following parameters follow the Type-Length-Value format as defined in Section 3.2.1. Any Type-Length-Value fields MUST come after the fixed-length fields defined in the previous section.

IPv4 Address Parameter (5)

<table>
<thead>
<tr>
<th>0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>++++++01234567890123456789012345678901</td>
</tr>
<tr>
<td>++++++01234567890123456789012345678901</td>
</tr>
<tr>
<td>++++++01234567890123456789012345678901</td>
</tr>
<tr>
<td>++++++01234567890123456789012345678901</td>
</tr>
</tbody>
</table>

Contains an IPv4 address of the sending endpoint. It is binary encoded.

IPv6 Address Parameter (6)
IPv6 Address: 128 bits (unsigned integer)

Contains an IPv6 [RFC8200] address of the sending endpoint. It is binary encoded.

Note: A sender MUST NOT use an IPv4-mapped IPv6 address [RFC4291], but SHOULD instead use an IPv4 Address parameter for an IPv4 address.

Combined with the Source Port Number in the SCTP common header, the value passed in an IPv4 or IPv6 Address parameter indicates a transport address the sender of the INIT will support for the association being initiated. That is, during the life time of this association, this IP address can appear in the source address field of an IP datagram sent from the sender of the INIT, and can be used as a destination address of an IP datagram sent from the receiver of the INIT.

More than one IP Address parameter can be included in an INIT chunk when the INIT sender is multi-homed. Moreover, a multi-homed endpoint might have access to different types of network; thus, more than one address type can be present in one INIT chunk, i.e., IPv4 and IPv6 addresses are allowed in the same INIT chunk.

If the INIT contains at least one IP Address parameter, then the source address of the IP datagram containing the INIT chunk and any additional address(es) provided within the INIT can be used as destinations by the endpoint receiving the INIT. If the INIT does not contain any IP Address parameters, the endpoint receiving the INIT MUST use the source address associated with the received IP datagram as its sole destination address for the association.

Note that not using any IP Address parameters in the INIT and INIT ACK is an alternative to make an association more likely to work across a NAT box.
The sender of the INIT SHOULD use this parameter to suggest to the receiver of the INIT for a longer life-span of the State Cookie.

Suggested Cookie Life-Span Increment: 32 bits (unsigned integer)

This parameter indicates to the receiver how much increment in milliseconds the sender wishes the receiver to add to its default cookie life-span.

This optional parameter MAY be added to the INIT chunk by the sender when it reattempts establishing an association with a peer to which its previous attempt of establishing the association failed due to a stale cookie operation error. The receiver MAY choose to ignore the suggested cookie life-span increase for its own security reasons.

Host Name Address (11)

The sender of an INIT chunk MUST NOT include this parameter. The usage of the Host Name Address parameter is deprecated.

Host Name: variable length

This field contains a host name in "host name syntax" per RFC 1123 Section 2.1 [RFC1123]. The method for resolving the host name is out of scope of SCTP.

Note: At least one null terminator is included in the Host Name string and MUST be included in the length.
The sender of INIT uses this parameter to list all the address types it can support.

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Type = 12            |          Length               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Address Type #1        |        Address Type #2        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            ......                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-++-+-+-+-+-+-+-+-+-+-+-+-+-+-++-+-+-+
```

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.

3.3.3. Initiation Acknowledgement (INIT ACK) (2)

The INIT ACK chunk is used to acknowledge the initiation of an SCTP association.

The parameter part of INIT ACK is formatted similarly to the INIT chunk. It uses two extra variable parameters: The State Cookie and the Unrecognized Parameter:

The format of the INIT ACK chunk is shown below:

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 2    |  Chunk Flags  |      Chunk Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Initiate Tag                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Advertised Receiver Window Credit                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Number of Outbound Streams   |  Number of Inbound Streams    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Initial TSN                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \   
/              Optional/Variable-Length Parameters              
\                                                               
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
Initiate Tag: 32 bits (unsigned integer)

The receiver of the INIT ACK records the value of the Initiate Tag parameter. This value MUST be placed into the Verification Tag field of every SCTP packet that the INIT ACK receiver transmits within this association.

The Initiate Tag MUST NOT take the value 0. See Section 5.3.1 for more on the selection of the Initiate Tag value.

If the value of the Initiate Tag in a received INIT ACK chunk is found to be 0, the receiver MUST destroy the association discarding its TCB. The receiver MAY send an ABORT for debugging purpose.

Advertised Receiver Window Credit (a_rwnd): 32 bits (unsigned integer)

This value represents the dedicated buffer space, in number of bytes, the sender of the INIT ACK has reserved in association with this window. During the life of the association, this buffer space SHOULD NOT be lessened (i.e., dedicated buffers taken away from this association).

Number of Outbound Streams (OS): 16 bits (unsigned integer)

Defines the number of outbound streams the sender of this INIT ACK chunk wishes to create in this association. The value of 0 MUST NOT be used, and the value MUST NOT be greater than the MIS value sent in the INIT chunk.

Note: A receiver of an INIT ACK with the OS value set to 0 SHOULD destroy the association discarding its TCB.

Number of Inbound Streams (MIS): 16 bits (unsigned integer)

Defines the maximum number of streams the sender of this INIT ACK chunk allows the peer end to create in this association. The value 0 MUST NOT be used.

Note: There is no negotiation of the actual number of streams but instead the two endpoints will use the min(requested, offered). See Section 5.1.1 for details.

Note: A receiver of an INIT ACK with the MIS value set to 0 SHOULD destroy the association discarding its TCB.

Initial TSN (I-TSN): 32 bits (unsigned integer)
Defines the initial TSN that the INIT ACK sender will use. The valid range is from 0 to 4294967295. This field MAY be set to the value of the Initiate Tag field.

### Fixed Parameters

<table>
<thead>
<tr>
<th>Fixed Parameters</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initiate Tag</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Advertised Receiver Window Credit</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Number of Outbound Streams</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Number of Inbound Streams</td>
<td>Mandatory</td>
</tr>
<tr>
<td>Initial TSN</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

### Variable Parameters

<table>
<thead>
<tr>
<th>Variable Parameters</th>
<th>Status</th>
<th>Type Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Cookie</td>
<td>Mandatory</td>
<td>7</td>
</tr>
<tr>
<td>IPv4 Address (Note 1)</td>
<td>Optional</td>
<td>5</td>
</tr>
<tr>
<td>IPv6 Address (Note 1)</td>
<td>Optional</td>
<td>6</td>
</tr>
<tr>
<td>Unrecognized Parameter</td>
<td>Optional</td>
<td>8</td>
</tr>
<tr>
<td>Reserved for ECN Capable (Note 2)</td>
<td>Optional</td>
<td>32768 (0x8000)</td>
</tr>
<tr>
<td>Host Name Address (Note 3)</td>
<td>Optional</td>
<td>11</td>
</tr>
</tbody>
</table>

Note 1: The INIT ACK chunks can contain any number of IP address parameters that can be IPv4 and/or IPv6 in any combination.

Note 2: The ECN Capable field is reserved for future use of Explicit Congestion Notification.

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 "Unresolvable Address" error cause.

IMPLEMENTATION NOTE: An implementation MUST be prepared to receive an INIT ACK that is quite large (more than 1500 bytes) due to the variable size of the State Cookie AND the variable address list. For example if a responder to the INIT has 1000 IPv4 addresses it wishes to send, it would need at least 8,000 bytes to encode this in the INIT ACK.

IMPLEMENTATION NOTE: If an INIT ACK chunk is received with known parameters that are not optional parameters of the INIT ACK chunk, then the receiver SHOULD process the INIT ACK chunk and send back a COOKIE ECHO. The receiver of the INIT ACK chunk MAY bundle an ERROR chunk with the COOKIE ECHO chunk. However, restrictive implementations MAY send back an ABORT chunk in response to the INIT ACK chunk.
In combination with the Source Port carried in the SCTP common header, each IP Address parameter in the INIT ACK indicates to the receiver of the INIT ACK a valid transport address supported by the sender of the INIT ACK for the lifetime of the association being initiated.

If the INIT ACK contains at least one IP Address parameter, then the source address of the IP datagram containing the INIT ACK and any additional address(es) provided within the INIT ACK may be used as destinations by the receiver of the INIT ACK. If the INIT ACK does not contain any IP Address parameters, the receiver of the INIT ACK MUST use the source address associated with the received IP datagram as its sole destination address for the association.

The State Cookie and Unrecognized Parameters use the Type-Length-Value format as defined in Section 3.2.1 and are described below. The other fields are defined the same as their counterparts in the INIT chunk.

3.3.3.1. Optional or Variable-Length Parameters

State Cookie

Parameter Type Value: 7

Parameter Length: Variable size, depending on size of Cookie.

Parameter Value:

This parameter value MUST contain all the necessary state and parameter information required for the sender of this INIT ACK to create the association, along with a Message Authentication Code (MAC). See Section 5.1.3 for details on State Cookie definition.

Unrecognized Parameter:

Parameter Type Value: 8

Parameter Length: Variable size.

Parameter Value:

This parameter is returned to the originator of the INIT chunk when the INIT contains an unrecognized parameter that has a value that indicates it SHOULD be reported to the sender. This parameter value field will contain unrecognized parameters copied from the INIT chunk complete with Parameter Type, Length, and Value fields.
3.3.4. Selective Acknowledgement (SACK) (3)

This chunk is sent to the peer endpoint to acknowledge received DATA chunks and to inform the peer endpoint of gaps in the received subsequences of DATA chunks as represented by their TSNs.

The SACK MUST contain the Cumulative TSN Ack, Advertised Receiver Window Credit (a_rwnd), Number of Gap Ack Blocks, and Number of Duplicate TSNs fields.

By definition, the value of the Cumulative TSN Ack parameter is the last TSN received before a break in the sequence of received TSNs occurs; the next TSN value following this one has not yet been received at the endpoint sending the SACK. This parameter therefore acknowledges receipt of all TSNs less than or equal to its value.

The handling of a_rwnd by the receiver of the SACK is discussed in detail in Section 6.2.1.

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 have not been received. By definition, all TSNs acknowledged by Gap Ack Blocks are greater than the value of the Cumulative TSN Ack.
| 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 |
|+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+|
|  Type = 3 | Chunk Flags | Chunk Length |
|+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+|
|                       Cumulative TSN Ack                      |
|+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+|
|                Advertised Receiver Window Credit (a_rwnd)     |
|+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+|
| Number of Gap Ack Blocks = N | Number of Duplicate TSNs = X |
|+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+|
|       Gap Ack Block #1 Start | Gap Ack Block #1 End         |
|+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+|
|                        Duplicate TSN 1                      |
|+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+|
|                        ...                              |
|+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+|
|                        Duplicate TSN X                      |
|+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+|

Chunk Flags: 8 bits

Set to all '0's on transmit and ignored on receipt.

Cumulative TSN Ack: 32 bits (unsigned integer)

This parameter contains the TSN of the last DATA chunk received in sequence before a gap. In the case where no DATA chunk has been received, this value is set to the peer’s Initial TSN minus one.

Advertised Receiver Window Credit (a_rwnd): 32 bits (unsigned integer)

This field indicates the updated receive buffer space in bytes of the sender of this SACK; see Section 6.2.1 for details.

Number of Gap Ack Blocks: 16 bits (unsigned integer)

Indicates the number of Gap Ack Blocks included in this SACK.
Number of Duplicate TSNs: 16 bit

This field contains the number of duplicate TSNs the endpoint has received. Each duplicate TSN is listed following the Gap Ack Block list.

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. This means that the DATA chunks with TSNs equal to (Cumulative TSN Ack + Gap Ack Block Start - 1) and (Cumulative TSN Ack + Gap Ack Block End + 1) have not been received.

Gap Ack Block Start: 16 bits (unsigned integer)

Indicates the Start offset TSN for this Gap Ack Block. To calculate the actual TSN number the Cumulative TSN Ack is added to this offset number. This calculated TSN identifies the first TSN in this Gap Ack Block that has been received.

Gap Ack Block End: 16 bits (unsigned integer)

Indicates the End offset TSN for this Gap Ack Block. To calculate the actual TSN number, the Cumulative TSN Ack is added to this offset number. This calculated TSN identifies the TSN of the last DATA chunk received in this Gap Ack Block.

For example, assume that the receiver has the following DATA chunks newly arrived at the time when it decides to send a Selective ACK,
then the parameter part of the SACK MUST be constructed as follows
(assuming the new a_rwnd is set to 4660 by the sender):

```
| Cumulative TSN Ack = 12 |
| a_rwnd = 4660 |
| num of block=2 | num of dup=0 |
| block #1 strt=2 | block #1 end=3 |
| block #2 strt=5 | block #2 end=5 |
```

Duplicate TSN: 32 bits (unsigned integer)

Indicates the number of times a TSN was received in duplicate
since the last SACK was sent. Every time a receiver gets a
duplicate TSN (before sending the SACK), it adds it to the list of
duplicates. The duplicate count is reinitialized to zero after
sending each SACK.

For example, if a receiver were to get the TSN 19 three times it
would list 19 twice in the outbound SACK. After sending the SACK, if
it received yet one more TSN 19 it would list 19 as a duplicate once
in the next outgoing SACK.
3.3.5. Heartbeat Request (HEARTBEAT) (4)

An endpoint SHOULD send this chunk to its peer endpoint to probe the
reachability of a particular destination transport address defined in
the present association.

The parameter field contains the Heartbeat Information, which is a
variable-length opaque data structure understood only by the sender.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 4    | Chunk Flags  |      Heartbeat Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                  \                           
                              Heartbeat Information TLV (Variable-Length)
                                  \                           
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Flags: 8 bits
Set to 0 on transmit and ignored on receipt.

Heartbeat Length: 16 bits (unsigned integer)
Set to the size of the chunk in bytes, including the chunk header
and the Heartbeat Information field.

Heartbeat Information: variable length
Defined as a variable-length parameter using the format described
in Section 3.2.1, i.e.:

```
|    Heartbeat Info Type=1      |         HB Info Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                  \                           
                              Sender-Specific Heartbeat Info
                                  \                           
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Sender-Specific Heartbeat Info field SHOULD include
information about the sender’s current time when this HEARTBEAT
chunk is sent and the destination transport address to which this HEARTBEAT is sent (see Section 8.3). This information is simply reflected back by the receiver in the HEARTBEAT ACK message (see Section 3.3.6). Note also that the HEARTBEAT message is both for reachability checking and for path verification (see Section 5.4). When a HEARTBEAT chunk is being used for path verification purposes, it MUST hold a 64-bit random nonce.

3.3.6. Heartbeat Acknowledgement (HEARTBEAT ACK) (5)

An endpoint MUST send this chunk to its peer endpoint as a response to a HEARTBEAT chunk (see Section 8.3). A HEARTBEAT ACK is always sent to the source IP address of the IP datagram containing the HEARTBEAT chunk to which this ack is responding.

The parameter field contains a variable-length opaque data structure.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 5    | Chunk Flags |    Heartbeat Ack Length       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                       
                                         Heartbeat Information TLV (Variable-Length) |
                                         
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Flags: 8 bits

Set to 0 on transmit and ignored on receipt.

Heartbeat Ack Length: 16 bits (unsigned integer)

Set to the size of the chunk in bytes, including the chunk header and the Heartbeat Information field.

Heartbeat Information: variable length

This field MUST contain the Heartbeat Information parameter of the Heartbeat Request to which this Heartbeat Acknowledgement is responding.

<table>
<thead>
<tr>
<th>Variable Parameters</th>
<th>Status</th>
<th>Type Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heartbeat Info</td>
<td>Mandatory</td>
<td>1</td>
</tr>
</tbody>
</table>
3.3.7. Abort Association (ABORT) (6)

The ABORT chunk is sent to the peer of an association to close the association. The ABORT chunk MAY contain Cause Parameters to inform the receiver about the reason of the abort. DATA chunks MUST NOT be bundled with ABORT. Control chunks (except for INIT, INIT ACK, and SHUTDOWN COMPLETE) MAY be bundled with an ABORT, but they MUST be placed before the ABORT in the SCTP packet or they will be ignored by the receiver.

If an endpoint receives an ABORT with a format error or no TCB is found, it MUST silently discard it. Moreover, under any circumstances, an endpoint that receives an ABORT MUST NOT respond to that ABORT by sending an ABORT of its own.

```
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 = 6    |Reserved     |T|           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \n/                   zero or more Error Causes                   /\n+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Chunk Flags: 8 bits

Reserved: 7 bits

Set to 0 on transmit and ignored on receipt.

T bit: 1 bit

The T bit is set to 0 if the sender filled in the Verification Tag expected by the peer. If the Verification Tag is reflected, the T bit MUST be set to 1. Reflecting means that the sent Verification Tag is the same as the received one.

Length: 16 bits (unsigned integer)

Set to the size of the chunk in bytes, including the chunk header and all the Error Cause fields present.

See Section 3.3.10 for Error Cause definitions.

Note: Special rules apply to this chunk for verification; please see Section 8.5.1 for details.
3.3.8. Shutdown Association (SHUTDOWN) (7)

An endpoint in an association MUST use this chunk to initiate a graceful close of the association with its peer. This chunk has the following format.

```
+---------------------------------------------+
|   Type = 7    | Chunk Flags |      Length = 8               |
+---------------------------------------------+
|                      Cumulative TSN Ack       |
+---------------------------------------------+
```

Chunk Flags: 8 bits

Set to 0 on transmit and ignored on receipt.

Length: 16 bits (unsigned integer)

Indicates the length of the parameter. Set to 8.

Cumulative TSN Ack: 32 bits (unsigned integer)

This parameter contains the TSN of the last chunk received in sequence before any gaps.

Note: Since the SHUTDOWN message does not contain Gap Ack Blocks, it cannot be used to acknowledge TSNs received out of order. In a SACK, lack of Gap Ack Blocks that were previously included indicates that the data receiver reneged on the associated DATA chunks. Since SHUTDOWN does not contain Gap Ack Blocks, the receiver of the SHUTDOWN MUST NOT interpret the lack of a Gap Ack Block as a renego. (See Section 6.2 for information on reneging.)

3.3.9. Shutdown Acknowledgement (SHUTDOWN ACK) (8)

This chunk MUST be used to acknowledge the receipt of the SHUTDOWN chunk at the completion of the shutdown process; see Section 9.2 for details.

The SHUTDOWN ACK chunk has no parameters.
Chunk Flags: 8 bits

Set to 0 on transmit and ignored on receipt.

3.3.10. Operation Error (ERROR) (9)

An endpoint sends this chunk to its peer endpoint to notify it of certain error conditions. It contains one or more error causes. An Operation Error is not considered fatal in and of itself, but MAY be used with an ABORT chunk to report a fatal condition. It has the following parameters:

```
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5  6  7  8  9  0  1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
|   Type = 9    | Chunk  Flags  |           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
\                    one or more Error Causes                   /
\                                                               \\  
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

Chunk Flags: 8 bits

Set to 0 on transmit and ignored on receipt.

Length: 16 bits (unsigned integer)

Set to the size of the chunk in bytes, including the chunk header and all the Error Cause fields present.

Error causes are defined as variable-length parameters using the format described in Section 3.2.1, that is:

```
  0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5  6  7  8  9  0  1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
|           Cause Code          |       Cause Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
\                    Cause-Specific Information                 /
\                                                               \\  
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-

Cause Code: 16 bits (unsigned integer)

Defines the type of error conditions being reported.

Cause Code
<table>
<thead>
<tr>
<th>Value</th>
<th>Cause Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Invalid Stream Identifier</td>
</tr>
<tr>
<td>2</td>
<td>Missing Mandatory Parameter</td>
</tr>
<tr>
<td>3</td>
<td>Stale Cookie Error</td>
</tr>
<tr>
<td>4</td>
<td>Out of Resource</td>
</tr>
<tr>
<td>5</td>
<td>Unresolvable Address</td>
</tr>
<tr>
<td>6</td>
<td>Unrecognized Chunk Type</td>
</tr>
<tr>
<td>7</td>
<td>Invalid Mandatory Parameter</td>
</tr>
<tr>
<td>8</td>
<td>Unrecognized Parameters</td>
</tr>
<tr>
<td>9</td>
<td>No User Data</td>
</tr>
<tr>
<td>10</td>
<td>Cookie Received While Shutting Down</td>
</tr>
<tr>
<td>11</td>
<td>Restart of an Association with New Addresses</td>
</tr>
<tr>
<td>12</td>
<td>User Initiated Abort</td>
</tr>
<tr>
<td>13</td>
<td>Protocol Violation</td>
</tr>
</tbody>
</table>

Cause Length: 16 bits (unsigned integer)

Set to the size of the parameter in bytes, including the Cause Code, Cause Length, and Cause-Specific Information fields.

Cause-Specific Information: variable length

This field carries the details of the error condition.

Section 3.3.10.1 - Section 3.3.10.13 define error causes for SCTP. Guidelines for the IETF to define new error cause values are discussed in Section 14.4.

3.3.10.1. Invalid Stream Identifier (1)

Cause of error

Invalid Stream Identifier: Indicates endpoint received a DATA chunk sent to a nonexistent stream.

```
  +++++++++++++++++++++++++++++++
  |     Cause Code=1              |
  +++++++++++++++++++++++++++++++
  |     Cause Length=8           |
  +++++++++++++++++++++++++++++++
  |        Stream Identifier      |
  |   (Reserved)                 |
  +++++++++++++++++++++++++++++++
```

Stream Identifier: 16 bits (unsigned integer)

Contains the Stream Identifier of the DATA chunk received in error.

Reserved: 16 bits
This field is reserved. It is set to all 0’s on transmit and ignored on receipt.

3.3.10.2. Missing Mandatory Parameter (2)

Cause of error

---

Missing Mandatory Parameter: Indicates that one or more mandatory TLV parameters are missing in a received INIT or INIT ACK.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
| Cause Code=2 | Cause Length=8+N*2 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
| Number of missing params=N |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
| Missing Param Type #1 | Missing Param Type #2 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
| Missing Param Type #N-1 | Missing Param Type #N |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

Number of Missing params: 32 bits (unsigned integer)

This field contains the number of parameters contained in the Cause-Specific Information field.

Missing Param Type: 16 bits (unsigned integer)

Each field will contain the missing mandatory parameter number.

3.3.10.3. Stale Cookie Error (3)

Cause of error

---

Stale Cookie Error: Indicates the receipt of a valid State Cookie that has expired.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
| Cause Code=3 | Cause Length=8 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
| Measure of Staleness (usec.) |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

Measure of Staleness: 32 bits (unsigned integer)

This field contains the difference, in microseconds, between the current time and the time the State Cookie expired.
The sender of this error cause MAY choose to report how long past expiration the State Cookie is by including a non-zero value in the Measure of Staleness field. If the sender does not wish to provide this information, it SHOULD set the Measure of Staleness field to the value of zero.

3.3.10.4. Out of Resource (4)

Cause of error
------------
Out of Resource: Indicates that the sender is out of resource. This is usually sent in combination with or within an ABORT.

<table>
<thead>
<tr>
<th>Cause Code=4</th>
<th>Cause Length=4</th>
</tr>
</thead>
</table>

3.3.10.5. Unresolvable Address (5)

Cause of error
------------
Unresolvable Address: Indicates that the sender is not able to resolve the specified address parameter (e.g., type of address is not supported by the sender). This is usually sent in combination with or within an ABORT.

<table>
<thead>
<tr>
<th>Cause Code=5</th>
<th>Cause Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unresolvable Address</td>
<td></td>
</tr>
</tbody>
</table>

Unresolvable Address: variable length

The Unresolvable Address field contains the complete Type, Length, and Value of the address parameter (or Host Name parameter) that contains the unresolvable address or host name.

3.3.10.6. Unrecognized Chunk Type (6)

Cause of error
------------
Unrecognized Chunk Type: This error cause is returned to the originator of the chunk if the receiver does not understand the chunk and the upper bits of the ‘Chunk Type’ are set to 01 or 11.

```
<table>
<thead>
<tr>
<th>Cause Code=6</th>
<th>Cause Length</th>
</tr>
</thead>
</table>
\Unrecognized Chunk\ |
```

Unrecognized Chunk: variable length

The Unrecognized Chunk field contains the unrecognized chunk from the SCTP packet complete with Chunk Type, Chunk Flags, and Chunk Length.

3.3.10.7. Invalid Mandatory Parameter (7)

Cause of error
--------------

Invalid Mandatory Parameter: This error cause is returned to the originator of an INIT or INIT ACK chunk when one of the mandatory parameters is set to an invalid value.

```
| Cause Code=7 | Cause Length=4 |
```

3.3.10.8. Unrecognized Parameters (8)

Cause of error
--------------

Unrecognized Parameters: This error cause is returned to the originator of the INIT ACK chunk if the receiver does not recognize one or more Optional TLV parameters in the INIT ACK chunk.

```
<table>
<thead>
<tr>
<th>Cause Code=8</th>
<th>Cause Length</th>
</tr>
</thead>
</table>
\Unrecognized Parameters\ |
```

Unrecognized Parameters: variable length
The Unrecognized Parameters field contains the unrecognized parameters copied from the INIT ACK chunk complete with TLV. This error cause is normally contained in an ERROR chunk bundled with the COOKIE ECHO chunk when responding to the INIT ACK, when the sender of the COOKIE ECHO chunk wishes to report unrecognized parameters.

3.3.10.9. No User Data (9)

Cause of error
---------------

No User Data: This error cause is returned to the originator of a DATA chunk if a received DATA chunk has no user data.

```
+----------------------------------+
| Cause Code=9                     |
| Cause Length=8                   |
+----------------------------------+
+----------------------------------+
| TSN value                        |
+----------------------------------+
```

TSN value: 32 bits (unsigned integer)

The TSN value field contains the TSN of the DATA chunk received with no user data field.

This cause code is normally returned in an ABORT chunk (see Section 6.2).

3.3.10.10. Cookie Received While Shutting Down (10)

Cause of error
---------------

Cookie Received While Shutting Down: A COOKIE ECHO was received while the endpoint was in the SHUTDOWN-ACK-SENT state. This error is usually returned in an ERROR chunk bundled with the retransmitted SHUTDOWN ACK.

```
+----------------------------------+
| Cause Code=10                    |
| Cause Length=4                   |
+----------------------------------+
```
3.3.10.11. Restart of an Association with New Addresses (11)

Cause of error
--------------

Restart of an association with new addresses: An INIT was received on an existing association. But the INIT added addresses to the association that were previously NOT part of the association. The new addresses are listed in the error code. This ERROR is normally sent as part of an ABORT refusing the INIT (see Section 5.2).

```
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=11         |      Cause Length=Variable    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                       New Address TLVs                        /
\                                                  /\                 \\
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Note: Each New Address TLV is an exact copy of the TLV that was found in the INIT chunk that was new, including the Parameter Type and the Parameter Length.

3.3.10.12. User-Initiated Abort (12)

Cause of error
--------------

This error cause MAY be included in ABORT chunks that are sent because of an upper-layer request. The upper layer can specify an Upper Layer Abort Reason that is transported by SCTP transparently and MAY be delivered to the upper-layer protocol at the peer.

```
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=12         |      Cause Length=Variable    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                    Upper Layer Abort Reason                   /
\                                                  /\                 \\
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

3.3.10.13. Protocol Violation (13)

Cause of error
--------------
This error cause MAY be included in ABORT chunks that are sent because an SCTP endpoint detects a protocol violation of the peer that is not covered by the error causes described in Section 3.3.10.1 to Section 3.3.10.12. An implementation MAY provide additional information specifying what kind of protocol violation has been detected.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Cause Code=13         |      Cause Length=Variable    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                    Additional Information                     /
\                                                               
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

3.3.11. Cookie Echo (COOKIE李CHO) (10)

This chunk is used only during the initialization of an association. It is sent by the initiator of an association to its peer to complete the initialization process. This chunk MUST precede any DATA chunk sent within the association, but MAY be bundled with one or more DATA chunks in the same packet.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 10   |Chunk  Flags   |         Length                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                     Cookie                                    /
\                                                               
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Flags: 8 bits

Set to 0 on transmit and ignored on receipt.

Length: 16 bits (unsigned integer)

Set to the size of the chunk in bytes, including the 4 bytes of the chunk header and the size of the cookie.

Cookie: variable size

This field MUST contain the exact cookie received in the State Cookie parameter from the previous INIT ACK.
An implementation SHOULD make the cookie as small as possible to ensure interoperability.

Note: A Cookie Echo does NOT contain a State Cookie parameter; instead, the data within the State Cookie’s Parameter Value becomes the data within the Cookie Echo’s Chunk Value. This allows an implementation to change only the first 2 bytes of the State Cookie parameter to become a COOKIE ECHO chunk.

3.3.12. Cookie Acknowledgement (COOKIE ACK) (11)

This chunk is used only during the initialization of an association. It is used to acknowledge the receipt of a COOKIE ECHO chunk. This chunk MUST precede any DATA or SACK chunk sent within the association, but MAY be bundled with one or more DATA chunks or SACK chunk’s in the same SCTP packet.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Type = 11           |Chunk Flags   |     Length = 4                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Flags: 8 bits

Set to 0 on transmit and ignored on receipt.

3.3.13. Shutdown Complete (SHUTDOWN COMPLETE) (14)

This chunk MUST be used to acknowledge the receipt of the SHUTDOWN ACK chunk at the completion of the shutdown process; see Section 9.2 for details.

The SHUTDOWN COMPLETE chunk has no parameters.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Type = 14           |Reserved     |T|      Length = 4               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Chunk Flags: 8 bits

Reserved: 7 bits

Set to 0 on transmit and ignored on receipt.

T bit: 1 bit
The T bit is set to 0 if the sender filled in the Verification Tag expected by the peer. If the Verification Tag is reflected, the T bit MUST be set to 1. Reflecting means that the sent Verification Tag is the same as the received one.

Note: Special rules apply to this chunk for verification, please see Section 8.5.1 for details.

4. SCTP Association State Diagram

During the life time of an SCTP association, the SCTP endpoint’s association progresses from one state to another in response to various events. The events that might potentially advance an association’s state include:

- SCTP user primitive calls, e.g., [ASSOCIATE], [SHUTDOWN], [ABORT],
- Reception of INIT, COOKIE ECHO, ABORT, SHUTDOWN, etc., control chunks, or
- Some timeout events.

The state diagram in the figures below illustrates state changes, together with the causing events and resulting actions. Note that some of the error conditions are not shown in the state diagram. Full descriptions of all special cases are found in the text.

Note: Chunk names are given in all capital letters, while parameter names have the first letter capitalized, e.g., COOKIE ECHO chunk type vs. State Cookie parameter. If more than one event/message can occur that causes a state transition, it is labeled (A), (B), etc.

```
-----          -------- (from any state)
/       \
rcv INIT |            | rcv ABORT [ABORT]
--------+            +----------------- or -----------------
v v delete TCB snd ABORT
generate Cookie 
snd INIT ACK ---| CLOSED ---
+-------------+
/ \ [ASSOCIATE]
/ \ create TCB
|   snd INIT
|   strt init timer

rcv valid

COOKİE ECHO

(1) create TCB
| COOKIE-WAIT (2)
|-------------------
snd COOKIE ACK

----- (from any state)
```
Notes:

1) If the State Cookie in the received COOKIE ECHO is invalid (i.e., failed to pass the integrity check), the receiver MUST silently discard the packet. Or, if the received State Cookie is expired (see Section 5.1.5), the receiver MUST send back an ERROR chunk. In either case, the receiver stays in the CLOSED state.

2) If the T1-init timer expires, the endpoint MUST retransmit INIT and restart the T1-init timer without changing state. This MUST be repeated up to 'Max.Init.Retransmits' times. After that, the endpoint MUST abort the initialization process and report the error to the SCTP user.

3) If the T1-cookie timer expires, the endpoint MUST retransmit COOKIE ECHO and restart the T1-cookie timer without changing state. This MUST be repeated up to 'Max.Init.Retransmits' times. After that, the endpoint MUST abort the initialization process and report the error to the SCTP user.
4) In the SHUTDOWN-SENT state, the endpoint MUST acknowledge any received DATA chunks without delay.

5) In the SHUTDOWN-RECEIVED state, the endpoint MUST NOT accept any new send requests from its SCTP user.

6) In the SHUTDOWN-RECEIVED state, the endpoint MUST transmit or retransmit data and leave this state when all data in queue is transmitted.

7) In the SHUTDOWN-ACK-SENT state, the endpoint MUST NOT accept any new send requests from its SCTP user.

The CLOSED state is used to indicate that an association is not created (i.e., doesn’t exist).

5. Association Initialization

Before the first data transmission can take place from one SCTP endpoint ("A") to another SCTP endpoint ("Z"), the two endpoints MUST complete an initialization process in order to set up an SCTP association between them.

The SCTP user at an endpoint can use the ASSOCIATE primitive to initialize an SCTP association to another SCTP endpoint.

IMPLEMENTATION NOTE: From an SCTP user’s point of view, an association might be implicitly opened, without an ASSOCIATE primitive (see Section 10.1 B) being invoked, by the initiating endpoint’s sending of the first user data to the destination endpoint. The initiating SCTP will assume default values for all mandatory and optional parameters for the INIT/INIT ACK.

Once the association is established, unidirectional streams are open for data transfer on both ends (see Section 5.1.1).

5.1. Normal Establishment of an Association

The initialization process consists of the following steps (assuming that SCTP endpoint "A" tries to set up an association with SCTP endpoint "Z" and "Z" accepts the new association):

A) "A" first sends an INIT chunk to "Z". In the INIT, "A" MUST provide its Verification Tag (Tag_A) in the Initiate Tag field. Tag_A SHOULD be a random number in the range of 1 to 4294967295 (see Section 5.3.1 for Tag value selection). After sending the INIT, "A" starts the T1-init timer and enters the COOKIE-WAIT state.

B) "Z" responds immediately with an INIT ACK chunk. The destination IP address of the INIT ACK MUST be set to the source IP address
of the INIT to which this INIT ACK is responding. In the response, besides filling in other parameters, "Z" MUST set the Verification Tag field to Tag_A, and also provide its own Verification Tag (Tag_Z) in the Initiate Tag field.

Moreover, "Z" MUST generate and send along with the INIT ACK a State Cookie. See Section 5.1.3 for State Cookie generation.

Note: After sending out INIT ACK with the State Cookie parameter, "Z" MUST NOT allocate any resources or keep any states for the new association. Otherwise, "Z" will be vulnerable to resource attacks.

C) Upon reception of the INIT ACK from "Z", "A" stops the T1-init timer and leave the COOKIE-WAIT state. "A" then sends the State Cookie received in the INIT ACK chunk in a COOKIE ECHO chunk, starts the T1-cookie timer, and enters the COOKIE-ECHOED state.

Note: The COOKIE ECHO chunk MAY be bundled with any pending outbound DATA chunks, but it MUST be the first chunk in the packet and until the COOKIE ACK is returned the sender MUST NOT send any other packets to the peer.

D) Upon reception of the COOKIE ECHO chunk, endpoint "Z" will reply with a COOKIE ACK chunk after building a TCB and moving to the ESTABLISHED state. A COOKIE ACK chunk MAY be bundled with any pending DATA chunks (and/or SACK chunks), but the COOKIE ACK chunk MUST be the first chunk in the packet.

IMPLEMENTATION NOTE: An implementation can choose to send the Communication Up notification to the SCTP user upon reception of a valid COOKIE ECHO chunk.

E) Upon reception of the COOKIE ACK, endpoint "A" will move from the COOKIE-ECHOED state to the ESTABLISHED state, stopping the T1-cookie timer. It can also notify its ULP about the successful establishment of the association with a Communication Up notification (see Section 10).

An INIT or INIT ACK chunk MUST NOT be bundled with any other chunk. They MUST be the only chunks present in the SCTP packets that carry them.

An endpoint MUST send the INIT ACK to the IP address from which it received the INIT.

Note: T1-init timer and T1-cookie timer SHOULD follow the same rules given in Section 6.3.
If an endpoint receives an INIT, INIT ACK, or COOKIE ECHO chunk but
decides not to establish the new association due to missing mandatory
parameters in the received INIT or INIT ACK, invalid parameter
values, or lack of local resources, it SHOULD respond with an ABORT
chunk. It SHOULD also specify the cause of abort, such as the type
of the missing mandatory parameters, etc., by including the error
cause parameters with the ABORT chunk. The Verification Tag field in
the common header of the outbound SCTP packet containing the ABORT
chunk MUST be set to the Initiate Tag value of the peer.

Note that a COOKIE ECHO chunk that does NOT pass the integrity check
is NOT considered an ‘invalid parameter’ and requires special
handling; see Section 5.1.5.

After the reception of the first DATA chunk in an association the
endpoint MUST immediately respond with a SACK to acknowledge the DATA
chunk. Subsequent acknowledgements SHOULD be done as described in
Section 6.2.

When the TCB is created, each endpoint MUST set its internal
Cumulative TSN Ack Point to the value of its transmitted Initial TSN
minus one.

IMPLEMENTATION NOTE: The IP addresses and SCTP port are generally
used as the key to find the TCB within an SCTP instance.

5.1.1. Handle Stream Parameters

In the INIT and INIT ACK chunks, the sender of the chunk MUST
indicate the number of outbound streams (OSs) it wishes to have in
the association, as well as the maximum inbound streams (MISs) it
will accept from the other endpoint.

After receiving the stream configuration information from the other
side, each endpoint MUST perform the following check: If the peer’s
MIS is less than the endpoint’s OS, meaning that the peer is
incapable of supporting all the outbound streams the endpoint wants
to configure, the endpoint MUST use MIS outbound streams and MAY
report any shortage to the upper layer. The upper layer can then
choose to abort the association if the resource shortage is
unacceptable.

After the association is initialized, the valid outbound stream
identifier range for either endpoint MUST be 0 to min(local OS,
remote MIS)-1.
5.1.2. Handle Address Parameters

During the association initialization, an endpoint uses the following rules to discover and collect the destination transport address(es) of its peer.

A) If there are no address parameters present in the received INIT or INIT ACK chunk, the endpoint MUST take the source IP address from which the chunk arrives and record it, in combination with the SCTP source port number, as the only destination transport address for this peer.

B) If there is a Host Name Address parameter present in the received INIT or INIT ACK chunk, the endpoint MUST immediately send an ABORT and MAY include an "Unresolvable Address" error cause to its peer. The ABORT SHOULD be sent to the source IP address from which the last peer packet was received.

C) If there are only IPv4/IPv6 addresses present in the received INIT or INIT ACK chunk, the receiver MUST derive and record all the transport addresses from the received chunk AND the source IP address that sent the INIT or INIT ACK. The transport addresses are derived by the combination of SCTP source port (from the common header) and the IP Address parameter(s) carried in the INIT or INIT ACK chunk and the source IP address of the IP datagram. The receiver SHOULD use only these transport addresses as destination transport addresses when sending subsequent packets to its peer.

D) An INIT or INIT ACK chunk MUST be treated as belonging to an already established association (or one in the process of being established) if the use of any of the valid address parameters contained within the chunk would identify an existing TCB.

IMPLEMENTATION NOTE: In some cases (e.g., when the implementation doesn’t control the source IP address that is used for transmitting), an endpoint might need to include in its INIT or INIT ACK all possible IP addresses from which packets to the peer could be transmitted.

After all transport addresses are derived from the INIT or INIT ACK chunk using the above rules, the endpoint selects one of the transport addresses as the initial primary path.

Note: The INIT ACK MUST be sent to the source address of the INIT.
The sender of INIT chunks MAY include a ‘Supported Address Types’ parameter in the INIT to indicate what types of addresses are acceptable.

IMPLEMENTATION NOTE: In the case that the receiver of an INIT ACK fails to resolve the address parameter due to an unsupported type, it can abort the initiation process and then attempt a reinitiation by using a ‘Supported Address Types’ parameter in the new INIT to indicate what types of address it prefers.

IMPLEMENTATION NOTE: If an SCTP endpoint that only supports either IPv4 or IPv6 receives IPv4 and IPv6 addresses in an INIT or INIT ACK chunk from its peer, it MUST use all the addresses belonging to the supported address family. The other addresses MAY be ignored. The endpoint SHOULD NOT respond with any kind of error indication.

IMPLEMENTATION NOTE: If an SCTP endpoint lists in the ‘Supported Address Types’ parameter either IPv4 or IPv6, but uses the other family for sending the packet containing the INIT chunk, or if it also lists addresses of the other family in the INIT chunk, then the address family that is not listed in the ‘Supported Address Types’ parameter SHOULD also be considered as supported by the receiver of the INIT chunk. The receiver of the INIT chunk SHOULD NOT respond with any kind of error indication.

5.1.3. Generating State Cookie

When sending an INIT ACK as a response to an INIT chunk, the sender of INIT ACK creates a State Cookie and sends it in the State Cookie parameter of the INIT ACK. Inside this State Cookie, the sender can include a MAC (see [RFC2104] for an example), a timestamp on when the State Cookie is created, and the lifespan of the State Cookie, along with all the information necessary for it to establish the association.

The following steps SHOULD be taken to generate the State Cookie:

1) Create an association TCB using information from both the received INIT and the outgoing INIT ACK chunk,
2) In the TCB, set the creation time to the current time of day, and the lifespan to the protocol parameter 'Valid.Cookie.Life' (see Section 15),
3) From the TCB, identify and collect the minimal subset of information needed to re-create the TCB, and generate a MAC using this subset of information and a secret key (see [RFC2104] for an example of generating a MAC), and
4) Generate the State Cookie by combining this subset of information and the resultant MAC.
After sending the INIT ACK with the State Cookie parameter, the sender SHOULD delete the TCB and any other local resource related to the new association, so as to prevent resource attacks.

The hashing method used to generate the MAC is strictly a private matter for the receiver of the INIT chunk. The use of a MAC is mandatory to prevent denial-of-service attacks. The secret key SHOULD be random ([RFC4086] provides some information on randomness guidelines); it SHOULD be changed reasonably frequently, and the timestamp in the State Cookie MAY be used to determine which key is used to verify the MAC.

An implementation SHOULD make the cookie as small as possible to ensure interoperability.

5.1.4. State Cookie Processing

When an endpoint (in the COOKIE-WAIT state) receives an INIT ACK chunk with a State Cookie parameter, it MUST immediately send a COOKIE ECHO chunk to its peer with the received State Cookie. The sender MAY also add any pending DATA chunks to the packet after the COOKIE ECHO chunk.

The endpoint MUST also start the T1-cookie timer after sending out the COOKIE ECHO chunk. If the timer expires, the endpoint MUST retransmit the COOKIE ECHO chunk and restart the T1-cookie timer. This is repeated until either a COOKIE ACK is received or ‘Max.Init.Retransmits’ (see Section 15) is reached causing the peer endpoint to be marked unreachable (and thus the association enters the CLOSED state).

5.1.5. State Cookie Authentication

When an endpoint receives a COOKIE ECHO chunk from another endpoint with which it has no association, it takes the following actions:

1) Compute a MAC using the TCB data carried in the State Cookie and the secret key (note the timestamp in the State Cookie MAY be used to determine which secret key to use). [RFC2104] can be used as a guideline for generating the MAC,

2) Authenticate the State Cookie as one that it previously generated by comparing the computed MAC against the one carried in the State Cookie. If this comparison fails, the SCTP packet, including the COOKIE ECHO and any DATA chunks, SHOULD be silently discarded,

3) Compare the port numbers and the Verification Tag contained within the COOKIE ECHO chunk to the actual port numbers and the Verification Tag within the SCTP common header of the received
packet. If these values do not match, the packet MUST be silently discarded.

4) Compare the creation timestamp in the State Cookie to the current local time. If the elapsed time is longer than the lifespan carried in the State Cookie, then the packet, including the COOKIE ECHO and any attached DATA chunks, SHOULD be discarded, and the endpoint MUST transmit an ERROR chunk with a "Stale Cookie" error cause to the peer endpoint.

5) If the State Cookie is valid, create an association to the sender of the COOKIE ECHO chunk with the information in the TCB data carried in the COOKIE ECHO and enter the ESTABLISHED state.

6) Send a COOKIE ACK chunk to the peer acknowledging receipt of the COOKIE ECHO. The COOKIE ACK MAY be bundled with an outbound DATA chunk or SACK chunk; however, the COOKIE ACK MUST be the first chunk in the SCTP packet.

7) Immediately acknowledge any DATA chunk bundled with the COOKIE ECHO with a SACK (subsequent DATA chunk acknowledgement SHOULD follow the rules defined in Section 6.2). As mentioned in step 6, if the SACK is bundled with the COOKIE ACK, the COOKIE ACK MUST appear first in the SCTP packet.

If a COOKIE ECHO is received from an endpoint with which the receiver of the COOKIE ECHO has an existing association, the procedures in Section 5.2 SHOULD be followed.

5.1.6. An Example of Normal Association Establishment

In the following example, "A" initiates the association and then sends a user message to "Z", then "Z" sends two user messages to "A" later (assuming no bundling or fragmentation occurs):
Figure 4: INITIATION Example

If the T1-init timer expires at "A" after the INIT or COOKIE ECHO chunks are sent, the same INIT or COOKIE ECHO chunk with the same Initiate Tag (i.e., Tag_A) or State Cookie is retransmitted and the timer restarted. This is repeated Max.Init.Retransmits times before "A" considers "Z" unreachable and reports the failure to its upper layer (and thus the association enters the CLOSED state).
When retransmitting the INIT, the endpoint MUST follow the rules defined in Section 6.3 to determine the proper timer value.

5.2. Handle Duplicate or Unexpected INIT, INIT ACK, COOKIE ECHO, and COOKIE ACK

During the life time of an association (in one of the possible states), an endpoint can receive from its peer endpoint one of the setup chunks (INIT, INIT ACK, COOKIE ECHO, and COOKIE ACK). The receiver treats such a setup chunk as a duplicate and process it as described in this section.

Note: An endpoint will not receive the chunk unless the chunk was sent to an SCTP transport address and is from an SCTP transport address associated with this endpoint. Therefore, the endpoint processes such a chunk as part of its current association.

The following scenarios can cause duplicated or unexpected chunks:

A) The peer has crashed without being detected, restarted itself, and sent out a new INIT chunk trying to restore the association,

B) Both sides are trying to initialize the association at about the same time,

C) The chunk is from a stale packet that was used to establish the present association or a past association that is no longer in existence,

D) The chunk is a false packet generated by an attacker, or

E) The peer never received the COOKIE ACK and is retransmitting its COOKIE ECHO.

The rules in the following sections are applied in order to identify and correctly handle these cases.

5.2.1. INIT Received in COOKIE-WAIT or COOKIE-ECHOED State (Item B)

This usually indicates an initialization collision, i.e., each endpoint is attempting, at about the same time, to establish an association with the other endpoint.

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.

Upon receipt of an INIT in the COOKIE-ECHOED 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), provided that no NEW address has been added to the forming association. If the INIT message indicates that a new address has been added to the association, then the entire INIT MUST be discarded, and SHOULD NOT do any changes to the existing association. An ABORT SHOULD be sent in response that MAY include the error ‘Restart of an association with new addresses’. The error SHOULD list the addresses that were added to the restarting association.

When responding in either state (COOKIE-WAIT or COOKIE-ECHOED) with an INIT ACK, the original parameters are combined with those from the newly received INIT chunk. The endpoint MUST also generate a State Cookie with the INIT ACK. The endpoint uses the parameters sent in its INIT to calculate the State Cookie.

After that, the endpoint MUST NOT change its state, the T1-init timer MUST be left running, and the corresponding TCB MUST NOT be destroyed. The normal procedures for handling State Cookies when a TCB exists will resolve the duplicate INITs to a single association.

For an endpoint that is in the COOKIE-ECHOED state, it MUST populate its Tie-Tags within both the association TCB and inside the State Cookie (see Section 5.2.2 for a description of the Tie-Tags).

5.2.2. Unexpected INIT in States Other than CLOSED, COOKIE-ECHOED, COOKIE-WAIT, and SHUTDOWN-ACK-SENT

Unless otherwise stated, upon receipt of an unexpected INIT for this association, the endpoint MUST generate an INIT ACK with a State Cookie. Before responding, the endpoint MUST check to see if the unexpected INIT adds new addresses to the association. If new addresses are added to the association, the endpoint MUST respond with an ABORT, copying the ‘Initiate Tag’ of the unexpected INIT into the ‘Verification Tag’ of the outbound packet carrying the ABORT. In the ABORT response, the cause of error MAY be set to ‘restart of an association with new addresses’. The error SHOULD list the addresses that were added to the restarting association. If no new addresses
are added, when responding to the INIT in the outbound INIT ACK, the endpoint MUST copy its current Tie-Tags to a reserved place within the State Cookie and the association’s TCB. We refer to these locations inside the cookie as the Peer’s-Tie-Tag and the Local-Tie-Tag. We will refer to the copy within an association’s TCB as the Local Tag and Peer’s Tag. The outbound SCTP packet containing this INIT ACK MUST carry a Verification Tag value equal to the Initiate Tag found in the unexpected INIT. And the INIT ACK MUST contain a new Initiate Tag (randomly generated; see Section 5.3.1). Other parameters for the endpoint SHOULD be copied from the existing parameters of the association (e.g., number of outbound streams) into the INIT ACK and cookie.

After sending out the INIT ACK or ABORT, the endpoint MUST take no further actions; i.e., the existing association, including its current state, and the corresponding TCB MUST NOT be changed.

Note: Only when a TCB exists and the association is not in a COOKIE-WAIT or SHUTDOWN-ACK-SENT state are the Tie-Tags populated with a value other than 0. For a normal association INIT (i.e., the endpoint is in the CLOSED state), the Tie-Tags MUST be set to 0 (indicating that no previous TCB existed).

5.2.3. Unexpected INIT ACK

If an INIT ACK is received by an endpoint in any state other than the COOKIE-WAIT state, the endpoint SHOULD discard the INIT ACK chunk. An unexpected INIT ACK usually indicates the processing of an old or duplicated INIT chunk.

5.2.4. Handle a COOKIE ECHO when a TCB Exists

When a COOKIE ECHO chunk is received by an endpoint in any state for an existing association (i.e., not in the CLOSED state) the following rules are applied:

1) Compute a MAC as described in step 1 of Section 5.1.5,

2) Authenticate the State Cookie as described in step 2 of Section 5.1.5 (this is case C or D above).

3) Compare the timestamp in the State Cookie to the current time. If the State Cookie is older than the lifespan carried in the State Cookie and the Verification Tags contained in the State Cookie do not match the current association’s Verification Tags, the packet, including the COOKIE ECHO and any DATA chunks, SHOULD be discarded. The endpoint also MUST transmit an ERROR chunk.
with a "Stale Cookie" error cause to the peer endpoint (this is case C or D in Section 5.2).

If both Verification Tags in the State Cookie match the Verification Tags of the current association, consider the State Cookie valid (this is case E in Section 5.2) even if the lifespan is exceeded.

4) If the State Cookie proves to be valid, unpack the TCB into a temporary TCB.

5) Refer to Table 2 to determine the correct action to be taken.

<table>
<thead>
<tr>
<th>Local Tag</th>
<th>Peer’s Tag</th>
<th>Local-Tie-Tag</th>
<th>Peer’s-Tie-Tag</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>M</td>
<td>M</td>
<td>(A)</td>
</tr>
<tr>
<td>M</td>
<td>X</td>
<td>A</td>
<td>A</td>
<td>(B)</td>
</tr>
<tr>
<td>M</td>
<td>0</td>
<td>A</td>
<td>A</td>
<td>(B)</td>
</tr>
<tr>
<td>X</td>
<td>M</td>
<td>0</td>
<td>0</td>
<td>(C)</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>(D)</td>
</tr>
</tbody>
</table>

Table 2: Handling of a COOKIE ECHO when a TCB Exists

Legend:

X - Tag does not match the existing TCB.
M - Tag matches the existing TCB.
0 - No Tie-Tag in cookie (unknown).
A - All cases, i.e., M, X, or 0.

Note: For any case not shown in Table 2, the cookie SHOULD be silently discarded.

Action

A) In this case, the peer might have restarted. When the endpoint recognizes this potential ‘restart’, the existing session is treated the same as if it received an ABORT followed by a new COOKIE ECHO with the following exceptions:

* Any SCTP DATA chunks MAY be retained (this is an implementation-specific option).
A notification of RESTART SHOULD be sent to the ULP instead of a "COMMUNICATION LOST" notification.

All the congestion control parameters (e.g., cwnd, ssthresh) related to this peer MUST be reset to their initial values (see Section 6.2.1).

After this, the endpoint enters the ESTABLISHED state.

If the endpoint is in the SHUTDOWN-ACK-SENT state and recognizes that the peer has restarted (Action A), it MUST NOT set up a new association but instead resend the SHUTDOWN ACK and send an ERROR chunk with a "Cookie Received While Shutting Down" error cause to its peer.

B) In this case, both sides might be attempting to start an association at about the same time, but the peer endpoint started its INIT after responding to the local endpoint’s INIT. Thus, it might have picked a new Verification Tag, not being aware of the previous tag it had sent this endpoint. The endpoint SHOULD stay in or enter the ESTABLISHED state, but it MUST update its peer’s Verification Tag from the State Cookie, stop any init or cookie timers that might be running, and send a COOKIE ACK.

C) In this case, the local endpoint’s cookie has arrived late. Before it arrived, the local endpoint sent an INIT and received an INIT ACK and finally sent a COOKIE ECHO with the peer’s same tag but a new tag of its own. The cookie SHOULD be silently discarded. The endpoint SHOULD NOT change states and SHOULD leave any timers running.

D) When both local and remote tags match, the endpoint SHOULD enter the ESTABLISHED state, if it is in the COOKIE-ECHOED state. It SHOULD stop any cookie timer that is running and send a COOKIE ACK.

Note: The "peer’s Verification Tag" is the tag received in the Initiate Tag field of the INIT or INIT ACK chunk.

5.2.4.1. An Example of a Association Restart

In the following example, "A" initiates the association after a restart has occurred. Endpoint "Z" had no knowledge of the restart until the exchange (i.e., Heartbeats had not yet detected the failure of "A") (assuming no bundling or fragmentation occurs):
Endpoint A                                          Endpoint Z
<------------------- Association is established------------------->
Tag=Tag_A                                             Tag=Tag_Z
<<<<<<<<<--------------------------------------------------------

{A crashes and restarts}
{app sets up a association with Z}
(build TCB)

INIT [I-Tag=Tag_A',
     & other info]     --------
(Start T1-init timer)
(Enter COOKIE-WAIT state)   \---> (find an existing TCB compose temp TCB and Cookie_Z with Tie-Tags to previous association)
                                 \---> (find an existing TCB compose temp TCB and Cookie_Z with Tie-Tags to previous association)
                                 \---> (find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset association).
                                 \---> (Find existing association, Tie-Tags match old tags, Tags do not match, i.e., case X X M M above, Announce Restart to ULP and reset associ
5.2.5. Handle Duplicate COOKIE ACK

At any state other than COOKIE-ECHOED, an endpoint SHOULD silently discard a received COOKIE ACK chunk.

5.2.6. Handle Stale COOKIE Error

Receipt of an ERROR chunk with a "Stale Cookie" error cause indicates one of a number of possible events:

A) The association failed to completely setup before the State Cookie issued by the sender was processed.

B) An old State Cookie was processed after setup completed.

C) An old State Cookie is received from someone that the receiver is not interested in having an association with and the ABORT chunk was lost.

When processing an ERROR chunk with a "Stale Cookie" error cause an endpoint SHOULD first examine if an association is in the process of being set up, i.e., the association is in the COOKIE-ECHOED state. In all cases, if the association is not in the COOKIE-ECHOED state, the ERROR chunk SHOULD be silently discarded.

If the association is in the COOKIE-ECHOED state, the endpoint MAY elect one of the following three alternatives.

1) Send a new INIT chunk to the endpoint to generate a new State Cookie and reattempt the setup procedure.

2) Discard the TCB and report to the upper layer the inability to set up the association.

3) Send a new INIT chunk to the endpoint, adding a Cookie Preservative parameter requesting an extension to the life time of the State Cookie. When calculating the time extension, an implementation SHOULD use the RTT information measured based on the previous COOKIE ECHO / ERROR exchange, and SHOULD add no more than 1 second beyond the measured RTT, due to long State Cookie life times making the endpoint more subject to a replay attack.

5.3. Other Initialization Issues

5.3.1. Selection of Tag Value

Initiate Tag values SHOULD be selected from the range of 1 to 2**32 - 1. It is very important that the Initiate Tag value be randomized to help protect against "man in the middle" and "sequence number" attacks. The methods described in [RFC4086] can be used for the
Initiate Tag randomization. Careful selection of Initiate Tags is also necessary to prevent old duplicate packets from previous associations being mistakenly processed as belonging to the current association.

Moreover, the Verification Tag value used by either endpoint in a given association MUST NOT change during the life time of an association. A new Verification Tag value MUST be used each time the endpoint tears down and then reestablishes an association to the same peer.

5.4. Path Verification

During association establishment, the two peers exchange a list of addresses. In the predominant case, these lists accurately represent the addresses owned by each peer. However, it is possible that a misbehaving peer might supply addresses that it does not own. To prevent this, the following rules are applied to all addresses of the new association:

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.

2) For the receiver of the COOKIE ECHO, the only CONFIRMED address is the address to which the INIT ACK was sent.

3) All other addresses not covered by rules 1 and 2 are considered UNCONFIRMED and are subject to probing for verification.

To probe an address for verification, an endpoint will send HEARTBEATS including a 64-bit random nonce and a path indicator (to identify the address that the HEARTBEAT is sent to) within the HEARTBEAT parameter.

Upon receipt of the HEARTBEAT ACK, a verification is made that the nonce included in the HEARTBEAT parameter is the one sent to the address indicated inside the HEARTBEAT parameter. When this match occurs, the address that the original HEARTBEAT was sent to is now considered CONFIRMED and available for normal data transfer.

These probing procedures are started when an association moves to the ESTABLISHED state and are ended when all paths are confirmed.

In each RTO, a probe MAY be sent on an active UNCONFIRMED path in an attempt to move it to the CONFIRMED state. If during this probing the path becomes inactive, this rate is lowered to the normal HEARTBEAT rate. At the expiration of the RTO timer, the error
counter of any path that was probed but not CONFIRMED is incremented by one and subjected to path failure detection, as defined in Section 8.2. When probing UNCONFIRMED addresses, however, the association overall error count is NOT incremented.

The number of HEARTBEATS sent at each RTO SHOULD be limited by the HB.Max.Burst parameter. It is an implementation decision as to how to distribute HEARTBEATS to the peer’s addresses for path verification.

Whenever a path is confirmed, an indication MAY be given to the upper layer.

An endpoint MUST NOT send any chunks to an UNCONFIRMED address, with the following exceptions:

- A HEARTBEAT including a nonce MAY be sent to an UNCONFIRMED address.
- A HEARTBEAT ACK MAY be sent to an UNCONFIRMED address.
- A COOKIE ACK MAY be sent to an UNCONFIRMED address, but it MUST be bundled with a HEARTBEAT including a nonce. An implementation that does NOT support bundling MUST NOT send a COOKIE ACK to an UNCONFIRMED address.
- A COOKIE ECHO MAY be sent to an UNCONFIRMED address, but it MUST be bundled with a HEARTBEAT including a nonce, and the packet MUST NOT exceed the path MTU. If the implementation does NOT support bundling or if the bundled COOKIE ECHO plus HEARTBEAT (including nonce) would exceed the path MTU, then the implementation MUST NOT send a COOKIE ECHO to an UNCONFIRMED address.

6. User Data Transfer

Data transmission MUST only happen in the ESTABLISHED, SHUTDOWN-PENDING, and SHUTDOWN-RECEIVED states. The only exception to this is that DATA chunks are allowed to be bundled with an outbound COOKIE ECHO chunk when in the COOKIE-WAIT state.

DATA chunks MUST only be received according to the rules below in ESTABLISHED, SHUTDOWN-PENDING, and SHUTDOWN-SENT. A DATA chunk received in CLOSED is out of the blue and SHOULD be handled per Section 8.4. A DATA chunk received in any other state SHOULD be discarded.

A SACK MUST be processed in ESTABLISHED, SHUTDOWN-PENDING, and SHUTDOWN-RECEIVED. An incoming SACK MAY be processed in COOKIE-
ECHOED. A SACK in the CLOSED state is out of the blue and SHOULD be processed according to the rules in Section 8.4. A SACK chunk received in any other state SHOULD be discarded.

An SCTP receiver MUST be able to receive a minimum of 1500 bytes in one SCTP packet. This means that an SCTP endpoint MUST NOT indicate less than 1500 bytes in its initial a_rwnd sent in the INIT or INIT ACK.

For transmission efficiency, SCTP defines mechanisms for bundling of small user messages and fragmentation of large user messages. The following diagram depicts the flow of user messages through SCTP.

In this section, the term "data sender" refers to the endpoint that transmits a DATA chunk and the term "data receiver" refers to the endpoint that receives a DATA chunk. A data receiver will transmit SACK chunks.

![SCTP User Message Flow Diagram](attachment:image.png)

Figure 6: Illustration of User Data Transfer

Notes:

1) When converting user messages into DATA chunks, an endpoint will fragment user messages larger than the current association path MTU into multiple DATA chunks. The data receiver will normally reassemble the fragmented message from DATA chunks before delivery to the user (see Section 6.9 for details).
2) Multiple DATA and control chunks MAY be bundled by the sender into a single SCTP packet for transmission, as long as the final size of the packet does not exceed the current path MTU. The receiver will unbundle the packet back into the original chunks. Control chunks MUST come before DATA chunks in the packet.

The fragmentation and bundling mechanisms, as detailed in Section 6.9 and Section 6.10, are OPTIONAL to implement by the data sender, but they MUST be implemented by the data receiver, i.e., an endpoint MUST properly receive and process bundled or fragmented data.

6.1. Transmission of DATA Chunks

This document is specified as if there is a single retransmission timer per destination transport address, but implementations MAY have a retransmission timer for each DATA chunk.

The following general rules MUST be applied by the data sender for transmission and/or retransmission of outbound DATA chunks:

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).

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.

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. The sender SHOULD send the first zero window probe after 1 RTO when it detects that the receiver has closed its window and SHOULD increase the probe interval exponentially afterwards. Also note that the cwnd SHOULD be adjusted according to Section 7.2.1. Zero window probing does not affect the calculation of cwnd.

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].
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.

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.

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).

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:

\[
\text{if } ((\text{flightsize} + \text{Max.Burst} \times \text{MTU}) < \text{cwnd}) \\
\text{cwnd} = \text{flightsize} + \text{Max.Burst} \times \text{MTU};
\]

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 when using the formula above, cwnd SHOULD NOT be changed permanently.

E) Then, the sender can send out as many new DATA chunks as rule A and rule B allow.

Multiple DATA chunks committed for transmission MAY be bundled in a single packet. Furthermore, DATA chunks being retransmitted MAY be bundled with new DATA chunks, as long as the resulting packet size does not exceed the path MTU. A ULP can request that no bundling is performed, but this only turns off any delays that an SCTP implementation might be using to increase bundling efficiency. It does not in itself stop all bundling from occurring (i.e., in case of congestion or retransmission).

Before an endpoint transmits a DATA chunk, if any received DATA chunks have not been acknowledged (e.g., due to delayed ack), the sender SHOULD create a SACK and bundle it with the outbound DATA
chunk, as long as the size of the final SCTP packet does not exceed the current MTU. See Section 6.2.

IMPLEMENTATION NOTE: When the window is full (i.e., transmission is disallowed by rule A and/or rule B), the sender MAY still accept send requests from its upper layer, but MUST transmit no more DATA chunks until some or all of the outstanding DATA chunks are acknowledged and transmission is allowed by rule A and rule B again.

Whenever a transmission or retransmission is made to any address, if the T3-rtx timer of that address is not currently running, the sender MUST start that timer. If the timer for that address is already running, the sender MUST restart the timer if the earliest (i.e., lowest TSN) outstanding DATA chunk sent to that address is being retransmitted. Otherwise, the data sender MUST NOT restart the timer.

When starting or restarting the T3-rtx timer, the timer value SHOULD be adjusted according to the timer rules defined in Section 6.3.2 and Section 6.3.3.

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.

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, the following (see Section 4 of [RFC7053] for a discussion of the benefits):

- The application requests that the I bit of the last DATA chunk of a user message be set when providing the user message to the SCTP implementation (see Section 10.1).
- The sender is in the SHUTDOWN-PENDING state.
- The sending of a DATA chunk fills the congestion or receiver window.
6.2. Acknowledgement on Reception of DATA Chunks

The SCTP endpoint MUST always acknowledge the reception of each valid DATA chunk when the DATA chunk received is inside its receive window.

When the receiver’s advertised window is 0, the receiver MUST drop any new incoming DATA chunk with a TSN larger than the largest TSN received so far. If the new incoming DATA chunk holds a TSN value less than the largest TSN received so far, then the receiver SHOULD drop the largest TSN held for reordering and accept the new incoming DATA chunk. In either case, if such a DATA chunk is dropped, the receiver MUST immediately send back a SACK with the current receive window showing only DATA chunks received and accepted so far. The dropped DATA chunk(s) MUST NOT be included in the SACK, as they were not accepted. The receiver MUST also have an algorithm for advertising its receive window to avoid receiver silly window syndrome (SWS), as described in [RFC1122]. The algorithm can be similar to the one described in Section 4.2.3.3 of [RFC1122].

The guidelines on delayed acknowledgement algorithm specified in Section 4.2 of [RFC5681] SHOULD be followed. Specifically, an acknowledgement SHOULD be generated for at least every second packet (not every second DATA chunk) received, and SHOULD be generated within 200 ms of the arrival of any unacknowledged DATA chunk. In some situations, it might be beneficial for an SCTP transmitter to be more conservative than the algorithms detailed in this document allow. However, an SCTP transmitter MUST NOT be more aggressive than the following algorithms allow.

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.

IMPLEMENTATION NOTE: The maximum delay for generating an acknowledgement MAY be configured by the SCTP administrator, either statically or dynamically, in order to meet the specific timing requirement of the protocol being carried.

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.
Acknowledgements MUST be sent in SACK chunks unless shutdown was requested by the ULP, in which case an endpoint MAY send an acknowledgement in the SHUTDOWN chunk. A SACK chunk can acknowledge the reception of multiple DATA chunks. See Section 3.3.4 for SACK chunk format. In particular, the SCTP endpoint MUST fill in the Cumulative TSN Ack field to indicate the latest sequential TSN (of a valid DATA chunk) it has received. Any received DATA chunks with TSN greater than the value in the Cumulative TSN Ack field are reported in the Gap Ack Block fields. The SCTP endpoint MUST report as many Gap Ack Blocks as can fit in a single SACK chunk limited by the current path MTU.

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.

When a packet arrives with duplicate DATA chunk(s) and with no new DATA chunk(s), the endpoint MUST immediately send a SACK with no delay. If a packet arrives with duplicate DATA chunk(s) bundled with new DATA chunks, the endpoint MAY immediately send a SACK. Normally, receipt of duplicate DATA chunks will occur when the original SACK chunk was lost and the peer’s RTO has expired. The duplicate TSN number(s) SHOULD be reported in the SACK as duplicate.

When an endpoint receives a SACK, it MAY use the duplicate TSN information to determine if SACK loss is occurring. Further use of this data is for future study.

The data receiver is responsible for maintaining its receive buffers. The data receiver SHOULD notify the data sender in a timely manner of changes in its ability to receive data. How an implementation manages its receive buffers is dependent on many factors (e.g., operating system, memory management system, amount of memory, etc.). However, the data sender strategy defined in Section 6.2.1 is based on the assumption of receiver operation similar to the following:

A) At initialization of the association, the endpoint tells the peer how much receive buffer space it has allocated to the association in the INIT or INIT ACK. The endpoint sets a_rwnd to this value.

B) As DATA chunks are received and buffered, decrement a_rwnd by the number of bytes received and buffered. This is, in effect,
closing rwnd at the data sender and restricting the amount of data it can transmit.

C) As DATA chunks are delivered to the ULP and released from the receive buffers, increment a_rwnd by the number of bytes delivered to the upper layer. This is, in effect, opening up rwnd on the data sender and allowing it to send more data. The data receiver SHOULD NOT increment a_rwnd unless it has released bytes from its receive buffer. For example, if the receiver is holding fragmented DATA chunks in a reassembly queue, it SHOULD NOT increment a_rwnd.

D) When sending a SACK, the data receiver SHOULD place the current value of a_rwnd into the a_rwnd field. The data receiver SHOULD take into account that the data sender will not retransmit DATA chunks that are acknowledged via the Cumulative TSN Ack (i.e., will drop from its retransmit queue).

Under certain circumstances, the data receiver MAY drop DATA chunks that it has received but hasn’t released from its receive buffers (i.e., delivered to the ULP). These DATA chunks might have been acknowledged in Gap Ack Blocks. For example, the data receiver might be holding data in its receive buffers while reassembling a fragmented user message from its peer when it runs out of receive buffer space. It MAY drop these DATA chunks even though it has acknowledged them in Gap Ack Blocks. If a data receiver drops DATA chunks, it MUST NOT include them in Gap Ack Blocks in subsequent SACKs until they are received again via retransmission. In addition, the endpoint SHOULD take into account the dropped data when calculating its a_rwnd.

An endpoint SHOULD NOT revoke a SACK and discard data. Only in extreme circumstances might an endpoint use this procedure (such as out of buffer space). The data receiver SHOULD take into account that dropping data that has been acknowledged in Gap Ack Blocks can result in suboptimal retransmission strategies in the data sender and thus in suboptimal performance.

The following example illustrates the use of delayed acknowledgements:
Endpoint A

{App sends 3 messages; strm 0}
DATA [TSN=7,Strm=0,Seq=3] -------------> (ack delayed)
(Start T3-rtx timer)

DATA [TSN=8,Strm=0,Seq=4] -------------> (send ack)
(cancel T3-rtx timer) <-----/

DATA [TSN=9,Strm=0,Seq=5] -------------> (ack delayed)
(Start T3-rtx timer)

...  
{App sends 1 message; strm 1}

(bundle SACK with DATA)

SACK [TSN Ack=9,block=0] \ 
DATA [TSN=6,Strm=1,Seq=2]

(cancel T3-rtx timer) <------/ (Start T3-rtx timer)

(ack delayed)

(send ack)

SACK [TSN Ack=6,block=0] -------------> (cancel T3-rtx timer)

Figure 7: Delayed Acknowledgement Example

If an endpoint receives a DATA chunk with no user data (i.e., the Length field is set to 16), it MUST send an ABORT with error cause set to "No User Data".

An endpoint SHOULD NOT send a DATA chunk with no user data part.

6.2.1. Processing a Received SACK

Each SACK an endpoint receives contains an a_rwnd value. This value represents the amount of buffer space the data receiver, at the time of transmitting the SACK, has left of its total receive buffer space (as specified in the INIT/INIT ACK). Using a_rwnd, Cumulative TSN Ack, and Gap Ack Blocks, the data sender can develop a representation of the peer’s receive buffer space.

One of the problems the data sender takes into account when processing a SACK is that a SACK can be received out of order. That is, a SACK sent by the data receiver can pass an earlier SACK and be received first by the data sender. If a SACK is received out of order, the data sender can develop an incorrect view of the peer’s receive buffer space.
Since there is no explicit identifier that can be used to detect out-of-order SACKs, the data sender uses heuristics to determine if a SACK is new.

An endpoint SHOULD use the following rules to calculate the rwnd, using the a_rwnd value, the Cumulative TSN Ack, and Gap Ack Blocks in a received SACK.

A) At the establishment of the association, the endpoint initializes the rwnd to the Advertised Receiver Window Credit (a_rwnd) the peer specified in the INIT or INIT ACK.

B) Any time a DATA chunk is transmitted (or retransmitted) to a peer, the endpoint subtracts the data size of the chunk from the rwnd of that peer.

C) Any time a DATA chunk is marked for retransmission, either via T3-rtx timer expiration (Section 6.3.3) or via Fast Retransmit (Section 7.2.4), add the data size of those chunks to the rwnd.

Note: If the implementation is maintaining a timer on each DATA chunk, then only DATA chunks whose timer expired would be marked for retransmission.

D) Any time a SACK arrives, the endpoint performs the following:

i) If Cumulative TSN Ack is less than the Cumulative TSN Ack Point, then drop the SACK. Since Cumulative TSN Ack is monotonically increasing, a SACK whose Cumulative TSN Ack is less than the Cumulative TSN Ack Point indicates an out-of-order SACK.

ii) Set rwnd equal to the newly received a_rwnd minus the number of bytes still outstanding after processing the Cumulative TSN Ack and the Gap Ack Blocks.

iii) If the SACK is missing a TSN that was previously acknowledged via a Gap Ack Block (e.g., the data receiver reneged on the data), then consider the corresponding DATA that might be possibly missing: Count one miss indication towards Fast Retransmit as described in Section 7.2.4, and if no retransmit timer is running for the destination address to which the DATA chunk was originally transmitted, then T3-rtx is started for that destination address.

iv) If the Cumulative TSN Ack matches or exceeds the Fast Recovery exitpoint (Section 7.2.4), Fast Recovery is exited.
6.3. Management of Retransmission Timer

An SCTP endpoint uses a retransmission timer T3-rtx to ensure data delivery in the absence of any feedback from its peer. The duration of this timer is referred to as RTO (retransmission timeout).

When an endpoint’s peer is multi-homed, the endpoint will calculate a separate RTO for each different destination transport address of its peer endpoint.

The computation and management of RTO in SCTP follow closely how TCP manages its retransmission timer. To compute the current RTO, an endpoint maintains two state variables per destination transport address: SRTT (smoothed round-trip time) and RTTVAR (round-trip time variation).

6.3.1. RTO Calculation

The rules governing the computation of SRTT, RTTVAR, and RTO are as follows:

C1) Until an RTT measurement has been made for a packet sent to the given destination transport address, set RTO to the protocol parameter ‘RTO.Initial’.
C2) When the first RTT measurement R is made, set

\[
SRTT \leftarrow R, \\
RTTVAR \leftarrow R/2, \text{ and} \\
RTO \leftarrow SRTT + 4 \times RTTVAR.
\]

C3) When a new RTT measurement R’ is made, set

\[
RTTVAR \leftarrow (1 - RTO.Beta) \times RTTVAR + RTO.Beta \times |SRTT - R'| \\
\text{and} \\
SRTT \leftarrow (1 - RTO.Alpha) \times SRTT + RTO.Alpha \times R'
\]

Note: The value of SRTT used in the update to RTTVAR is its value before updating SRTT itself using the second assignment.

After the computation, update RTO \leftarrow SRTT + 4 \times RTTVAR.

C4) When data is in flight and when allowed by rule C5 below, a new RTT measurement MUST be made each round trip. Furthermore, new RTT measurements SHOULD be made no more than once per round trip for a given destination transport address. There are two reasons for this recommendation: First, it appears that
measuring more frequently often does not in practice yield any significant benefit [ALLMAN99]; second, if measurements are made more often, then the values of RTO.Alpha and RTO.Beta in rule C3 above SHOULD be adjusted so that SRTT and RTTVAR still adjust to changes at roughly the same rate (in terms of how many round trips it takes them to reflect new values) as they would if making only one measurement per round-trip and using RTO.Alpha and RTO.Beta as given in rule C3. However, the exact nature of these adjustments remains a research issue.

C5) Karn’s algorithm: RTT measurements MUST NOT be made using packets that were retransmitted (and thus for which it is ambiguous whether the reply was for the first instance of the chunk or for a later instance)

IMPLEMENTATION NOTE: RTT measurements SHOULD only be made using a chunk with TSN r if no chunk with TSN less than or equal to r is retransmitted since r is first sent.

C6) Whenever RTO is computed, if it is less than RTO.Min seconds then it is rounded up to RTO.Min seconds. The reason for this rule is that RTOs that do not have a high minimum value are susceptible to unnecessary timeouts [ALLMAN99].

C7) A maximum value MAY be placed on RTO provided it is at least RTO.max seconds.

There is no requirement for the clock granularity G used for computing RTT measurements and the different state variables, other than:

G1) Whenever RTTVAR is computed, if RTTVAR = 0, then adjust RTTVAR <- G.

Experience [ALLMAN99] has shown that finer clock granularities (<= 100 msec) perform somewhat better than more coarse granularities.

6.3.2. Retransmission Timer Rules

The rules for managing the retransmission timer are as follows:

R1) Every time a DATA chunk is sent to any address (including a retransmission), if the T3-rtx timer of that address is not running, start it running so that it will expire after the RTO of that address. The RTO used here is that obtained after any doubling due to previous T3-rtx timer expirations on the corresponding destination address as discussed in rule E2 below.
R2) Whenever all outstanding data sent to an address have been acknowledged, turn off the T3-rtx timer of that address.

R3) Whenever a SACK is received that acknowledges the DATA chunk with the earliest outstanding TSN for that address, restart the T3-rtx timer for that address with its current RTO (if there is still outstanding data on that address).

R4) Whenever a SACK is received missing a TSN that was previously acknowledged via a Gap Ack Block, start the T3-rtx for the destination address to which the DATA chunk was originally transmitted if it is not already running.

The following example shows the use of various timer rules (assuming that the receiver uses delayed acks).

```
Endpoint A                                      Endpoint Z
        (App begins to send)
Data [TSN=7,Strm=0,Seq=3] ------------> (ack delayed)
        (Start T3-rtx timer)
        (App sends 1 message; strm 1)
        (bundle ack with data)
DATA [TSN=8,Strm=0,Seq=4] ----\     /-- SACK [TSN Ack=7,Block=0]
\   /     DATA [TSN=6,Strm=1,Seq=2]
\ /     (Start T3-rtx timer)
\/
(Start T3-rtx timer)  <------/   \--> (ack delayed)
(ack delayed)
(send ack)
SACK [TSN Ack=6,Block=0] ----------------> (Cancel T3-rtx timer)
        ..
        (send ack)
(Cancel T3-rtx timer)  <------------- SACK [TSN Ack=8,Block=0]
```

Figure 8: Timer Rule Examples

6.3.3. Handle T3-rtx Expiration

Whenever the retransmission timer T3-rtx expires for a destination address, do the following:

E1) For the destination address for which the timer expires, adjust its ssthresh with rules defined in Section 7.2.3 and set the cwnd <- MTU.

E2) For the destination address for which the timer expires, set RTO <- RTO * 2 ("back off the timer"). The maximum value discussed...
in rule C7 above (RTO.max) MAY be used to provide an upper bound to this doubling operation.

E3) Determine how many of the earliest (i.e., lowest TSN) outstanding DATA chunks for the address for which the T3-rtx has expired will fit into a single packet, subject to the MTU constraint for the path corresponding to the destination transport address to which the retransmission is being sent (this might be different from the address for which the timer expires; see Section 6.4). Call this value K. Bundle and retransmit those K DATA chunks in a single packet to the destination endpoint.

E4) Start the retransmission timer T3-rtx on the destination address to which the retransmission is sent, if rule R1 above indicates to do so. The RTO to be used for starting T3-rtx SHOULD be the one for the destination address to which the retransmission is sent, which, when the receiver is multi-homed, might be different from the destination address for which the timer expired (see Section 6.4 below).

After retransmitting, once a new RTT measurement is obtained (which can happen only when new data has been sent and acknowledged, per rule C5, or for a measurement made from a HEARTBEAT; see Section 8.3), the computation in rule C3 is performed, including the computation of RTO, which might result in "collapsing" RTO back down after it has been subject to doubling (rule E2).

Note: Any DATA chunks that were sent to the address for which the T3-rtx timer expired but did not fit in one MTU (rule E3 above) SHOULD be marked for retransmission and sent as soon as cwnd allows (normally, when a SACK arrives).

The final rule for managing the retransmission timer concerns failover (see Section 6.4.1):

F1) Whenever an endpoint switches from the current destination transport address to a different one, the current retransmission timers are left running. As soon as the endpoint transmits a packet containing DATA chunk(s) to the new transport address, start the timer on that transport address, using the RTO value of the destination address to which the data is being sent, if rule R1 indicates to do so.

6.4. Multi-Homed SCTP Endpoints

An SCTP endpoint is considered multi-homed if there are more than one transport address that can be used as a destination address to reach that endpoint.
Moreover, the ULP of an endpoint selects one of the multiple
destination addresses of a multi-homed peer endpoint as the primary
path (see Section 5.1.2 and Section 10.1 for details).

By default, an endpoint SHOULD always transmit to the primary path,
unless the SCTP user explicitly specifies the destination transport
address (and possibly source transport address) to use.

An endpoint SHOULD transmit reply chunks (e.g., INIT ACK, COOKIE ACK,
HEARTBEAT ACK) 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.

When a receiver of a duplicate DATA chunk sends a SACK to a multi-
homed endpoint, it MAY be beneficial to vary the destination address
and not use the source address of the DATA chunk. The reason is that
receiving a duplicate from a multi-homed endpoint might indicate that
the return path (as specified in the source address of the DATA
chunk) for the SACK is broken.

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 to which
the original data was sent. 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.

Retransmissions do not affect the total outstanding data count.
However, if the DATA chunk is retransmitted onto a different
destination address, both the outstanding data counts on the new
destination address and the old destination address to which the data
chunk was last sent is adjusted accordingly.
6.4.1. Failover from an Inactive Destination Address

Some of the transport addresses of a multi-homed SCTP endpoint might become inactive due to either the occurrence of certain error conditions (see Section 8.2) or adjustments from the SCTP user.

When there is outbound data to send and the primary path becomes inactive (e.g., due to failures), or where the SCTP user explicitly requests to send data to an inactive destination transport address, before reporting an error to its ULP, the SCTP endpoint SHOULD try to send the data to an alternate active destination transport address if one exists.

When retransmitting data that timed out, if the endpoint is multi-homed, it needs to consider each source-destination address pair in its retransmission selection policy. When retransmitting timed-out data, the endpoint SHOULD attempt to pick the most divergent source-destination pair from the original source-destination pair to which the packet was transmitted.

Note: Rules for picking the most divergent source-destination pair are an implementation decision and are not specified within this document.

6.5. Stream Identifier and Stream Sequence Number

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.

The Stream Sequence Number in all the streams MUST start from 0 when the association is established. Also, when the Stream Sequence Number reaches the value 65535 the next Stream Sequence Number MUST be set to 0.

6.6. Ordered and Unordered Delivery

Within a stream, an endpoint MUST deliver DATA chunks received with the U flag set to 0 to the upper layer according to the order of their Stream Sequence Number. If DATA chunks arrive out of order of their Stream Sequence Number, the endpoint MUST hold the received DATA chunks from delivery to the ULP until they are reordered.
However, an SCTP endpoint can indicate that no ordered delivery is required for a particular DATA chunk transmitted within the stream by setting the U flag of the DATA chunk to 1.

When an endpoint receives a DATA chunk with the U flag set to 1, it bypasses the ordering mechanism and immediately deliver the data to the upper layer (after reassembly if the user data is fragmented by the data sender).

This provides an effective way of transmitting "out-of-band" data in a given stream. Also, a stream can be used as an "unordered" stream by simply setting the U flag to 1 in all DATA chunks sent through that stream.

IMPLEMENTATION NOTE: When sending an unordered DATA chunk, an implementation MAY choose to place the DATA chunk in an outbound packet that is at the head of the outbound transmission queue if possible.

The ‘Stream Sequence Number’ field in a DATA chunk with U flag set to 1 has no significance. The sender can fill it with arbitrary value, but the receiver MUST ignore the field.

Note: When transmitting ordered and unordered data, an endpoint does not increment its Stream Sequence Number when transmitting a DATA chunk with U flag set to 1.

6.7. Report Gaps in Received DATA TSNs

Upon the reception of a new DATA chunk, an endpoint examines the continuity of the TSNs received. If the endpoint detects a gap in the received DATA chunk sequence, it SHOULD send a SACK with Gap Ack Blocks immediately. The data receiver continues sending a SACK after receipt of each SCTP packet that doesn’t fill the gap.

Based on the Gap Ack Block from the received SACK, the endpoint can calculate the missing DATA chunks and make decisions on whether to retransmit them (see Section 6.2.1 for details).

Multiple gaps can be reported in one single SACK (see Section 3.3.4).

When its peer is multi-homed, the SCTP endpoint SHOULD always try to send the SACK to the same destination address from which the last DATA chunk was received.

Upon the reception of a SACK, the endpoint MUST remove all DATA chunks that have been acknowledged by the SACK’s Cumulative TSN Ack from its transmit queue. The endpoint MUST also treat all the DATA
chunks with TSNs not included in the Gap Ack Blocks reported by the SACK as "missing". The number of "missing" reports for each outstanding DATA chunk MUST be recorded by the data sender in order to make retransmission decisions. See Section 7.2.4 for details.

The following example shows the use of SACK to report a gap.

```
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)
```

Figure 9: Reporting a Gap using SACK

The maximum number of Gap Ack Blocks that can be reported within a single SACK chunk is limited by the current path MTU. When a single SACK cannot cover all the Gap Ack Blocks needed to be reported due to the MTU limitation, the endpoint MUST send only one SACK, reporting the Gap Ack Blocks from the lowest to highest TSNs, within the size limit set by the MTU, and leave the remaining highest TSN numbers unacknowledged.

### 6.8. CRC32c Checksum Calculation

When sending an SCTP packet, the endpoint MUST strengthen the data integrity of the transmission by including the CRC32c checksum value calculated on the packet, as described below.

After the packet is constructed (containing the SCTP common header and one or more control or DATA chunks), the transmitter MUST

1) fill in the proper Verification Tag in the SCTP common header and initialize the checksum field to '0's,

2) calculate the CRC32c checksum of the whole packet, including the SCTP common header and all the chunks (refer to Appendix B for details of the CRC32c algorithm); and
3) put the resultant value into the checksum field in the common header, and leave the rest of the bits unchanged.

When an SCTP packet is received, the receiver MUST first check the CRC32c checksum as follows:

1) Store the received CRC32c checksum value aside.
2) Replace the 32 bits of the checksum field in the received SCTP packet with all ‘0’ s and calculate a CRC32c checksum value of the whole received packet.
3) Verify that the calculated CRC32c checksum is the same as the received CRC32c checksum. If it is not, the receiver MUST treat the packet as an invalid SCTP packet.

The default procedure for handling invalid SCTP packets is to silently discard them.

Any hardware implementation SHOULD be done in a way that is verifiable by the software.

6.9. Fragmentation and Reassembly

An endpoint MAY support fragmentation when sending DATA chunks, but it MUST support reassembly when receiving DATA chunks. If an endpoint supports fragmentation, it MUST fragment a user message if the size of the user message to be sent causes the outbound SCTP packet size to exceed the current MTU. If an implementation does not support fragmentation of outbound user messages, the endpoint MUST return an error to its upper layer and not attempt to send the user message.

Note: If an implementation that supports fragmentation makes available to its upper layer a mechanism to turn off fragmentation, it might do so. However, in so doing, it MUST react just like an implementation that does NOT support fragmentation, i.e., it MUST reject sends that exceed the current Path MTU (P-MTU).

IMPLEMENTATION NOTE: In this error case, the Send primitive discussed in Section 10.1 would need to return an error to the upper layer.

If its peer is multi-homed, the endpoint SHOULD choose a size no larger than the association Path MTU. The association Path MTU is the smallest Path MTU of all destination addresses.

Note: Once a message is fragmented, it cannot be re-fragmented. Instead, if the PMTU has been reduced, then IP fragmentation MUST be used. Please see Section 7.3 for details of PMTU discovery.
When determining when to fragment, the SCTP implementation MUST take into account the SCTP packet header as well as the DATA chunk header(s). The implementation MUST also take into account the space required for a SACK chunk if bundling a SACK chunk with the DATA chunk.

Fragmentation takes the following steps:

1) The data sender MUST break the user message into a series of DATA chunks such that each chunk plus SCTP overhead fits into an IP datagram smaller than or equal to the association Path MTU.

2) The transmitter MUST then assign, in sequence, a separate TSN to each of the DATA chunks in the series. The transmitter assigns the same SSN to each of the DATA chunks. If the user indicates that the user message is to be delivered using unordered delivery, then the U flag of each DATA chunk of the user message MUST be set to 1.

3) The transmitter MUST also set the B/E bits of the first DATA chunk in the series to ‘10’, the B/E bits of the last DATA chunk in the series to ‘01’, and the B/E bits of all other DATA chunks in the series to ‘00’.

An endpoint MUST recognize fragmented DATA chunks by examining the B/E bits in each of the received DATA chunks, and queue the fragmented DATA chunks for reassembly. Once the user message is reassembled, SCTP passes the reassembled user message to the specific stream for possible reordering and final dispatching.

Note: If the data receiver runs out of buffer space while still waiting for more fragments to complete the reassembly of the message, it SHOULD dispatch part of its inbound message through a partial delivery API (see Section 10), freeing some of its receive buffer space so that the rest of the message can be received.

6.10. Bundling

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.

If its peer endpoint is multi-homed, the sending endpoint SHOULD choose a size no larger than the latest MTU of the current primary path.
When bundling control chunks with DATA chunks, an endpoint MUST place control chunks first in the outbound SCTP packet. The transmitter MUST transmit DATA chunks within an SCTP packet in increasing order of TSN.

Note: Since control chunks are placed first in a packet and since DATA chunks are transmitted before SHUTDOWN or SHUTDOWN ACK chunks, DATA chunks cannot be bundled with SHUTDOWN or SHUTDOWN ACK chunks.

Partial chunks MUST NOT be placed in an SCTP packet. A partial chunk is a chunk that is not completely contained in the SCTP packet; i.e., the SCTP packet is too short to contain all the bytes of the chunk as indicated by the chunk length.

An endpoint MUST process received chunks in their order in the packet. The receiver uses the Chunk Length field to determine the end of a chunk and beginning of the next chunk taking account of the fact that all chunks end on a 4-byte boundary. If the receiver detects a partial chunk, it MUST drop the chunk.

An endpoint MUST NOT bundle INIT, INIT ACK, or SHUTDOWN COMPLETE with any other chunks.

7. Congestion Control

Congestion control is one of the basic functions in SCTP. For some applications, it might be likely that adequate resources will be allocated to SCTP traffic to ensure prompt delivery of time-critical data -- thus, it would appear to be unlikely, during normal operations, that transmissions encounter severe congestion conditions. However, SCTP operates under adverse operational conditions, which can develop upon partial network failures or unexpected traffic surges. In such situations, SCTP follows correct congestion control steps to recover from congestion quickly in order to get data delivered as soon as possible. In the absence of network congestion, these preventive congestion control algorithms are expected to show no impact on the protocol performance.

IMPLEMENTATION NOTE: As far as its specific performance requirements are met, an implementation is always allowed to adopt a more conservative congestion control algorithm than the one defined below.

The congestion control algorithms used by SCTP are based on [RFC5681]. This section describes how the algorithms defined in [RFC5681] are adapted for use in SCTP. We first list differences in protocol designs between TCP and SCTP, and then describe SCTP’s congestion control scheme. The description will use the same terminology as in TCP congestion control whenever appropriate.
SCTP congestion control is always applied to the entire association, and not to individual streams.

7.1. SCTP Differences from TCP Congestion Control

Gap Ack Blocks in the SCTP SACK carry the same semantic meaning as the TCP SACK. TCP considers the information carried in the SACK as advisory information only. SCTP considers the information carried in the Gap Ack Blocks in the SACK chunk as advisory. In SCTP, any DATA chunk that has been acknowledged by SACK, including DATA that arrived at the receiving end out of order, is not considered fully delivered until the Cumulative TSN Ack Point passes the TSN of the DATA chunk (i.e., the DATA chunk has been acknowledged by the Cumulative TSN Ack field in the SACK). Consequently, the value of cwnd controls the amount of outstanding data, rather than (as in the case of non-SACK TCP) the upper bound between the highest acknowledged sequence number and the latest DATA chunk that can be sent within the congestion window. SCTP SACK leads to different implementations of Fast Retransmit and Fast Recovery than non-SACK TCP. As an example, see [FALL96].

The biggest difference between SCTP and TCP, however, is multi-homing. SCTP is designed to establish robust communication associations between two endpoints each of which might be reachable by more than one transport address. Potentially different addresses might lead to different data paths between the two endpoints; thus, ideally one needs a separate set of congestion control parameters for each of the paths. The treatment here of congestion control for multi-homed receivers is new with SCTP and might require refinement in the future. The current algorithms make the following assumptions:

- The sender usually uses the same destination address until being instructed by the upper layer to do otherwise; however, SCTP MAY change to an alternate destination in the event an address is marked inactive (see Section 8.2). Also, SCTP MAY retransmit to a different transport address than the original transmission.

- The sender keeps a separate congestion control parameter set for each of the destination addresses it can send to (not each source-destination pair but for each destination). The parameters SHOULD decay if the address is not used for a long enough time period.

- For each of the destination addresses, an endpoint does slow start upon the first transmission to that address.

Note: TCP guarantees in-sequence delivery of data to its upper-layer protocol within a single TCP session. This means that when TCP
notices a gap in the received sequence number, it waits until the gap is filled before delivering the data that was received with sequence numbers higher than that of the missing data. On the other hand, SCTP can deliver data to its upper-layer protocol even if there is a gap in TSN if the Stream Sequence Numbers are in sequence for a particular stream (i.e., the missing DATA chunks are for a different stream) or if unordered delivery is indicated. Although this does not affect cwnd, it might affect rwnd calculation.

7.2. SCTP Slow-Start and Congestion Avoidance

The slow-start and congestion avoidance algorithms MUST be used by an endpoint to control the amount of data being injected into the network. The congestion control in SCTP is employed in regard to the association, not to an individual stream. In some situations, it might be beneficial for an SCTP sender to be more conservative than the algorithms allow; however, an SCTP sender MUST NOT be more aggressive than the following algorithms allow.

Like TCP, an SCTP endpoint uses the following three control variables to regulate its transmission rate.

- Receiver advertised window size (rwnd, in bytes), which is set by the receiver based on its available buffer space for incoming packets.
  Note: This variable is kept on the entire association.
- Congestion control window (cwnd, in bytes), which is adjusted by the sender based on observed network conditions.
  Note: This variable is maintained on a per-destination-address basis.
- Slow-start threshold (ssthresh, in bytes), which is used by the sender to distinguish slow-start and congestion avoidance phases.
  Note: This variable is maintained on a per-destination-address basis.

SCTP also requires one additional control variable, partial_bytes_acked, which is used during congestion avoidance phase to facilitate cwnd adjustment.

Unlike TCP, an SCTP sender MUST keep a set of these control variables cwnd, ssthresh, and partial_bytes_acked for EACH destination address of its peer (when its peer is multi-homed). Only one rwnd is kept for the whole association (no matter if the peer is multi-homed or has a single address).
7.2.1. Slow-Start

Beginning data transmission into a network with unknown conditions or after a sufficiently long idle period requires SCTP to probe the network to determine the available capacity. The slow-start algorithm is used for this purpose at the beginning of a transfer, or after repairing loss detected by the retransmission timer.

- The initial cwnd before data transmission MUST be set to min(4*MTU, max (2*MTU, 4380 bytes)).
- 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.
- The initial value of sssthresh SHOULD be arbitrarily high (e.g., the size of the largest possible advertised window).
- 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 Section 6.1.B) SHOULD be supported.
- When cwnd is less than or equal to sssthresh, an SCTP endpoint MUST use the slow-start algorithm to increase cwnd only if the current congestion window is being fully utilized, an incoming SACK advances the Cumulative TSN Ack Point, and the data sender is not in Fast Recovery. Only when these three conditions are met can the cwnd be increased; otherwise, the cwnd MUST NOT be increased. If these conditions are met, then cwnd MUST be increased by, at most, the lesser of 1) the total size of the previously outstanding DATA chunk(s) acknowledged, and 2) the destination’s path MTU. This upper bound protects against the ACK-Splitting attack outlined in [SAVAGE99].

In instances where its peer endpoint is multi-homed, if an endpoint receives a SACK that advances its Cumulative TSN Ack Point, then it SHOULD update its cwnd (or cwnds) apportioned to the destination addresses to which it transmitted the acknowledged data. However, if the received SACK does not advance the Cumulative TSN Ack Point, the endpoint MUST NOT adjust the cwnd of any of the destination addresses.

Because an endpoint’s cwnd is not tied to its Cumulative TSN Ack Point, as duplicate SACKs come in, even though they might not advance the Cumulative TSN Ack Point an endpoint can still use them to clock out new data. That is, the data newly acknowledged by the SACK diminishes the amount of data now in flight to less than cwnd, and so the current, unchanged value of cwnd now allows new data to be sent. On the other hand, the increase of cwnd MUST be tied to the Cumulative TSN Ack Point advancement as specified above. Otherwise, the duplicate SACKs will not only clock out new data, but also will...
adversely clock out more new data than what has just left the network, during a time of possible congestion.

- While the endpoint does not transmit data on a given transport address, the cwnd of the transport address SHOULD be adjusted to max(cwnd/2, 4*MTU) once per RTO. Before the first cwnd adjustment, the ssthresh of the transport address SHOULD be set to the cwnd.

### 7.2.2. Congestion Avoidance

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 as follows:

- SCTP MAY increment cwnd by 1*MTU.
- SCTP SHOULD increment cwnd by 1*MTU once per RTT when the sender has cwnd or more bytes of data outstanding for the corresponding transport address.
- SCTP MUST NOT increment cwnd by more than 1*MTU per RTT.

In practice, an implementation can achieve this goal in the following way:

- partial_bytes_acked is initialized to 0.
- 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.
- (1) when partial_bytes_acked is greater than cwnd and (2) before the arrival of the SACK the sender had less than cwnd bytes of data outstanding (i.e., before the arrival of the SACK, flightsize was less than cwnd), reset partial_bytes_acked to cwnd.
- (1) when partial_bytes_acked is equal to or greater than cwnd and (2) before the arrival of the SACK the sender had cwnd or more bytes of data outstanding (i.e., before the 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.
- Same as in the slow start, when the sender does not transmit DATA on a given transport address, the cwnd of the transport address SHOULD be adjusted to max(cwnd / 2, 4*MTU) per RTO.
When all of the data transmitted by the sender has been acknowledged by the receiver, partial_bytes_acked is initialized to 0.

### 7.2.3. Congestion Control

Upon detection of packet losses from SACK (see Section 7.2.4), an endpoint SHOULD do the following:

\[
\text{ssthresh} = \max\left(\frac{cwnd}{2}, 4*\text{MTU}\right) \\
cwnd = \text{ssthresh} \\
\text{partial_bytes_acked} = 0
\]

Basically, a packet loss causes cwnd to be cut in half.

When the T3-rtx timer expires on an address, SCTP SHOULD perform slow start by:

\[
\text{ssthresh} = \max\left(\frac{cwnd}{2}, 4*\text{MTU}\right) \\
cwnd = 1*\text{MTU} \\
\text{partial_bytes_acked} = 0
\]

and ensure that no more than one SCTP packet will be in flight for that address until the endpoint receives acknowledgement for successful delivery of data to that address.

### 7.2.4. Fast Retransmit on Gap Reports

In the absence of data loss, an endpoint performs delayed acknowledgement. However, whenever an endpoint notices a hole in the arriving TSN sequence, it SHOULD start sending a SACK back every time a packet arrives carrying data until the hole is filled.

Whenever an endpoint receives a SACK that indicates that some TSNs are missing, it SHOULD wait for two further miss indications (via subsequent SACKs for a total of three missing reports) on the same TSNs before taking action with regard to Fast Retransmit.

Miss indications SHOULD follow the HTNA (Highest TSN Newly Acknowledged) algorithm. For each incoming SACK, miss indications are incremented only for missing TSNs prior to the highest TSN newly acknowledged in the SACK. A newly acknowledged DATA chunk is one not previously acknowledged in a SACK. If an endpoint is in Fast Recovery and a SACK arrives that advances the Cumulative TSN Ack Point, the miss indications are incremented for all TSNs reported missing in the SACK.
When the third consecutive miss indication is received for a TSN(s), the data sender does the following:

1) Mark the DATA chunk(s) with three miss indications for retransmission.
2) If not in Fast Recovery, adjust the ssthresh and cwnd of the destination address(es) to which the missing DATA chunks were last sent, according to the formula described in Section 7.2.3.
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.
4) Restart the T3-rtx timer only if the last SACK acknowledged the lowest outstanding TSN number sent to that address, or the endpoint is retransmitting the first outstanding DATA chunk sent to that address.
5) Mark the DATA chunk(s) as being fast retransmitted and thus ineligible for a subsequent Fast Retransmit. Those TSNs marked for retransmission due to the Fast-Retransmit algorithm that did not fit in the sent datagram carrying K other TSNs are also marked as ineligible for a subsequent Fast Retransmit. However, as they are marked for retransmission they will be retransmitted later on as soon as cwnd allows.
6) If not in Fast Recovery, enter Fast Recovery and mark the highest outstanding TSN as the Fast Recovery exit point. When a SACK acknowledges all TSNs up to and including this exit point, Fast Recovery is exited. While in Fast Recovery, the ssthresh and cwnd SHOULD NOT change for any destinations due to a subsequent Fast Recovery event (i.e., one SHOULD NOT reduce the cwnd further due to a subsequent Fast Retransmit).

Note: Before the above adjustments, if the received SACK also acknowledges new DATA chunks and advances the Cumulative TSN Ack Point, the cwnd adjustment rules defined in Section 7.2.1 and Section 7.2.2 MUST be applied first.

7.2.5. Making Changes to 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.
7.3. Path MTU Discovery

[RFC4821], [RFC8201], and [RFC1191] specify "Packetization Layer Path MTU Discovery", whereby an endpoint maintains an estimate of the maximum transmission unit (MTU) along a given Internet path and refrains from sending packets along that path that exceed the MTU, other than occasional attempts to probe for a change in the Path MTU (PMTU). [RFC4821] is thorough in its discussion of the MTU discovery mechanism and strategies for determining the current end-to-end MTU setting as well as detecting changes in this value.

An endpoint SHOULD apply these techniques, and SHOULD do so on a per-destination-address basis.

There are two important SCTP-specific points regarding Path MTU discovery:

1) SCTP associations can span multiple addresses. An endpoint MUST maintain separate MTU estimates for each destination address of its peer.

2) The sender SHOULD track an association PMTU that will be the smallest PMTU discovered for all of the peer’s destination addresses. When fragmenting messages into multiple parts this association PMTU SHOULD be used to calculate the size of each fragment. This will allow retransmissions to be seamlessly sent to an alternate address without encountering IP fragmentation.

8. Fault Management

8.1. Endpoint Failure Detection

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 that 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 that is currently 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.

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.

8.2. Path Failure Detection

When its peer endpoint is multi-homed, an endpoint SHOULD keep an error counter for each of the destination transport addresses of the peer endpoint.

Each time the T3-rtx timer expires on any address, or when a HEARTBEAT sent to an idle address is not acknowledged within an RTO, the error counter of that destination address will be incremented. When the value in the error counter exceeds the protocol parameter ‘Path.Max.Retrans’ of that destination address, the endpoint SHOULD mark the destination transport address as inactive, and a notification SHOULD be sent to the upper layer.

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 have significant consequences for 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.

Note: When configuring the SCTP endpoint, the user SHOULD avoid having the value of ‘Association.Max.Retrans’ larger than the summation of the ‘Path.Max.Retrans’ of all the destination addresses for the remote endpoint. Otherwise, all the destination addresses might become inactive while the endpoint still considers the peer endpoint reachable. When this condition occurs, how SCTP chooses to function is implementation specific.
When the primary path is marked inactive (due to excessive retransmissions, for instance), the sender MAY automatically transmit new packets to an alternate destination address if one exists and is active. If more than one alternate address is active when the primary path is marked inactive, only ONE transport address SHOULD be chosen and used as the new destination transport address.

8.3. Path Heartbeat

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).

A destination transport address is considered "idle" if no new chunk that can be used for updating path RTT (usually including first transmission DATA, INIT, COOKIE ECHO, HEARTBEAT, etc.) and no HEARTBEAT has been sent to it within the current heartbeat period of that address. This applies to both active and inactive destination addresses.

The upper layer can optionally initiate the following functions:

A) Disable heartbeat on a specific destination transport address of a given association,

B) Change the HB.interval,

C) Re-enable heartbeat on a specific destination transport address of a given association, and

D) Request an on-demand HEARTBEAT on a specific destination transport address of a given association.

The endpoint SHOULD increment the respective error counter of the destination transport address each time a HEARTBEAT is sent to that address and not acknowledged within one RTO.

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 in reachability of this...
destination address. After this, the endpoint SHOULD continue HEARTBEAT on this destination address but SHOULD stop increasing the counter.

The sender of the HEARTBEAT chunk SHOULD include in the Heartbeat Information field of the chunk the current time when the packet is sent out and the destination address to which the packet is sent.

IMPLEMENTATION NOTE: An alternative implementation of the heartbeat mechanism that can be used is to increment the error counter variable every time a HEARTBEAT is sent to a destination. Whenever a HEARTBEAT ACK arrives, the sender SHOULD clear the error counter of the destination that the HEARTBEAT was sent to. This in effect would clear the previously stroked error (and any other error counts as well).

The receiver of the HEARTBEAT SHOULD immediately respond with a HEARTBEAT ACK that contains the Heartbeat Information TLV, together with any other received TLVs, copied unchanged from the received HEARTBEAT chunk.

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 count (as defined in Section 8.1).

The receiver of the HEARTBEAT ACK SHOULD also perform an RTT measurement for that destination transport address using the time value carried in the HEARTBEAT ACK chunk.

On an idle destination address that is allowed to heartbeat, it is RECOMMENDED that a HEARTBEAT chunk is sent once per RTO of that destination address plus the protocol parameter ‘HB.interval’, with jittering of +/- 50% of the RTO value, and exponential backoff of the RTO if the previous HEARTBEAT is unanswered.

A primitive is provided for the SCTP user to change the HB.interval and turn on or off the heartbeat on a given destination address. The heartbeat interval set by the SCTP user is added to the RTO of that destination (including any exponential backoff). Only one heartbeat SHOULD be sent each time the heartbeat timer expires (if multiple destinations are idle). It is an implementation decision on how to choose which of the candidate idle destinations to heartbeat to (if more than one destination is idle).
Note: When tuning the heartbeat interval, there is a side effect that SHOULD be taken into account. When this value is increased, i.e., the HEARTBEAT takes longer, the detection of lost ABORT messages takes longer as well. If a peer endpoint ABORTs the association for any reason and the ABORT chunk is lost, the local endpoint will only discover the lost ABORT by sending a DATA chunk or HEARTBEAT chunk (thus causing the peer to send another ABORT). This is to be considered when tuning the HEARTBEAT timer. If the HEARTBEAT is disabled, only sending DATA to the association will discover a lost ABORT from the peer.

8.4. Handle "Out of the Blue" Packets

An SCTP packet is called an "out of the blue" (OOTB) packet if it is correctly formed (i.e., passed the receiver’s CRC32c check; see Section 6.8), but the receiver is not able to identify the association to which this packet belongs.

The receiver of an OOTB packet MUST do the following:

1) If the OOTB packet is to or from a non-unicast address, a receiver SHOULD silently discard the packet. Otherwise,
2) If the OOTB packet contains an ABORT chunk, the receiver MUST silently discard the OOTB packet and take no further action. Otherwise,
3) If the packet contains an INIT chunk with a Verification Tag set to ‘0’, process it as described in Section 5.1. If, for whatever reason, the INIT cannot be processed normally and an ABORT has to be sent in response, the Verification Tag of the packet containing the ABORT chunk MUST be the Initiate Tag of the received INIT chunk, and the T bit of the ABORT chunk has to be set to 0, indicating that the Verification Tag is NOT reflected.
4) If the packet contains a COOKIE ECHO in the first chunk, process it as described in Section 5.1. Otherwise,
5) If the packet contains a SHUTDOWN ACK chunk, the receiver SHOULD respond to the sender of the OOTB packet with a SHUTDOWN COMPLETE. When sending the SHUTDOWN COMPLETE, the receiver of the OOTB packet MUST fill in the Verification Tag field of the outbound packet with the Verification Tag received in the SHUTDOWN ACK and set the T bit in the Chunk Flags to indicate that the Verification Tag is reflected. Otherwise,
6) If the packet contains a SHUTDOWN COMPLETE chunk, the receiver SHOULD silently discard the packet and take no further action. Otherwise,
7) If the packet contains a "Stale Cookie" ERROR or a COOKIE ACK, the SCTP packet SHOULD be silently discarded. Otherwise,
8) The receiver SHOULD respond to the sender of the OOTB packet with an ABORT. When sending the ABORT, the receiver of the OOTB
The endpoint MUST fill in the Verification Tag field of the outbound packet with the value found in the Verification Tag field of the OOTB packet and set the T bit in the Chunk Flags to indicate that the Verification Tag is reflected. After sending this ABORT, the receiver of the OOTB packet MUST discard the OOTB packet and MUST NOT take any further action.

8.5. Verification Tag

The Verification Tag rules defined in this section apply when sending or receiving SCTP packets that do not contain an INIT, SHUTDOWN COMPLETE, COOKIE ECHO (see Section 5.1), ABORT, or SHUTDOWN ACK chunk. The rules for sending and receiving SCTP packets containing one of these chunk types are discussed separately in Section 8.5.1.

When sending an SCTP packet, the endpoint MUST fill in the Verification Tag field of the outbound packet with the tag value in the Initiate Tag parameter of the INIT or INIT ACK received from its peer.

When receiving an SCTP packet, the endpoint MUST ensure that the value in the Verification Tag field of the received SCTP packet matches its own tag. If the received Verification Tag value does not match the receiver’s own tag value, the receiver MUST silently discard the packet and MUST NOT process it any further except for those cases listed in Section 8.5.1 below.

8.5.1. Exceptions in Verification Tag Rules

A) Rules for packet carrying INIT:

* The sender MUST set the Verification Tag of the packet to 0.

* When an endpoint receives an SCTP packet with the Verification Tag set to 0, it SHOULD verify that the packet contains only an INIT chunk. Otherwise, the receiver MUST silently discard the packet.

B) Rules for packet carrying ABORT:

* The endpoint MUST always fill in the Verification Tag field of the outbound packet with the destination endpoint’s tag value, if it is known.

* If the ABORT is sent in response to an OOTB packet, the endpoint MUST follow the procedure described in Section 8.4.
The receiver of an ABORT MUST accept the packet if the Verification Tag field of the packet matches its own tag and the T bit is not set OR if it is set to its peer’s tag and the T bit is set in the Chunk Flags. Otherwise, the receiver MUST silently discard the packet and take no further action.

C) Rules for packet carrying SHUTDOWN COMPLETE:

* When sending a SHUTDOWN COMPLETE, if the receiver of the SHUTDOWN ACK has a TCB, then the destination endpoint’s tag MUST be used, and the T bit MUST NOT be set. Only where no TCB exists SHOULD the sender use the Verification Tag from the SHUTDOWN ACK, and MUST set the T bit.

* The receiver of a SHUTDOWN COMPLETE accepts the packet if the Verification Tag field of the packet matches its own tag and the T bit is not set OR if it is set to its peer’s tag and the T bit is set in the Chunk Flags. Otherwise, the receiver MUST silently discard the packet and take no further action. An endpoint MUST ignore the SHUTDOWN COMPLETE if it is not in the SHUTDOWN-ACK-SENT state.

D) Rules for packet carrying a COOKIE ECHO:

* When sending a COOKIE ECHO, the endpoint MUST use the value of the Initiate Tag received in the INIT ACK.

* The receiver of a COOKIE ECHO follows the procedures in Section 5.

E) Rules for packet carrying a SHUTDOWN ACK:

* If the receiver is in COOKIE-ECHOED or COOKIE-WAIT state the procedures in Section 8.4 SHOULD be followed; in other words, it is treated as an Out Of The Blue packet.

9. Termination of Association

An endpoint SHOULD terminate its association when it exits from service. An association can be terminated by either abort or shutdown. An abort of an association is abortive by definition in that any data pending on either end of the association is discarded and not delivered to the peer. A shutdown of an association is considered a graceful close where all data in queue by either endpoint is delivered to the respective peers. However, in the case of a shutdown, SCTP does not support a half-open state (like TCP) wherein one side might continue sending data while the other end is closed. When either endpoint performs a shutdown, the association on
9.1. Abort of an Association

When an endpoint decides to abort an existing association, it MUST send an ABORT chunk to its peer endpoint. The sender MUST fill in the peer’s Verification Tag in the outbound packet and MUST NOT bundle any DATA chunk with the ABORT. If the association is aborted on request of the upper layer, a User-Initiated Abort error cause (see Section 3.3.10.12) SHOULD be present in the ABORT chunk.

An endpoint MUST NOT respond to any received packet that contains an ABORT chunk (also see Section 8.4).

An endpoint receiving an ABORT MUST apply the special Verification Tag check rules described in Section 8.5.1.

After checking the Verification Tag, the receiving endpoint MUST remove the association from its record and SHOULD report the termination to its upper layer. If a User-Initiated Abort error cause is present in the ABORT chunk, the Upper Layer Abort Reason SHOULD be made available to the upper layer.

9.2. Shutdown of an Association

Using the SHUTDOWN primitive (see Section 10.1), the upper layer of an endpoint in an association can gracefully close the association. This will allow all outstanding DATA chunks from the peer of the shutdown initiator to be delivered before the association terminates.

Upon receipt of the SHUTDOWN primitive from its upper layer, the endpoint enters the SHUTDOWN-PENDING state and remains there until all outstanding data has been acknowledged by its peer. The endpoint accepts no new data from its upper layer, but retransmits data to the far end if necessary to fill gaps.

Once all its outstanding data has been acknowledged, the endpoint sends a SHUTDOWN chunk to its peer including in the Cumulative TSN Ack field the last sequential TSN it has received from the peer. It SHOULD then start the T2-shutdown timer and enter the SHUTDOWN-SENT state. If the timer expires, the endpoint MUST resend the SHUTDOWN with the updated last sequential TSN received from its peer.

The rules in Section 6.3 MUST be followed to determine the proper timer value for T2-shutdown. To indicate any gaps in TSN, the endpoint MAY also bundle a SACK with the SHUTDOWN chunk in the same SCTP packet.
An endpoint SHOULD limit the number of retransmissions of the SHUTDOWN chunk to the protocol parameter ‘Association.Max.Retrs’.

If this threshold is exceeded, the endpoint SHOULD destroy the TCB and report the peer endpoint unreachable to the upper layer (and thus the association enters the CLOSED state). The reception of any packet from its peer (i.e., as the peer sends all of its queued DATA chunks) SHOULD clear the endpoint’s retransmission count and restart the T2-shutdown timer, giving its peer ample opportunity to transmit all of its queued DATA chunks that have not yet been sent.

Upon reception of the SHUTDOWN, the peer endpoint does the following:

- enter the SHUTDOWN-RECEIVED state,
- stop accepting new data from its SCTP user, and
- verify, by checking the Cumulative TSN Ack field of the chunk, that all its outstanding DATA chunks have been received by the SHUTDOWN sender.

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.

If there are still outstanding DATA chunks left, the SHUTDOWN receiver MUST continue to follow normal data transmission procedures defined in Section 6, until all outstanding DATA chunks are acknowledged; however, the SHUTDOWN receiver MUST NOT accept new data from its SCTP user.

While in the SHUTDOWN-SENT state, the SHUTDOWN sender MUST immediately respond to each received packet containing one or more DATA chunks with a SHUTDOWN chunk and restart the T2-shutdown timer. If a SHUTDOWN chunk by itself cannot acknowledge all of the received DATA chunks (i.e., there are TSNs that can be acknowledged that are larger than the cumulative TSN, and thus gaps exist in the TSN sequence), or if duplicate TSNs have been received, then a SACK chunk MUST also be sent.

The sender of the SHUTDOWN MAY also start an overall guard timer ‘T5-shutdown-guard’ to bound the overall time for the shutdown sequence. At the expiration of this timer, the sender SHOULD abort the association by sending an ABORT chunk. If the ‘T5-shutdown-guard’ timer is used, it SHOULD be set to the RECOMMENDED value of 5 times ‘RTO.Max’.

If the receiver of the SHUTDOWN has no more outstanding DATA chunks, the SHUTDOWN receiver MUST send a SHUTDOWN ACK and start a T2-
shutdown timer of its own, entering the SHUTDOWN-ACK-SENT state. If
the timer expires, the endpoint MUST resend the SHUTDOWN ACK.

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).

Upon the receipt of the SHUTDOWN ACK, the SHUTDOWN sender MUST stop the
T2-shutdown timer, send a SHUTDOWN COMPLETE chunk to its peer,
and remove all record of the association.

Upon reception of the SHUTDOWN COMPLETE chunk, the endpoint will verify that it is in the SHUTDOWN-ACK-SENT state; if it is not, the
chunk SHOULD be discarded. If the endpoint is in the SHUTDOWN-ACK-
SENT state, the endpoint SHOULD stop the T2-shutdown timer and remove all knowledge of the association (and thus the association enters the
CLOSED state).

An endpoint SHOULD ensure that all its outstanding DATA chunks have been acknowledged before initiating the shutdown procedure.

An endpoint SHOULD reject any new data request from its upper layer if it is in the SHUTDOWN-PENDING, SHUTDOWN-SENT, SHUTDOWN-RECEIVED, or SHUTDOWN-ACK-SENT state.

If an endpoint is in the SHUTDOWN-ACK-SENT state and receives an INIT chunk (e.g., if the SHUTDOWN COMPLETE was lost) with source and
destination transport addresses (either in the IP addresses or in the INIT chunk) that belong to this association, it SHOULD discard the INIT chunk and retransmit the SHUTDOWN ACK chunk.

Note: Receipt of an INIT with the same source and destination IP addresses as used in transport addresses assigned to an endpoint but with a different port number indicates the initialization of a separate association.

The sender of the INIT or COOKIE ECHO SHOULD respond to the receipt of a SHUTDOWN ACK with a stand-alone SHUTDOWN COMPLETE in an SCTP packet with the Verification Tag field of its common header set to the same tag that was received in the SHUTDOWN ACK packet. This is considered an Out of the Blue packet as defined in Section 8.4. The
sender of the INIT lets T1-init continue running and remains in the
COOKIE-WAIT or COOKIE-ECHOED state. Normal T1-init timer expiration will cause the INIT or COOKIE chunk to be retransmitted and thus start a new association.
If a SHUTDOWN is received in the COOKIE-WAIT or COOKIE ECHOED state, the SHUTDOWN chunk SHOULD be silently discarded.

If an endpoint is in the SHUTDOWN-SENT state and receives a SHUTDOWN chunk from its peer, the endpoint SHOULD respond immediately with a SHUTDOWN ACK to its peer, and move into the SHUTDOWN-ACK-SENT state restarting its T2-shutdown timer.

If an endpoint is in the SHUTDOWN-ACK-SENT state and receives a SHUTDOWN ACK, it MUST stop the T2-shutdown timer, send a SHUTDOWN COMPLETE chunk to its peer, and remove all record of the association.

10. Interface with Upper Layer

The Upper Layer Protocols (ULPs) request services by passing primitives to SCTP and receive notifications from SCTP for various events.

The primitives and notifications described in this section can be used as a guideline for implementing SCTP. The following functional description of ULP interface primitives is shown for illustrative purposes. Different SCTP implementations can have different ULP interfaces. However, all SCTPs are expected to provide a certain minimum set of services to guarantee that all SCTP implementations can support the same protocol hierarchy.

10.1. ULP-to-SCTP

The following sections functionally characterize a ULP/SCTP interface. The notation used is similar to most procedure or function calls in high-level languages.

The ULP primitives described below specify the basic functions that SCTP performs to support inter-process communication. Individual implementations define their own exact format, and provide combinations or subsets of the basic functions in single calls.

A) Initialize

Format: INITIALIZE ([local port],[local eligible address list])
-> local SCTP instance name

This primitive allows SCTP to initialize its internal data structures and allocate necessary resources for setting up its operation environment. Once SCTP is initialized, ULP can communicate directly with other endpoints without re-invoking this primitive.
SCTP will return a local SCTP instance name to the ULP.
Mandatory attributes:

* None.

Optional attributes:
The following types of attributes can be passed along with the primitive:

* local port - SCTP port number, if ULP wants it to be specified.
* local eligible address list - an address list that the local SCTP endpoint binds. By default, if an address list is not included, all IP addresses assigned to the host are used by the local endpoint.

IMPLEMENTATION NOTE: If this optional attribute is supported by an implementation, it will be the responsibility of the implementation to enforce that the IP source address field of any SCTP packets sent out by this endpoint contains one of the IP addresses indicated in the local eligible address list.

B) Associate

Format: ASSOCIATE(local SCTP instance name, destination transport addr, outbound stream count) -> association id [,destination transport addr list] [,outbound stream count]

This primitive allows the upper layer to initiate an association to a specific peer endpoint. The peer endpoint is specified by one of the transport addresses that defines the endpoint (see Section 1.3). If the local SCTP instance has not been initialized, the ASSOCIATE is considered an error.

An association id, which is a local handle to the SCTP association, will be returned on successful establishment of the association. If SCTP is not able to open an SCTP association with the peer endpoint, an error is returned.

Other association parameters can be returned, including the complete destination transport addresses of the peer as well as the outbound stream count of the local endpoint. One of the transport addresses from the returned destination addresses will be selected by the local endpoint as default primary path for sending SCTP packets to this peer. The returned "destination transport addr list" can be used by the ULP to change the default primary path or to force sending a packet to a specific transport address.

IMPLEMENTATION NOTE: If ASSOCIATE primitive is implemented as a blocking function call, the ASSOCIATE primitive can return
association parameters in addition to the association id upon successful establishment. If ASSOCIATE primitive is implemented as a non-blocking call, only the association id is returned and association parameters are passed using the COMMUNICATION UP notification.

Mandatory attributes:

* local SCTP instance name - obtained from the INITIALIZE operation.
* destination transport addr - specified as one of the transport addresses of the peer endpoint with which the association is to be established.
* outbound stream count - the number of outbound streams the ULP would like to open towards this peer endpoint.

Optional attributes:

* None.

C) Shutdown

Format: SHUTDOWN(association id) -> result

Gracefully closes an association. Any locally queued user data will be delivered to the peer. The association will be terminated only after the peer acknowledges all the SCTP packets sent. A success code will be returned on successful termination of the association. If attempting to terminate the association results in a failure, an error code is returned.

Mandatory attributes:

* association id - local handle to the SCTP association.

Optional attributes:

* None.

D) Abort

Format: ABORT(association id [, Upper Layer Abort Reason]) -> result

Ungracefully closes an association. Any locally queued user data will be discarded, and an ABORT chunk is sent to the peer. A success code will be returned on successful abort of the association. If attempting to abort the association results in a failure, an error code is returned.

Mandatory attributes:

* association id - local handle to the SCTP association.
Optional attributes:

* Upper Layer Abort Reason - reason of the abort to be passed to the peer.
  * None.

Send

Format: SEND(association id, buffer address, byte count
[,payload protocol-id] [,sack-immediately flag] ) ->

result

This is the main method to send user data via SCTP.

Mandatory attributes:

* association id - local handle to the SCTP association.
* buffer address - the location where the user message to be transmitted is stored.
* byte count - the size of the user data in number of bytes.

Optional attributes:

* context - an optional 32-bit integer that will be carried in the sending failure notification to the ULP if the transportation of this user message fails.
* stream id - to indicate which stream to send the data on. If not specified, stream 0 will be used.
* life time - specifies the life time of the user data. The user data will not be sent by SCTP after the life time expires. This parameter can be used to avoid efforts to transmit stale user messages. SCTP notifies the ULP if the data cannot be initiated to transport (i.e., sent to the destination via SCTP’s send primitive) within the life time variable. However, the user data will be transmitted if SCTP has attempted to transmit a chunk before the life time expired.

IMPLEMENTATION NOTE: In order to better support the data life time option, the transmitter can hold back the assigning of the TSN number to an outbound DATA chunk to the last moment. And, for implementation simplicity, once a TSN number has been assigned the sender considers the send of this DATA chunk as committed, overriding any life time option attached to the DATA chunk.

* destination transport address - specified as one of the destination transport addresses of the peer endpoint to which this packet is sent. Whenever possible, SCTP uses this
destination transport address for sending the packets, instead
of the current primary path.
* unordered flag - this flag, if present, indicates that the
user would like the data delivered in an unordered fashion to
the peer (i.e., the U flag is set to 1 on all DATA chunks
carrying this message).
* no-bundle flag - instructs SCTP not to bundle this user data
with other outbound DATA chunks. When faced with network
congestion, SCTP might still bundle the data, even when this
flag is present.
* payload protocol-id - a 32-bit unsigned integer that is to be
passed to the peer indicating the type of payload protocol
data being transmitted. This value is passed as opaque data
by SCTP.
* sack-immediately flag - set the I bit on the last DATA chunk
used for the user message to be transmitted.
F) Set Primary

Format: SETPRIMARY(association id, destination transport address,
[source transport address]) -> result

Instructs the local SCTP to use the specified destination
transport address as the primary path for sending packets.
The result of attempting this operation is returned. If the
specified destination transport address is not present in the
"destination transport address list" returned earlier in an
associate command or communication up notification, an error is
returned.
Mandatory attributes:

* association id - local handle to the SCTP association.
* destination transport address - specified as one of the
transport addresses of the peer endpoint, which is used as the
primary address for sending packets. This overrides the
current primary address information maintained by the local
SCTP endpoint.

Optional attributes:

* source transport address - optionally, some implementations
can allow you to set the default source address placed in all
outgoing IP datagrams.
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 primitive reads the first user message in the SCTP in-queue into the buffer specified by ULP, if there is one available. The size of the message read, in bytes, will be returned. It might, depending on the specific implementation, also return other information such as the sender’s address, the stream id on which it is received, whether there are more messages available for retrieval, etc. For ordered messages, their Stream Sequence Number might also be returned.

Depending upon the implementation, if this primitive is invoked when no message is available the implementation returns an indication of this condition or blocks the invoking process until data does become available.

Mandatory attributes:

* association id - local handle to the SCTP association
* buffer address - the memory location indicated by the ULP to store the received message.
* buffer size - the maximum size of data to be received, in bytes.

Optional attributes:

* stream id - to indicate which stream to receive the data on.
* stream sequence number - the Stream Sequence Number assigned by the sending SCTP peer.
* 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 accompanies this primitive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.
* payload protocol-id - a 32-bit unsigned integer that is received from the peer indicating the type of payload protocol of the received data. This value is passed as opaque data by SCTP.

H) Status

Format: STATUS(association id) -> status data

This primitive returns a data block containing the following information:

association connection state,
destination transport address list,
destination transport address reachability states,
current receiver window size,
current congestion window sizes,
number of unacknowledged DATA chunks,
number of DATA chunks pending receipt,
primary path,
most recent SRTT on primary path,
RTO on primary path,
SRTT and RTO on other destination addresses, etc.

Mandatory attributes:

* association id - local handle to the SCTP association.

Optional attributes:

* None.

I) Change Heartbeat

Format: CHANGE HEARTBEAT(association id, destination transport address, new state [,interval]) -> result

Instructs the local endpoint to enable or disable heartbeat on
the specified destination transport address.
The result of attempting this operation is returned.
Note: Even when enabled, heartbeat will not take place if the
destination transport address is not idle.
Mandatory attributes:

* association id - local handle to the SCTP association.
* destination transport address - specified as one of the
  transport addresses of the peer endpoint.
* new state - the new state of heartbeat for this destination
  transport address (either enabled or disabled).

Optional attributes:

* interval - if present, indicates the frequency of the
  heartbeat if this is to enable heartbeat on a destination
  transport address. This value is added to the RTO of the
destination transport address. This value, if present,
affects all destinations.

J) Request HeartBeat

Format: REQUESTHEARTBEAT(association id, destination transport address) -> result

Instructs the local endpoint to perform a HeartBeat on the
specified destination transport address of the given association.
The returned result indicates whether the transmission of the
HEARTBEAT chunk to the destination address is successful.
Mandatory attributes:
* association id - local handle to the SCTP association.
* destination transport address - the transport address of the
  association on which a heartbeat is issued.

K) Get SRTT Report

Format: GETSRTTREPORT(association id, destination transport
   address) -> srtt result

Instructs the local SCTP to report the current SRTT measurement
on the specified destination transport address of the given
association. The returned result can be an integer containing
the most recent SRTT in milliseconds.

Mandatory attributes:

* association id - local handle to the SCTP association.
* destination transport address - the transport address of the
  association on which the SRTT measurement is to be reported.

L) Set Failure Threshold

Format: SETFAILURETHRESHOLD(association id, destination transport
   address, failure threshold) -> result

This primitive allows the local SCTP to customize the
reachability failure detection threshold ‘Path.Max.Retrans’ for
the specified destination address.

Mandatory attributes:

* association id - local handle to the SCTP association.
* destination transport address - the transport address of the
  association on which the failure detection threshold is to be
  set.
* failure threshold - the new value of ‘Path.Max.Retrans’ for
  the destination address.

M) Set Protocol Parameters

Format: SETPROTOCOLPARAMETERS(association id, [destination
   transport address,] protocol parameter list) -> result

This primitive allows the local SCTP to customize the protocol
parameters.

Mandatory attributes:

* association id - local handle to the SCTP association.
* 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.
Optional attributes:

* destination transport address - some of the protocol parameters might be set on a per destination transport address basis.

N) Receive Unsent Message

Format: RECEIVE_UNSENT(data retrieval id, buffer address, buffer size [,stream id] [, stream sequence number] [,partial flag] [,payload protocol-id])

* data retrieval id - the identification passed to the ULP in the failure notification.
* buffer address - the memory location indicated by the ULP to store the received message.
* buffer size - the maximum size of data to be received, in bytes.

Optional attributes:

* stream id - this is a return value that is set to indicate which stream the data was sent to.
* stream sequence number - this value is returned indicating the Stream Sequence Number that was associated with the message.
* 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 accompanies this primitive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.
* payload protocol-id - The 32 bit unsigned integer that was sent to be sent to the peer indicating the type of payload protocol of the received data.

O) Receive Unacknowledged Message

Format: RECEIVE_UNACKED(data retrieval id, buffer address, buffer size, [,stream id] [,stream sequence number] [,partial flag] [,payload protocol-id])

* data retrieval id - the identification passed to the ULP in the failure notification.
* buffer address - the memory location indicated by the ULP to store the received message.
* buffer size - the maximum size of data to be received, in bytes.

Optional attributes:
* stream id - this is a return value that is set to indicate which stream the data was sent to.
* stream sequence number - this value is returned indicating the Stream Sequence Number that was associated with the message.
* 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 accompanies this primitive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.
* payload protocol-id - the 32-bit unsigned integer that was sent to the peer indicating the type of payload protocol of the received data.

P) Destroy SCTP Instance

Format: DESTROY(local SCTP instance name)

* local SCTP instance name - this is the value that was passed to the application in the initialize primitive and it indicates which SCTP instance is to be destroyed.

10.2. SCTP-to-ULP

It is assumed that the operating system or application environment provides a means for the SCTP to asynchronously signal the ULP process. When SCTP does signal a ULP process, certain information is passed to the ULP.

IMPLEMENTATION NOTE: In some cases, this might be done through a separate socket or error channel.

A) DATA ARRIVE notification
SCTP invokes this notification on the ULP when a user message is successfully received and ready for retrieval. The following might optionally be passed with the notification:

* association id - local handle to the SCTP association.
* stream id - to indicate which stream the data is received on.

B) SEND FAILURE notification
If a message cannot be delivered, SCTP invokes this notification on the ULP. The following might optionally be passed with the notification:

* association id - local handle to the SCTP association.
* data retrieval id - an identification used to retrieve unsent and unacknowledged data.
* cause code - indicating the reason of the failure, e.g., size too large, message life time expiration, etc.
C) NETWORK STATUS CHANGE notification
When a destination transport address is marked inactive (e.g., when SCTP detects a failure) or marked active (e.g., when SCTP detects a recovery), SCTP invokes this notification on the ULP. The following is passed with the notification:

* association id - local handle to the SCTP association.
* destination transport address - this indicates the destination transport address of the peer endpoint affected by the change.
* new-status - this indicates the new status.

D) COMMUNICATION UP notification
This notification is used when SCTP becomes ready to send or receive user messages, or when a lost communication to an endpoint is restored.
IMPLEMENTATION NOTE: If the ASSOCIATE primitive is implemented as a blocking function call, the association parameters are returned as a result of the ASSOCIATE primitive itself. In that case, COMMUNICATION UP notification is optional at the association initiator’s side. The following is passed with the notification:

* association id - local handle to the SCTP association.
* status - This indicates what type of event has occurred.
* destination transport address list - the complete set of transport addresses of the peer.
* outbound stream count - the maximum number of streams allowed to be used in this association by the ULP.
* inbound stream count - the number of streams the peer endpoint has requested with this association (this might not be the same number as ‘outbound stream count’).

E) COMMUNICATION LOST notification
When SCTP loses communication to an endpoint completely (e.g., via Heartbeats) or detects that the endpoint has performed an abort operation, it invokes this notification on the ULP. The following is passed with the notification:

* association id - local handle to the SCTP association.
* status - this indicates what type of event has occurred; the status might indicate that a failure OR a normal termination event occurred in response to a shutdown or abort request.

The following might be passed with the notification:

* data retrieval id - an identification used to retrieve unsent and unacknowledged data.
* last-acked - the TSN last acked by that peer endpoint.
* last-sent - the TSN last sent to that peer endpoint.
* Upper Layer Abort Reason - the abort reason specified in case of a user-initiated abort.

F) COMMUNICATION ERROR notification
When SCTP receives an ERROR chunk from its peer and decides to notify its ULP, it can invoke this notification on the ULP.
The following can be passed with the notification:

* association id - local handle to the SCTP association.
* error info - this indicates the type of error and optionally some additional information received through the ERROR chunk.

G) RESTART notification
When SCTP detects that the peer has restarted, it might send this notification to its ULP.
The following can be passed with the notification:

* association id - local handle to the SCTP association.

H) SHUTDOWN COMPLETE notification
When SCTP completes the shutdown procedures (Section 9.2), this notification is passed to the upper layer.
The following can be passed with the notification:

* association id - local handle to the SCTP association.

11. Security Considerations

11.1. Security Objectives

As a common transport protocol designed to reliably carry time-sensitive user messages, such as billing or signaling messages for telephony services, between two networked endpoints, SCTP has the following security objectives.

- availability of reliable and timely data transport services
- integrity of the user-to-user information carried by SCTP

11.2. SCTP Responses to Potential Threats

SCTP could potentially be used in a wide variety of risk situations. It is important for operators of systems running SCTP to analyze their particular situations and decide on the appropriate countermeasures.

Operators of systems running SCTP might consult [RFC2196] for guidance in securing their site.
11.2.1.  Countering Insider Attacks

The principles of [RFC2196] might be applied to minimize the risk of theft of information or sabotage by insiders. Such procedures include publication of security policies, control of access at the physical, software, and network levels, and separation of services.

11.2.2.  Protecting against Data Corruption in the Network

Where the risk of undetected errors in datagrams delivered by the lower-layer transport services is considered to be too great, additional integrity protection is required. If this additional protection were provided in the application layer, the SCTP header would remain vulnerable to deliberate integrity attacks. While the existing SCTP mechanisms for detection of packet replays are considered sufficient for normal operation, stronger protections are needed to protect SCTP when the operating environment contains significant risk of deliberate attacks from a sophisticated adversary.

The SCTP Authentication extension SCTP-AUTH [RFC4895] MAY be used when the threat environment requires stronger integrity protections, but does not require confidentiality.

11.2.3.  Protecting Confidentiality

In most cases, the risk of breach of confidentiality applies to the signaling data payload, not to the SCTP or lower-layer protocol overheads. If that is true, encryption of the SCTP user data only might be considered. As with the supplementary checksum service, user data encryption MAY be performed by the SCTP user application. Alternately, the user application MAY use an implementation-specific API to request that the IP Encapsulating Security Payload (ESP) [RFC4303] be used to provide confidentiality and integrity.

Particularly for mobile users, the requirement for confidentiality might include the masking of IP addresses and ports. In this case, ESP SHOULD be used instead of application-level confidentiality. If ESP is used to protect confidentiality of SCTP traffic, an ESP cryptographic transform that includes cryptographic integrity protection MUST be used, because if there is a confidentiality threat there will also be a strong integrity threat.

Whenever ESP is in use, application-level encryption is not generally required.
Regardless of where confidentiality is provided, the Internet Key Exchange Protocol version 2 (IKEv2) [RFC7296] SHOULD be used for key management.

Operators might consult [RFC4301] for more information on the security services available at and immediately above the Internet Protocol layer.

11.2.4. Protecting against Blind Denial-of-Service Attacks

A blind attack is one where the attacker is unable to intercept or otherwise see the content of data flows passing to and from the target SCTP node. Blind denial-of-service attacks can take the form of flooding, masquerade, or improper monopolization of services.

11.2.4.1. Flooding

The objective of flooding is to cause loss of service and incorrect behavior at target systems through resource exhaustion, interference with legitimate transactions, and exploitation of buffer-related software bugs. Flooding can be directed either at the SCTP node or at resources in the intervening IP Access Links or the Internet. Where the latter entities are the target, flooding will manifest itself as loss of network services, including potentially the breach of any firewalls in place.

In general, protection against flooding begins at the equipment design level, where it includes measures such as:

- avoiding commitment of limited resources before determining that the request for service is legitimate.
- giving priority to completion of processing in progress over the acceptance of new work.
- identification and removal of duplicate or stale queued requests for service.
- not responding to unexpected packets sent to non-unicast addresses.

Network equipment is expected to be capable of generating an alarm and log if a suspicious increase in traffic occurs. The log provides information such as the identity of the incoming link and source address(es) used, which will help the network or SCTP system operator to take protective measures. Procedures are expected to be in place for the operator to act on such alarms if a clear pattern of abuse emerges.
The design of SCTP is resistant to flooding attacks, particularly in its use of a four-way startup handshake, its use of a cookie to defer commitment of resources at the responding SCTP node until the handshake is completed, and its use of a Verification Tag to prevent insertion of extraneous packets into the flow of an established association.

The IP Authentication Header and Encapsulating Security Payload might be useful in reducing the risk of certain kinds of denial-of-service attacks.

Support for 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 "Unresolvable Address" error cause.

11.2.4.2. Blind Masquerade

Masquerade can be used to deny service in several ways:

- by tying up resources at the target SCTP node to which the impersonated node has limited access. For example, the target node can by policy permit a maximum of one SCTP association with the impersonated SCTP node. The masquerading attacker can attempt to establish an association purporting to come from the impersonated node so that the latter cannot do so when it requires it.

- by deliberately allowing the impersonation to be detected, thereby provoking counter-measures that cause the impersonated node to be locked out of the target SCTP node.

- by interfering with an established association by inserting extraneous content such as a SHUTDOWN request.

SCTP reduces the risk of blind masquerade attacks through IP spoofing by use of the four-way startup handshake. Because the initial exchange is memory-less, no lockout mechanism is triggered by blind masquerade attacks. In addition, the INIT ACK containing the State Cookie is transmitted back to the IP address from which it received the INIT. Thus, the attacker would not receive the INIT ACK containing the State Cookie. SCTP protects against insertion of extraneous packets into the flow of an established association by use of the Verification Tag.

Logging of received INIT requests and abnormalities such as unexpected INIT ACKs might be considered as a way to detect patterns of hostile activity. However, the potential usefulness of such
logging has to be weighed against the increased SCTP startup processing it implies, rendering the SCTP node more vulnerable to flooding attacks. Logging is pointless without the establishment of operating procedures to review and analyze the logs on a routine basis.

11.2.4.3. Improper Monopolization of Services

Attacks under this heading are performed openly and legitimately by the attacker. They are directed against fellow users of the target SCTP node or of the shared resources between the attacker and the target node. Possible attacks include the opening of a large number of associations between the attacker's node and the target, or transfer of large volumes of information within a legitimately established association.

Policy limits are expected to be placed on the number of associations per adjoining SCTP node. SCTP user applications are expected to be capable of detecting large volumes of illegitimate or "no-op" messages within a given association and either logging or terminating the association as a result, based on local policy.

11.3. SCTP Interactions with Firewalls

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, as 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.

11.4. Protection of Non-SCTP-Capable Hosts

To provide a non-SCTP-capable host with the same level of protection against attacks as for SCTP-capable ones, all SCTP stacks MUST implement the ICMP handling described in Appendix C.

When an SCTP stack receives a packet containing multiple control or DATA chunks and the processing of the packet requires the sending of multiple chunks in response, the sender of the response chunk(s) MUST NOT send more than one packet. If bundling is supported, multiple response chunks that fit into a single packet MAY be bundled together into one single response packet. If bundling is not supported, then the sender MUST NOT send more than one response chunk and MUST
discard all other responses. Note that this rule does NOT apply to a
SACK chunk, since a SACK chunk is, in itself, a response to DATA and
a SACK does not require a response of more DATA.

An SCTP implementation SHOULD abort the association if it receives a
SACK acknowledging a TSN that has not been sent.

An SCTP implementation that receives an INIT that would require a
large packet in response, due to the inclusion of multiple ERROR
parameters, MAY (at its discretion) elect to omit some or all of the
ERROR parameters to reduce the size of the INIT ACK. Due to a
combination of the size of the COOKIE parameter and the number of
addresses a receiver of an INIT indicates to a peer, it is always
possible that the INIT ACK will be larger than the original INIT. An
SCTP implementation SHOULD attempt to make the INIT ACK as small as
possible to reduce the possibility of byte amplification attacks.

12. Network Management Considerations

The MIB module for SCTP defined in [RFC3873] applies for the version
of the protocol specified in this document.

13. Recommended Transmission Control Block (TCB) Parameters

This section details a set of parameters that are expected to be
contained within the TCB for an implementation. This section is for
illustrative purposes and is not considered to be requirements on an
implementation or as an exhaustive list of all parameters inside an
SCTP TCB. Each implementation might need its own additional
parameters for optimization.

13.1. Parameters Necessary for the SCTP Instance

Associations: A list of current associations and mappings to the data
consumers for each association. This might be in the
form of a hash table or other implementation-dependent
structure. The data consumers might be process
identification information such as file descriptors,
named pipe pointer, or table pointers dependent on how
SCTP is implemented.

Secret Key: A secret key used by this endpoint to compute the MAC.
This SHOULD be a cryptographic quality random number
with a sufficient length. Discussion in [RFC4086] can
be helpful in selection of the key.

Address List: The list of IP addresses that this instance has bound.
This information is passed to one’s peer(s) in INIT and
INIT ACK chunks.
SCTP Port: The local SCTP port number to which the endpoint is bound.

13.2. Parameters Necessary per Association (i.e., the TCB)

Peer Verification Tag: Tag value to be sent in every packet and is received in the INIT or INIT ACK chunk.

My Verification Tag: Tag expected in every inbound packet and sent in the INIT or INIT ACK chunk.

State: A state variable indicating what state the association is in, i.e., COOKIE-WAIT, COOKIE-ECHOED, ESTABLISHED, SHUTDOWN-PENDING, SHUTDOWN-SENT, SHUTDOWN-RECEIVED, SHUTDOWN-ACK-SENT.
Note: No "CLOSED" state is illustrated since if an association is "CLOSED" its TCB SHOULD be removed.

Peer Transport Address List: A list of SCTP transport addresses to which the peer is bound. This information is derived from the INIT or INIT ACK and is used to associate an inbound packet with a given association. Normally, this information is hashed or keyed for quick lookup and access of the TCB.

Primary Path: This is the current primary destination transport address of the peer endpoint. It might also specify a source transport address on this endpoint.

Overall Error Count: The overall association error count.

Overall Error Threshold: The threshold for this association that if the Overall Error Count reaches will cause this association to be torn down.

Peer Rwnd: Current calculated value of the peer’s rwnd.

Next TSN: The next TSN number to be assigned to a new DATA chunk. This is sent in the INIT or INIT ACK chunk to the peer and incremented each time a DATA chunk is assigned a TSN (normally just prior to transmit or during fragmentation).

Last Rcvd TSN: This is the last TSN received in sequence. This value is set initially by taking the peer’s initial TSN, received in the INIT or INIT ACK chunk, and subtracting one from it.

Mapping Array: An array of bits or bytes indicating which out-of-order TSNs have been received (relative to the Last Rcvd TSN). If no gaps exist, i.e., no out-of-order packets have been received, this array will be set to all zero. This structure might be in the form of a circular buffer or bit array.

Ack State: This flag indicates if the next received packet is to be responded to with a SACK. This is initialized to 0. When a packet is received it is incremented. If this value reaches 2 or more, a SACK is sent and the value
is reset to 0. Note: This is used only when no DATA chunks are received out of order. When DATA chunks are out of order, SACKs are not delayed (see Section 6).

Inbound Streams: An array of structures to track the inbound streams, normally including the next sequence number expected and possibly the stream number.

Outbound Streams: An array of structures to track the outbound streams, normally including the next sequence number to be sent on the stream.

Reasm Queue: A reassembly queue.

Local Transport Address List: The list of local IP addresses bound in to this association.

Association PMTU: The smallest PMTU discovered for all of the peer’s transport addresses.

13.3. Per Transport Address Data

For each destination transport address in the peer’s address list derived from the INIT or INIT ACK chunk, a number of data elements need to be maintained including:

Error Count: The current error count for this destination.
Error Threshold: Current error threshold for this destination, i.e., what value marks the destination down if error count reaches this value.
cwnd: The current congestion window.
ssthresh: The current ssthresh value.
RTO: The current retransmission timeout value.
SRTT: The current smoothed round-trip time.
RTTVAR: The current RTT variation.
partial bytes acked: The tracking method for increase of cwnd when in congestion avoidance mode (see Section 7.2.2).
state: The current state of this destination, i.e., DOWN, UP, ALLOW-HB, NO-HEARTBEAT, etc.
PMTU: The current known path MTU.
Per Destination Timer: A timer used by each destination.
RTO-Pending: A flag used to track if one of the DATA chunks sent to this address is currently being used to compute an RTT. If this flag is 0, the next DATA chunk sent to this destination is expected to be used to compute an RTT and this flag is expected to be set. Every time the RTT calculation completes (i.e., the DATA chunk is SACK’d), clear this flag.
last-time: The time to which this destination was last sent. This can be to determine if a HEARTBEAT is needed.
13.4. General Parameters Needed

Out Queue: A queue of outbound DATA chunks.
In Queue: A queue of inbound DATA chunks.

14. IANA Considerations

SCTP defines three registries that IANA maintains:

- through definition of additional chunk types,
- through definition of additional parameter types, or
- through definition of additional cause codes within ERROR chunks.

SCTP requires that the IANA Port Numbers registry be opened for SCTP port registrations, Section 14.6 describes how. An IESG-appointed Expert Reviewer supports IANA in evaluating SCTP port allocation requests.

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 for 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.

c) A detailed definition and description of 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 Section 14.2.
14.2. IETF Chunk Flags Registration

The assignment of new chunk flags is done through an RFC Required action, as defined in [RFC8126]. Documentation for 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.

14.3. IETF-Defined Chunk Parameter Extension

The assignment of new chunk parameter type codes is done through an IETF Review action as defined in [RFC8126]. Documentation of the chunk parameter MUST contain the following information:

a) Name of the parameter type.

b) Detailed description of the structure of the parameter field. This structure MUST conform to the general Type-Length-Value format described in Section 3.2.1.

c) Detailed definition of each component of the parameter value.

d) Detailed description of the intended use of this parameter type, and an indication of whether and under what circumstances multiple instances of this parameter type can be found within the same chunk.

e) Each parameter type MUST be unique across all chunks.

14.4. IETF-Defined Additional Error Causes

Additional cause codes can be allocated in the range 11 to 65535 through a Specification Required action as defined in [RFC8126]. Provided documentation MUST include the following information:

a) Name of the error condition.
b) Detailed description of the conditions under which an SCTP endpoint issues an ERROR (or ABORT) with this cause code.

c) Expected action by the SCTP endpoint that receives an ERROR (or ABORT) chunk containing this cause code.

d) Detailed description of the structure and content of data fields that accompany this cause code.

The initial word (32 bits) of a cause code parameter MUST conform to the format shown in Section 3.3.10, i.e.:

- first 2 bytes contain the cause code value
- last 2 bytes contain the length of the cause parameter.

14.5. Payload Protocol Identifiers

Except for value 0, which is reserved by SCTP to indicate an unspecified payload protocol identifier in a DATA chunk, SCTP will not be responsible for standardizing or verifying any payload protocol identifiers; SCTP simply receives the identifier from the upper layer and carries it with the corresponding payload data.

The upper layer, i.e., the SCTP user, SHOULD standardize any specific protocol identifier with IANA if it is so desired. The use of any specific payload protocol identifier is out of the scope of SCTP.

14.6. Port Numbers Registry

SCTP services can use contact port numbers to provide service to unknown callers, as in TCP and UDP. IANA is therefore requested to open the existing "Service Name and Transport Protocol Port Number Registry" for SCTP using the following rules, which we intend to mesh well with existing port-number registration procedures. An IESG-appointed expert reviewer supports IANA in evaluating SCTP port allocation requests, according to the procedure defined in [RFC8126]. The details of this process are defined in [RFC6335].

This document registers the following ports. (These registrations are to be considered models to follow for future allocation requests.)
The discard service, which accepts SCTP connections on port 9, discards all incoming application data and sends no data in response. Thus, SCTP’s discard port is analogous to TCP’s discard port, and might be used to check the health of an SCTP stack.

File Transfer Protocol (FTP) data (20) and control ports (21).

The Secure Shell (SSH) remote login service, which allows secure shell logins to a host.

World Wide Web HTTP over SCTP.

Border Gateway Protocol over SCTP.

World Wide Web HTTP over TLS/SSL over SCTP.
15. Suggested SCTP Protocol Parameter Values

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

IMPLEMENTATION NOTE: The SCTP implementation can allow ULP to customize some of these protocol parameters (see Section 10).

Note: RTO.Min SHOULD be set as described above in this section.

16. Acknowledgements

An undertaking represented by this updated document is not a small feat and represents the summation of the initial authors of RFC 2960: Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson.

Add to that, the comments from everyone who contributed to the original RFC:


Then, add the authors of the SCTP implementor’s guide, I. Arias-Rodriguez, K. Poon, A. Caro, and M. Tuexen.
Then add to these the efforts of all the subsequent seven SCTP interoperability tests and those who commented on RFC 4460 as shown in its acknowledgements:


A special thanks to Mark Allman, who should actually be a co-author for his work on the max-burst, but managed to wiggle out due to a technicality. Also, we would like to acknowledge Lyndon Ong and Phil Conrad for their valuable input and many contributions.

And finally, you have this document, and those who have commented upon that including Alfred Hoenes and Ronnie Sellars.

My thanks cannot be adequately expressed to all of you who have participated in the coding, testing, and updating process of this document. All I can say is, Thank You!

Randall Stewart - Editor

17. References

17.1. Normative References


17.2. Informative References


Appendix A. Explicit Congestion Notification

ECN as specified in [RFC3168] (updated by [RFC8311]) describes an extension to IP that details a method for becoming aware of congestion outside of datagram loss. This is a feature that an implementation MAY choose to add to SCTP. This appendix details the minor differences implementers will need to be aware of if they choose to implement this feature. In general, [RFC3168] (updated by [RFC8311]) SHOULD be followed, with the following exceptions.

Negotiation:

[RFC3168] (updated by [RFC8311]) details the negotiation of ECN during the SYN and SYN-ACK stages of a TCP connection. The sender of the SYN sets 2 bits in the TCP flags, and the sender of the SYN-ACK sets only 1 bit. The reasoning behind this is to ensure that both sides are truly ECN capable. For SCTP, this is not necessary. To indicate that an endpoint is ECN capable, an endpoint SHOULD add to the INIT and or INIT ACK chunk the TLV reserved for ECN. This TLV contains no parameters, and thus has the following format:

```
0                   1                   2                   3
+-------------------+-------------------+-------------------+-------------------+
| Parameter Type = 32768 | Parameter Length = 4 |
+-------------------+-------------------+-------------------+-------------------+
```

ECN-Echo:

[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 that the CE bit has
arrived from the network. For SCTP, this same indication is made by including the ECNE chunk. This chunk contains one data element, i.e., the lowest TSN associated with the IP datagram marked with the CE bit, and looks as follows:

```
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----------------------------------------------+
| Chunk Type=12 | Flags=00000000 | Chunk Length = 8 |
+-----------------------------------------------+
| Lowest TSN Number                              |
+-----------------------------------------------+
```

Note: The ECNE is considered a Control chunk.

CWR:

[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 is termed the CWR bit. For SCTP, the same indication is made by including the CWR chunk. This chunk contains one data element, i.e., the TSN number that was sent in the ECNE chunk. This element represents the lowest TSN number in the datagram that was originally marked with the CE bit.

```
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----------------------------------------------+
| Chunk Type=13 | Flags=00000000 | Chunk Length = 8 |
+-----------------------------------------------+
| Lowest TSN Number                              |
+-----------------------------------------------+
```

Note: The CWR is considered a Control chunk.

Appendix B. CRC32c Checksum Calculation

We define a ‘reflected value’ as one that is the opposite of the normal bit order of the machine. The 32-bit CRC (Cyclic Redundancy Check) is calculated as described for CRC32c and uses the polynomial code 0x11EDC6F41 (Castagnoli93) or

\[
x^{32} + x^{28} + x^{27} + x^{26} + x^{25} + x^{24} + x^{22} + x^{20} + x^{19} + x^{18} +
\]

\[
x^{14} + x^{13} + x^{11} + x^{10} + x^{9} + x^{8} + x^{6} + x^{0}.
\]

The CRC is computed using a procedure similar to ETHERNET CRC [ITU.V42.1994], modified to reflect transport-level usage.
CRC computation uses polynomial division. A message bit-string M is transformed to a polynomial, M(X), and the CRC is calculated from M(X) using polynomial arithmetic.

When CRCs are used at the link layer, the polynomial is derived from on-the-wire bit ordering: the first bit ‘on the wire’ is the high-order coefficient. Since SCTP is a transport-level protocol, it cannot know the actual serial-media bit ordering. Moreover, different links in the path between SCTP endpoints can use different link-level bit orders.

A convention therefore is established for mapping SCTP transport messages to polynomials for purposes of CRC computation. The bit-ordering for mapping SCTP messages to polynomials is that bytes are taken most-significant first, but within each byte, bits are taken least-significant first. The first byte of the message provides the eight highest coefficients. Within each byte, the least-significant SCTP bit gives the most-significant polynomial coefficient within that byte, and the most-significant SCTP bit is the least-significant polynomial coefficient in that byte. (This bit ordering is sometimes called ‘mirrored’ or ‘reflected’ [WILLIAMS93].) CRC polynomials are to be transformed back into SCTP transport-level byte values, using a consistent mapping.

The SCTP transport-level CRC value can be calculated as follows:

- CRC input data are assigned to a byte stream, numbered from 0 to N-1.
- The transport-level byte stream is mapped to a polynomial value. An N-byte PDU with j bytes numbered 0 to N-1 is considered as coefficients of a polynomial M(x) of order 8N-1, with bit 0 of byte j being coefficient x^(8(N-j)-8), and bit 7 of byte j being coefficient x^(8(N-j)-1).
- The CRC remainder register is initialized with all 1s and the CRC is computed with an algorithm that simultaneously multiplies by x^32 and divides by the CRC polynomial.
- The polynomial is multiplied by x^32 and divided by G(x), the generator polynomial, producing a remainder R(x) of degree less than or equal to 31.
- The coefficients of R(x) are considered a 32-bit sequence.
- The bit sequence is complemented. The result is the CRC polynomial.
The CRC polynomial is mapped back into SCTP transport-level bytes. The coefficient of \( x^{31} \) gives the value of bit 7 of SCTP byte 0, and the coefficient of \( x^{24} \) gives the value of bit 0 of byte 0. The coefficient of \( x^{7} \) gives bit 7 of byte 3, and the coefficient of \( x^{0} \) gives bit 0 of byte 3. The resulting 4-byte transport-level sequence is the 32-bit SCTP checksum value.

IMPLEMENTATION NOTE: Standards documents, textbooks, and vendor literature on CRCs often follow an alternative formulation, in which the register used to hold the remainder of the long-division algorithm is initialized to zero rather than all-1s, and instead the first 32 bits of the message are complemented. The long-division algorithm used in our formulation is specified such that the initial multiplication by \( 2^{32} \) and the long-division are combined into one simultaneous operation. For such algorithms, and for messages longer than 64 bits, the two specifications are precisely equivalent. That equivalence is the intent of this document.

Implementors of SCTP are warned that both specifications are to be found in the literature, sometimes with no restriction on the long-division algorithm. The choice of formulation in this document is to permit non-SCTP usage, where the same CRC algorithm can be used to protect messages shorter than 64 bits.

There can be a computational advantage in validating the association against the Verification Tag, prior to performing a checksum, as invalid tags will result in the same action as a bad checksum in most cases. The exceptions for this technique would be INIT and some SHUTDOWN-COMPLETE exchanges, as well as a stale COOKIE ECHO. These special-case exchanges represent small packets and will minimize the effect of the checksum calculation.

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. While neither especially slow nor especially fast, as software table-lookup CRCs go, it has the advantage of working on both big-endian and little-endian CPUs, using the same (host-order) lookup tables, and using only the predefined ntohl() and htonl() operations. The code is somewhat modified from [WILLIAMS93], to ensure portability between big-endian and little-endian architectures. (Note that if the byte endian-ness of the target architecture is known to be little-endian, the final bit-reversal and byte-reversal steps can be folded into a single operation.)

```
<CODE BEGINS>
/****************************************************************
```
/* Note: The definitions for Ross Williams’s table generator */
/* would be TB_WIDTH=4, TB_POLY=0x1EDC6F41, TB_REVER=TRUE. */
/* For Mr. Williams’s direct calculation code, use the settings */
/* cm_width=32, cm_poly=0x1EDC6F41, cm_init=0xFFFFFFFF, */
/* cm_refin=TRUE, cm_refot=TRUE, cm_xorot=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,
  0x5D1D08BFUL, 0xAF768BBCUL, 0x2BC267848UL, 0x4E4DFB44UL,
  0x820DBEDEUL, 0xD2D60DDDUL, 0xC186FE29UL, 0x33ED7D2AUL,
  0xE72719C1UL, 0x154C9AC2UL, 0x061C6936UL, 0xF477EA35UL,
  0xAA64D611UL, 0x580F5512UL, 0x4B5FA6E6UL, 0xB93425E5UL,
  0x6DFE410EUL, 0x9F95C20DUL, 0x8CC531F9UL, 0x7EAEB2FAUL,
  0x30E349B1UL, 0xC288CA24UL, 0xD1D83946UL, 0x23B3BA45UL,
  0xF779DEAEUL, 0x05125DADUL, 0x1642AE59UL, 0xE4292D5AUL,
  0xBA3A117EUL, 0x48519627UL, 0x5B016189UL, 0xA96EA28AUL,
  0x7DA08661UL, 0x8FCB0562UL, 0x9C9BF696UL, 0x6EF07595UL,
  0x417B1DBCUL, 0xB310EBF2UL, 0xAA406D48UL, 0x525BEE48UL,
  0x86E18A3UL, 0x748A90A0UL, 0x67DAFA54UL, 0x95B17957UL,
  0x0CBA24573UL, 0x399C6702UL, 0x2A993584UL, 0x08F28B67UL,
  0x0C38D26CUL, 0xEF5351F6UL, 0xED03A927UL, 0x1F682198UL,
  0x5152DAD3UL, 0xA34E59D0UL, 0xB01EEAA4UL, 0x42752927UL,
  0x96BF4DCCUL, 0x64D4CECFUL, 0x77843D38UL, 0x85EFBE38UL,
  0xDBFC821CUL, 0x2997011FUL, 0x3AC7F2EBUL, 0xC8AC71E8UL,
  0x1C661503UL, 0xEE0D9600UL, 0xFD5D65F4UL, 0x0F36E6F7UL,
  0x61C69362UL, 0x93AD1061UL, 0x80FD395UL, 0x72966096UL,
  0xA65C047DUL, 0x543787E7UL, 0x4767748AUL, 0xB50CF789UL,
  0xEBF1CBADUL, 0x197448AEUL, 0xA024B5A5UL, 0xF84F3859UL,
  0x2C855CB2UL, 0xDEEEDEBF1UL, 0xCDBE2C45UL, 0x3F5D4AF6UL,
  0x719854D0UL, 0x83F3D70EUL, 0x90A324FAUL, 0x62C8A7F9UL,
  0xB602C312UL, 0x44694011UL, 0x5739B3E5UL, 0xA55230E6UL,
  0xEB410CC2UL, 0x092A8FC1UL, 0x1A77C35UL, 0x8E811FF36UL,
  0xC3DB9F8DUL, 0xCE8B19DEUL, 0x9DE0EB24UL, 0x2FB8B929UL,
  0x82F63B78UL, 0x709DB87BUL, 0x63CD4B8FUL, 0x91A6C88CUL,
#endif

/* 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 CRC32c table file <%s>.
", 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");
    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{
");
    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

#endif

if (fclose(tf) != 0)
    printf("Unable to close <%s>.
", OUTPUT_FILE);
else
    printf("\nThe CRC32c table has been written to <%s>.\n", OUTPUT_FILE);
    return (0);
}

/* 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 are "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, and then did an
    * end-for-end bit-reversal.
    * Note that a 32-bit bit-reversal is identical to four in-place
    * 8-bit 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;
    unsigned int 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);
}

Appendix C. ICMP Handling

Whenever an ICMP message is received by an SCTP endpoint, the
following procedures MUST be followed to ensure proper utilization of
the information being provided by layer 3.
ICMP1) An implementation MAY ignore all ICMPv4 messages where the type field is not set to "Destination Unreachable".

ICMP2) An implementation MAY ignore all ICMPv6 messages where the type field is not "Destination Unreachable", "Parameter Problem", or "Packet Too Big".

ICMP3) An implementation SHOULD ignore any ICMP messages where the code indicates "Port Unreachable".

ICMP4) An implementation MAY ignore all ICMPv6 messages of type "Parameter Problem" if the code is not "Unrecognized Next Header Type Encountered".

ICMP5) An implementation MUST use the payload of the ICMP message (v4 or v6) to locate the association that sent the message to which ICMP is responding. If the association cannot be found, an implementation SHOULD ignore the ICMP message.

ICMP6) An implementation MUST validate that the Verification Tag contained in the ICMP message matches the Verification Tag of the peer. If the Verification Tag is not 0 and does NOT match, discard the ICMP message. If it is 0 and the ICMP message contains enough bytes to verify that the chunk type is an INIT chunk and that the Initiate Tag matches the tag of the peer, continue with ICMP7. If the ICMP message is too short or the chunk type or the Initiate Tag does not match, silently discard the packet.

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.

ICMP8) If the ICMP code is an "Unrecognized Next Header Type Encountered" or a "Protocol Unreachable", an implementation MUST treat this message as an abort with the T bit set if it does not contain an INIT chunk. If it does contain an INIT chunk and the association is in the COOKIE-WAIT state, handle the ICMP message like an ABORT.

ICMP9) If the ICMP type is "Destination Unreachable", the implementation MAY move the destination to 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.
Note that these procedures differ from [RFC1122] and from its requirements for processing of port-unreachable messages and the requirements that an implementation MUST abort associations in response to a "protocol unreachable" message. Port-unreachable messages are not processed, since an implementation will send an ABORT, not a port unreachable. The stricter handling of the "protocol unreachable" message is due to security concerns for hosts that do NOT support SCTP.

Authors’ Addresses

Randall R. Stewart
Netflix, Inc.
2455 Heritage Green Ave
Davenport, FL 33837
United States

Email: randall@lakerest.net

Michael Tuexen
Muenster University of Applied Sciences
Stegerwaldstrasse 39
Steinfurt 48565
Germany

Email: tuexen@fh-muenster.de

Karen E. E. Nielsen
Kamstrup A/S
Industrivej 28
Skanderborg DK-8660
Denmark

Email: kee@kamstrup.com
Propagating Explicit Congestion Notification Across IP Tunnel Headers
Separated by a Shim
draft-ietf-tsvwg-rfc6040update-shim-09

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 January 9, 2020.

Copyright Notice

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

- Legacy implementations of tunnels without any ECN support already exist
- A network might be designed so that there is usually no bottleneck within the tunnel
- 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.

As a tunnel egress reassembles sets of outer fragments [I-D.ietf-intarea-tunnels] into packets, it SHOULD propagate CE markings on the basis that a congestion indication on a packet applies to all the octets in the packet. On average, a tunnel egress SHOULD approximately preserve the number of CE-marked and ECT(1)-marked octets arriving and leaving (counting the size of inner headers, but not encapsulating headers that are being stripped). This process proceeds irrespective of the addresses on the inner headers.

Even if only enough incoming CE-marked octets have arrived for part of the departing packet, the next departing packet SHOULD be immediately CE-marked. This ensures that CE-markings are propagated immediately, rather than held back waiting for more incoming CE-marked octets. Once there are no outstanding CE-marked octets, if only enough incoming ECT(1)-marked octets have arrived for part of the departing packet, the next departing packet SHOULD be immediately marked ECT(1).

For instance, an algorithm for marking departing packets could maintain a pair of counters, the first representing the balance of arriving CE-marked octets minus departing CE-marked octets and the second representing a similar balance of ECT(1)-marked octets. The algorithm:

- adds the size of every CE-marked or ECT(1)-marked packet that arrives to the appropriate counter;
o if the CE counter is positive, it CE-marks the next packet to depart and subtracts its size from the CE counter;

o if the CE counter is negative but the ECT(1) counter is positive, it marks the next packet to depart as ECT(1) and subtracts its size from the ECT((1) counter;

o (the previous two steps will often leave a negative remainder in the counters, which is deliberate);

o if neither counter is positive, it marks the next packet to depart as ECT(0);

o until all the fragments of a packet have arrived, it does not commit any updates to the counters so that, if reassembly fails and the partly reassembled packet has to be discarded, none of the discarded fragments will have updated any of the counters.

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.

A tunnel end-point that claims to support the present specification MUST NOT use an approach that results in a significantly different ECN-marking outcome to that defined by the "SHOULD" statements throughout this section. "SHOULD" is only used to allow similar perhaps more efficient approaches that result in approximately the same outcome.

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 Tunnelling 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 Tunnelling 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 [I-D.ietf-nvo3-geneve];

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 RFCXXXXX MUST follow the configuration requirements in Section 4 of RFCXXXXX to ensure it clears the outer IP ECN field to 0b00 when the outer PSN header is IP (v4 or v6).
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:
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

```
| V=0 | Type=3 | Reserved | E | P | Request Nonce |
| +---------------------------------------------------+---|---|---|---|---|
| 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 |
```

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:

- 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).

- 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:

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZZ</td>
<td>ECN Capability</td>
<td>RFCXXXX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZZ</td>
<td>ECN Capability</td>
<td>RFCXXXX</td>
</tr>
</tbody>
</table>
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]

11.2. Informative References


11.2. Informative References


Internet-Draft      ECN over IP-shim-(L2)-IP Tunnels         July 2019


Briscoe        Expires January 9, 2020          [Page 21]
Internet-Draft      ECN over IP-shim-(L2)-IP Tunnels           July 2019


Author’s Address

Bob Briscoe
Independent
UK

EMail: ietf@bobbriscoe.net
URL:   http://bobbriscoe.net/
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.
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  4
       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
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(2^8), 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:

- 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;
- 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 \( \text{GF}(2^m) \) 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

\( \text{GF}(q) \) 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 < ew_size <= 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 < cr <= 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 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:

- 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: \( ew\_size \leq ew\_max\_size \);
- 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: \( ew\_size \leq ew\_max\_size \).

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:

```c
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:

```c
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:

```c
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.

```c
/**
 * 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);
}
```

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 guarantees 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.
*/

Roca & Teibi Expires December 20, 2019 [Page 13]
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;
          }
        }
      }
  }
}
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:

   repair[i] = cc_tab[0] * src_0[i] XOR cc_tab[1] * src_1[i] XOR ... XOR cc_tab[ew_size - 1] * 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 FCCI). This FCCI 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

- 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 FCCI, 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.

```plaintext
+--------+--------+
| E      | WSR    |
+--------+--------+
| 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 |
+--------+--------+
```

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.

```
+--------------------------------+  
|           IP Header            |  
+--------------------------------+  
|        Transport Header        |  
+--------------------------------+  
|              ADU               |  
+--------------------------------+  
| Explicit Source FEC Payload ID |  
+--------------------------------+  
```

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.

```
 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  
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  
|                   Encoding Symbol ID (ESI)                   |  
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+  
```

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.

```
+--------------------------------+
|           IP Header            |
+--------------------------------+
|        Transport Header        |
+--------------------------------+
|     Repair FEC Payload ID      |
+--------------------------------+
|         Repair Symbol          |
+--------------------------------+
```

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;

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Repair_Key              |  DT   |NSS (# src symb in ew) |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            FSS_ESI                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

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

- 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:

- Organisation: Inria
- 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]).
- Maturity: prototype.
- Coverage: this software complies with the Sliding Window RLC FEC Scheme.
- Licensing: proprietary.
- Contact: 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;

- 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;

- 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(2^8)

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(2^8) 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(2^8) 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(2^8), 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:

- 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.
- XXXX refers to the Sliding Window Random Linear Codes (RLC) over GF(2^8) 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 Gorry 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


12.2. Informative References


[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/>.

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.
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 $1/16 \times 100 = 6.25\%$ of times.

Since the RLC FEC Schemes use of this PRNG will be limited to 16-bit seed values only, 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 65,536*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.

<table>
<thead>
<tr>
<th>value</th>
<th>occurrences</th>
<th>percentage (%)</th>
<th>(total of 20000000000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1250036799</td>
<td>6.2502</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1249995831</td>
<td>6.2500</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1250038674</td>
<td>6.2502</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1250000881</td>
<td>6.2500</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1250023929</td>
<td>6.2501</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1249986320</td>
<td>6.2499</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1249995587</td>
<td>6.2500</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1250020363</td>
<td>6.2501</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1249995276</td>
<td>6.2500</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1249982856</td>
<td>6.2499</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1249984111</td>
<td>6.2499</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1250009551</td>
<td>6.2500</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1249955768</td>
<td>6.2498</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1249994654</td>
<td>6.2500</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1250000569</td>
<td>6.2500</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1249978831</td>
<td>6.2499</td>
<td></td>
</tr>
</tbody>
</table>

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.
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:

- the source ADU flow they protect may or may not have real-time constraints;
- the source ADU flow may be a Constant Bitrate (CBR) or Variable BitRate (VBR) flow;
- with a VBR source ADU flow, the flow’s minimum and maximum bitrates may or may not be known;
- 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:

\[
\text{dw_max_size} = \frac{\text{max_lat} \times \text{br_in}}{8 \times E}
\]

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:

\[
dw_{max\_size} = (\text{max\_lat} \times br_{out} \times cr) / (8 \times E)
\]

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:

\[
\text{ew\_max\_size} = \text{dw\_max\_size} \times \text{WSR} / 255
\]

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:

\[
dw_{max\_size} = \text{max\_NSS\_observed} \times 255 / \text{WSR}
\]

A receiver can then chose an appropriate linear system maximum size:

\[
ls_{max\_size} \geq dw_{max\_size}
\]

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:

\[
\text{max\_lat\_for\_encoding} = \text{max\_lat} \times \text{WSR} / 255
\]

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:

\[ \text{ls}_{\text{max}} \text{size} > \text{dw}_{\text{max}} \text{size} \]
Usually the following choice is a good trade-off between decoding performance and extra CPU overhead:

\[ \text{ls\_max\_size} = 2 \times \text{dw\_max\_size} \]

When the \text{dw\_max\_size} is very small, it may be preferable to keep a minimum \text{ls\_max\_size} value (e.g., \text{LS\_MIN\_SIZE\_DEFAULT} = 40 symbols). Going below this threshold will not save a significant amount of memory nor CPU cycles. Therefore:

\[ \text{ls\_max\_size} = \max(2 \times \text{dw\_max\_size}, \text{LS\_MIN\_SIZE\_DEFAULT}) \]

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 \text{ls\_max\_size}. In that case lost ADUs will be recovered without relying on this optimization.

\[ \text{ls\_max\_size} \]

\[ \text{--------} \]

\[ \text{late source symbols (pot. decoded but not delivered)} \]

\[ \text{--------} \]

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 \text{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
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. The main advantage of TinyMT32 over MT is the use of a small internal state, compatible with most target platforms that include embedded devices, while keeping a reasonably good randomness that represents a significant improvement compared to the Park-Miller Linear Congruential PRNG. However, neither the TinyMT nor MT PRNG are meant to be used for cryptographic applications.
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, that can be found at [TinyMT-web] (TinyMT web site) and [TinyMT-dev] (Github site). This specialisation aims at having a simple-to-use and deterministic PRNG, as explained below. However, the TinyMT32 PRNG is not meant to be used for cryptographic applications.

TinyMT is a new small-sized variant introduced in 2011 of the Mersenne Twister (MT) PRNG [MT98]. This document focusses on the TinyMT32 variant (rather than TinyMT64) of the TinyMT PRNG, which outputs 32-bit unsigned integers.

The purpose of TinyMT is not to replace Mersenne Twister: TinyMT has a far shorter period ($2^{127} - 1$) than MT. The merit of TinyMT is in the small size of the internal state of 127 bits, far smaller than the 19937 bits of MT. The outputs of TinyMT satisfy several statistical tests for non-cryptographic randomness, including BigCrush in TestU01 [TestU01] and AdaptiveCrush [AdaptiveCrush],

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 ................................. 10
    7.2. Informative References .............................. 10
8. Authors’ Addresses ......................................... 11
leaving it well-placed for non-cryptographic usage, especially given
the small size of its internal state (see [TinyMT-web]). 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]) that suffers several known limitations (see for instance
[PTVF92], section 7.1, p. 279, and [RLC-ID], Appendix B).

The TinyMT32 PRNG initialization depends, among other things, on a
parameter set, namely (mat1, mat2, tmat). In order to facilitate the
use of this PRNG and make the sequence of pseudo-random numbers
depend only on the seed value, this specification requires the use of
a specific parameter set (see Section 3.1). This is a major
difference with respect to the implementation version 1.1
(2015/04/24) that leaves this parameter set unspecified.

Finally, the determinism of this PRNG, for a given seed, has been
carefully checked (see Section 3.3). It means that the same sequence
of pseudo-random numbers should be generated, no matter the target
execution platform and compiler, for a given initial seed value.
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:

- mat1 = 0x8f7011ee = 2406486510
- mat2 = 0xfc78ff1f = 4235788063
- tmat = 0x3793fdff = 932445695

This parameter set is the first entry of the precalculated parameter
sets in file tinymt32dc/tinymt32dc.0.1048576.txt, by Kenji Rikitake,
and available at [TinyMT-params]. This is also the parameter set
used in [KR12].
The TinyMT32 PRNG reference implementation is reproduced in Figure 1. This is a C language implementation, written for C99 [C99]. This reference implementation differs from the original source code as follows:

- the original copyright and license have been removed by the original authors who are now authors of this document, in accordance with BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info);
- the source code initially spread over the tinymt32.h and tinymt32.c files has been merged;
- the unused parts of the original source code have been removed. This is the case of the tinymt32_init_by_array() alternative initialisation function. This is also the case of the period_certification() function after having checked it is not required with the chosen parameter set;
- the unused constants TINYMT32_MEXP and TINYMT32_MUL have been removed;
- the appropriate parameter set has been added to the initialization function;
- the function order has been changed;
- certain internal variables have been renamed for compactness purposes;
- the const qualifier has been added to the constant definitions;
- the code that was dependant on the representation of negative integers by 2’s complements has been replaced by a more portable version;

<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/tinymt32dc.0.1048576.txt, by
 * Kenji Rikitake, available at:
 * https://github.com/jj1bdx/tinymtdc-longbatch/
 * 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;
    for (int i = 1; i < MIN_LOOP; i++) {
        s->status[i & 3] ^= i + UINT32_C(1812433253)
                         ^ (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);
    }
}
void tinymt32_generate_uint32 (tinymt32_t* s) {
    tinymt32_next_state(s);
    return tinymt32_temper(s);
}

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);

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;
    /*
    * The if (t1 & 1) {...} block below replaces:
    *     t0 ^= -((int32_t)(t1 & 1)) & s->tmat;
    * 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 (t1 & 1) {
        t0 ^= s->tmat;
    }
    return t0;
}

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 permitted 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 admits 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:

```c
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 have the same output as that provided by the reference implementation of Figure 1. In order to increase the compliancy confidence, this document proposes the following criteria. Using a seed value of 1, the first 50 values returned by `tinymt32_generate_uint32(s)` as 32-bit unsigned integers are equal to values provided in Figure 2, to be read line by line. 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.

<table>
<thead>
<tr>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
<th>Value 4</th>
<th>Value 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2545341989</td>
<td>981918433</td>
<td>3715302833</td>
<td>2387538352</td>
<td>3591001365</td>
</tr>
<tr>
<td>3820442102</td>
<td>2114400566</td>
<td>2196103051</td>
<td>2783359912</td>
<td>764534509</td>
</tr>
<tr>
<td>643179475</td>
<td>1822416315</td>
<td>881558334</td>
<td>4207026366</td>
<td>3690273640</td>
</tr>
<tr>
<td>3240535687</td>
<td>2921447122</td>
<td>3984931427</td>
<td>4092394160</td>
<td>44209675</td>
</tr>
<tr>
<td>2188315343</td>
<td>2908663843</td>
<td>1834519336</td>
<td>3774670961</td>
<td>3019990707</td>
</tr>
<tr>
<td>4065554902</td>
<td>1239765502</td>
<td>4035716197</td>
<td>3412127188</td>
<td>552822483</td>
</tr>
<tr>
<td>161364450</td>
<td>353727785</td>
<td>140085994</td>
<td>149132008</td>
<td>2547770827</td>
</tr>
<tr>
<td>4064042525</td>
<td>4078297538</td>
<td>2057335507</td>
<td>622384752</td>
<td>2041665899</td>
</tr>
<tr>
<td>2193913817</td>
<td>1080849512</td>
<td>33160901</td>
<td>662956935</td>
<td>642999063</td>
</tr>
<tr>
<td>3384709977</td>
<td>1723171522</td>
<td>3866752252</td>
<td>521822317</td>
<td>2292524454</td>
</tr>
</tbody>
</table>

Figure 2: First 50 decimal values (to be read per line) 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. However, neither the TinyMT nor MT PRNG are meant to be used for cryptographic applications.

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 Carl Wallace, Stewart Bryant, Greg Skinner, Mike Heard, 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, Adam Roach, Roman Danyliw, Barry Leiba, Martin Vigoureux, Eric Vyncke for their highly valuable feedbacks that greatly contributed to improve this specification.

7. References
7.1. Normative References


7.2. Informative References


Authors' Addresses

Mutsuo Saito
Hiroshima University
Japan
EMail: saito@math.sci.hiroshima-u.ac.jp

Makoto Matsumoto
Hiroshima University
Japan
EMail: m-mat@math.sci.hiroshima-u.ac.jp
Vincent Roca
INRIA
Univ. Grenoble Alpes
France
EMail: vincent.roca@inria.fr

Emmanuel Baccelli
INRIA
France
EMail: emmanuel.baccelli@inria.fr
The Impact of Transport Header Confidentiality on Network Operation and Evolution of the Internet
draft-ietf-tsvwg-transport-encrypt-07

Abstract

This document describes implications of applying end-to-end encryption at the transport layer. It identifies in-network uses of transport layer header information. It then reviews the implications of developing end-to-end transport protocols that use authentication to protect the integrity of transport information or encryption to provide confidentiality of the transport protocol header and expected implications of transport protocol design and network operation. Since transport measurement and analysis of the impact of network characteristics have been important to the design of current transport protocols, it also considers the impact on transport and application evolution.

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 5, 2020.

Copyright Notice

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

There is increased interest in, and deployment of, protocols that employ end-to-end encryption at the transport layer, including the transport layer headers. An example of such a transport is the QUIC transport protocol [I-D.ietf-quic-transport], currently being standardised in the IETF. Encryption of transport layer headers and payload data has many benefits in terms of protecting user privacy. These benefits have been widely discussed [RFC7258], [RFC7624], and
this document strongly supports the increased use of encryption in transport protocols. Encryption can also be used to prevent unwanted modification of transport header information by middleboxes. There are also, however, some costs, in that the widespread use of transport encryption requires changes to network operations, and complicates network measurement for research, operational, and standardisation purposes. The direction in which the use of transport header confidentiality evolves could have significant implications on the way the Internet architecture develops, and therefore needs to be considered as a part of protocol design.

The remainder of this document discusses some consequences of applying end-to-end encryption at the transport layer. It reviews the implications of developing end-to-end transport protocols that use encryption to provide confidentiality of the transport protocol header, and considers the effect of such changes on transport protocol design and network operations. It also considers anticipated implications on transport and application evolution.

Transports are also increasingly encrypting and authenticating the payload (i.e., the application data carried within the transport connection) end-to-end. Such protection is encouraged, and the implications of protecting the payload are not further discussed in this memo.

2. Context and Rationale

The transport layer provides end-to-end interactions between endpoints (processes) using an Internet path. Transport protocols layer directly over the network-layer service and are sent in the payload of network-layer packets. They support end-to-end communication between applications, supported by higher-layer protocols, running on the end systems (transport endpoints). This simple architectural view hides one of the core functions of the transport: to discover and adapt to the Internet path that is currently used. The design of Internet transport protocols is as much about trying to avoid the unwanted side effects of congestion on a flow and other capacity-sharing flows, avoiding congestion collapse, adapting to changes in the path characteristics, etc., as it is about end-to-end feature negotiation, flow control and optimising for performance of a specific application.

To achieve stable Internet operations the IETF transport community has to date relied heavily on measurement and the insights of the network operations community to understand the trade-offs, and to inform selection of appropriate mechanisms, to ensure a safe, reliable, and robust Internet (e.g., [RFC1273]). In turn, the network operator (and access provider) community has relied on being
able to understand the pattern and requirements of traffic passing over the Internet, both in aggregate and at the flow level.

2.1. Use of Transport Header Information in the Network

In-network measurement of transport flow characteristics can be used to enhance performance, and control cost and service reliability. Some operators have deployed functionality in middleboxes to both support network operations and deploy to enhance performance. A reliance on the presence and semantics of specific header information leads to ossification, where an endpoint could be required to supply a specific header to receive the network service that it desires. In some case this could be benign or advantageous to the protocol (e.g., recognising the start of a connection, or explicitly exposing protocol information can be expected to provide more consistent decisions by on-path devices than the use of diverse methods to infer semantics from other flow properties). In other cases the ossification could frustrate the evolution of the protocol (e.g., a mechanism implemented in a network device, such as a firewall, that required a header field to have only a specific known set of values would prevent the device from forwarding packets using a different version of a protocol that introduces a feature that changes the value of this field).

Experience developing Transport Layer Security [RFC8446], required a design that recognised that deployed middleboxes relied on the exposed information in Transport Layer Security (TLS) 1.2. Examples of the impact of ossification can be found in the development of Multipath TCP (MPTCP) and the TCP Fast Open option. The design of MPTCP had to be revised to account for middleboxes, so called "TCP Normalizers", that monitor the evolution of the window advertised in the TCP headers and that reset connections if the window does not grow as expected. Similarly, TCP Fast Open has had issues with middleboxes that remove unknown TCP options, that drop segments with unknown TCP options, that drop segments that contain data and have the SYN bit set, that drop packets with SYN/ACK that acknowledge data, or that disrupt connections that send data before the three-way handshake completes. In both cases, the issue was caused by middleboxes that had a hard-coded understanding of transport behaviour, and that interacted poorly with transports that tried to change that behaviour. Other examples have included middleboxes that rewrite TCP sequence and acknowledgement numbers but are unaware of the (newer) SACK option and don’t correctly rewrite selective acknowledgements to match the changes made to the fixed TCP header.
2.2. Encryption of Transport Header Information

Encryption is expected to form a basis for many future transport protocol designs. There are many motivations for deploying encrypted transports [RFC7624] (i.e., transport protocols that use encryption to provide confidentiality of some or all of the transport-layer header information), and encryption of transport payloads (i.e., Confidentiality of the payload data). Increasing public concerns about interference with Internet traffic have led to a rapidly expanding deployment of encryption to protect end-user privacy, e.g., QUIC [I-D.ietf-quic-transport]. Using encryption to provide confidentiality of the transport layer therefore brings some well-known privacy and security benefits. Although it is important that protocols either do not expose information where the usage could change in future protocols or that methods that utilise the information are robust to potential changes as protocols evolve over time.

Introducing cryptographic integrity checks for header fields can prevent undetected manipulation of the field by network devices, or undetected addition of information to a packet. This does not prevent inspection of the information by a device on path, and it is possible that such devices could develop mechanisms that rely on the presence of such a field or a known value in the field. In this context, specification of a non-encrypted transport header field explicitly allows protocol designers to make specific header information observable in the network. This supports other uses of this information by on-path devices, and at the same time this can be expected to lead to ossification of the transport header, because network forwarding could evolve to depend on the presence and/or value of these fields. To avoid unwanted inspection, a protocol could intentionally vary the format and/or value of exposed header fields (sometimes known as Greasing [I-D.thomson-quic-grease]).

A protocol design that uses header encryption with secure key distribution can provide confidentiality of some or all of the protocol header information. This prevents an on-path device from observing the header field. This prevents mechanisms being built that directly rely on the information or seek to infer semantics of an exposed header field and can therefore help reduce ossification of the transport layer. While encryption can hide transport header information, it does not prevent ossification of the network service.

People seeking to understand network traffic could come to rely on pattern inferences and other heuristics as the basis for network decision and to derive measurement data, creating new dependencies on the transport protocol (or the patterns of traffic it can generate). This use of machine-learning methods usually demands large data sets,
presenting its own requirements for collecting and distributing the data.

2.3. Encryption tradeoffs

The are architectural challenges and considerations in the way transport protocols are designed, and the ability to characterise and compare different transport solutions [Measure]. The decision about which transport headers fields are made observable offers trade-offs around authentication and confidentiality versus observability, network operations and management, and ossification. The impact differs depending on the activity, for example:

Network Operations and Research: Observable transport headers enable explicit measurement and analysis of protocol performance, network anomalies, and failure pathologies at any point along the Internet path. In many cases, it is important to relate observations to specific equipment/configurations or network segments.

Concealing transport header information makes performance/behaviour unavailable to passive observers along the path, Operators will be unable to use this information directly and may turn to more ambitious ways to collect, estimate, or infer that data. Operational practices aimed at guessing transport parameters are out of scope for this document, and are only mentioned here to recognize that encryption does not prevent operators from attempting to apply practices that were used with unencrypted transport headers.

Confidentiality of the transport payload could be provided while leaving some, or all, transport headers unencrypted (or providing this information in a network-layer extension), possibly with authentication. This provides many of the privacy and security benefits while supporting operations and research, but at the cost of ossifying the exposed headers.

Protection from Denial of Service: Observable transport headers currently provide useful input to classify and detect anomalous events (e.g., changes in application behaviour, distributed denial of service attacks). For this application to be effective, this needs to be able to uniquely disambiguate unwanted traffic.

Concealing transport header information would prevent separating anomalous traffic based on transport information. This could result in less-efficient identification of unwanted traffic or development of different methods (e.g., rate-limiting of uncharacterised traffic). Additional mechanisms will need to be
introduced, such as heuristics to disambiguate any unwanted traffic.

Network Troubleshooting and Diagnostics: Observable transport headers can be used by operators to support network troubleshooting and diagnostics. A flow experiencing packet loss or jitter looks like an unaffected flow when only observing network layer headers.

Concealing transport header information eliminates the incentive for operators to troubleshoot, since they cannot interpret the data. It can limit understanding of transport dynamics, such as the impact of packet loss or latency on the flows, or localizing the network segment causing the packet loss or latency. Additional mechanisms will be needed to help reconstruct or replace transport-level metrics for troubleshooting and diagnostics. These can add complexity and operational costs (e.g., in deploying additional functions in equipment or adding traffic overhead).

Network Traffic Analysis: Observable transport headers can support network traffic analysis to determine which transport protocols and features are being used across a network segment and to measure trends in the pattern of usage. For some applications end-to-end measurements/traces are sufficient, in other applications it is important to relate observations to specific equipment/configurations or network segments.

Concealing transport header information can make analysis harder or impossible. This could impact the ability for an operator to anticipate the need for network upgrades and roll-out. It can also impact the on-going traffic engineering activities performed by operators (such as determining which parts of the path contribute delay, jitter or loss). While this impact could, in many cases, be small, there are scenarios where operators directly support particular services (e.g., to explore issues relating to Quality of Service, QoS; the ability to perform fast re-routing of critical traffic, or support to mitigate the characteristics of specific radio links). The more complex the underlying infrastructure the more important this impact.

Open and Verifiable Network Data: Observable transport headers can provide open and verifiable measurement data. The ability of other stake holders to review transport header traces helps develop insight into performance and traffic contribution of specific variants of a protocol. Independently observed data is important to help ensure the health of the research and development communities.
Concealing transport header information can reduce the range of actors that 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 practices.

Compliance: Observable transport headers coupled with published transport specifications allow operators and regulators to check compliance. Independently verifiable performance metrics can also 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 the need to deploy 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]).

When transport header information is concealed, it is not possible to observe transport header information. Methods are still needed to confirm that the traffic produced conforms to the expectations of the operator or developer.

3. Current uses of Transport Headers within the Network

Despite transport headers having end-to-end meaning, some of these transport headers have come to be used in various ways within the Internet. 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 would affect how protocol information is used [RFC8404]. To understand these implications, it is first necessary to understand how transport layer headers are currently observed and/or modified by middleboxes within the network.

Transport protocols can be designed to encrypt or authenticate transport header fields. Authentication at the transport layer can be used to detect any changes to an immutable header field that were made by a network device along a path. The intentional modification of transport headers by middleboxes (such as Network Address Translation, NAT, or Firewalls) is not considered. Common issues concerning IP address sharing are described in [RFC6269].
3.1. Observing Transport Information in the Network

If in-network observation of transport protocol headers is needed, this requires knowledge of the format of the transport header:

- Flows need to be identified at the level required to perform the observation;
- The protocol and version of the header need to be visible, e.g., by defining the wire image [I-D.trammell-wire-image]. As protocols evolve over time and there could be a need to introduce new transport headers. This could require interpretation of protocol version information or connection setup information;
- The location and syntax of any observed transport headers need to be known. IETF transport protocols can specify this information.

The following subsections describe various ways that observable transport information has been utilised.

3.1.1. Flow Identification

Flow identification is a common function. For example, performed by measurement activities, QoS classification, firewalls, Denial of Service, DOS, prevention. This becomes more complex and less easily achieved when multiplexing is used at or above the transport layer.

Observable transport header information (together with information in the network header), has been used to identify a flow and the connection state of the flow, together with the protocol options being used. In some usages, a low-numbered (well-known) transport port number has been used to identify a protocol (although port information alone is not sufficient to guarantee identification of a protocol, since applications can use arbitrary ports, multiple sessions can be multiplexed on a single port, and ports can be re-used by subsequent sessions).

Transport protocols, such as TCP and the Stream Control Transport Protocol (SCTP) specify a standard base header that includes sequence number information and other data, with the possibility to negotiate additional headers at connection setup, identified by an option number in the transport header. UDP-based protocols can use, but sometimes do not use, well-known port numbers. Some flows can instead be identified by observing signalling protocol data (e.g., [RFC3261], [I-D.ietf-rtcweb-overview]) or through the use of magic numbers placed in the first byte(s) of the datagram payload [RFC7983].
Concealing transport header information can remove information used to classify flows by passive observers along the path and operators will be unable to use this information directly. Careful use of the network layer features can help address provide similar information in the case where the network is unable to inspect transport protocol headers. Operators could also turn to more ambitious ways to collect, estimate, or infer that data (for example heuristics based on the analysis of traffic patterns). For example, an operator no longer has access to Session Description Protocol (SDP) session descriptions to classify a flow carry as audio traffic. Instead, the operator might use heuristics to infer that short UDP packets with regular spacing carry audio traffic. Operational practices aimed at guessing transport parameters are out of scope for this document, and are only mentioned here to recognize that encryption does not prevent operators from attempting to apply practices that were used with unencrypted transport headers.

Confidentiality of the transport payload could be provided while leaving some, or all, transport headers unencrypted (or providing this information in a network-layer extension), possibly with authentication. This provides many of the privacy and security benefits while supporting operations and research, but at the cost of ossifying the exposed headers.

3.1.2. Metrics derived from Transport Layer Headers

Observable transport headers enable explicit measure and analysis protocol performance, network anomalies, and failure pathologies at any point along the Internet path. Some actors manage their portion of the Internet by characterizing the performance of link/network segments. Passive monitoring can observe traffic that does not encrypt the transport header information to make inferences from transport headers to derive these performance metrics.

A variety of open source and commercial tools have been deployed that utilise this information. The following metrics can be derived from transport header information:

Traffic Rate and Volume: Header information (e.g., sequence number and packet size) allows derivation of volume measures per-application, to characterise the traffic that uses a network segment or the pattern of network usage. This can be measured per endpoint or for an aggregate of endpoints (e.g., to assess subscriber usage). It can also be used to trigger measurement-based traffic shaping and to implement QoS support within the network and lower layers. Volume measures can be valuable for capacity planning and providing detail of trends, rather than the volume per subscriber.
Loss Rate and Loss Pattern: Flow loss rate can be derived (e.g., from transport sequence numbers) and has been used as a metric for performance assessment and to characterise transport behaviour. Understanding the location and root cause of loss can help an operator determine whether this requires corrective action. Network operators have used the variation in patterns of loss as a key performance metric, utilising this to detect changes in the offered service.

There are various causes of loss, including corruption of link frames (e.g., interference on a radio link), buffer overflow (e.g., due to congestion), policing (traffic management), buffer management (e.g., Active Queue Management, AQM [RFC7567]), and inadequate provision of traffic preemption. Understanding flow loss rate requires either maintaining per flow packet counters or by observing sequence numbers in transport headers. Loss can be monitored at the interface level by devices in the network. It is often valuable to understand the conditions under which packet loss occurs. This usually requires relating loss to the traffic flowing on the network node/segment at the time of loss.

Observation of transport feedback information (e.g., RTP Control Protocol (RTCP) reception reports [RFC3550], TCP SACK blocks) can increase understanding of the impact of loss and help identify cases where loss could have been wrongly identified, or the transport did not require the lost packet. It is sometimes more helpful to understand the pattern of loss, than the loss rate, because losses can often occur as bursts, rather than randomly-timed events.

Throughput and Goodput: The throughput achieved by a flow can be determined even when transport header information is concealed, providing the individual flow can be identified. Goodput [RFC7928] is a measure of useful data exchanged (the ratio of useful/total volume of traffic sent by a flow). This requires ability to differentiate loss and retransmission of packets (e.g., by observing packet sequence numbers in the TCP or the Real-time Transport Protocol, RTP, headers [RFC3550]).

Latency: Latency is a key performance metric that impacts application response time and user-perceived response time. It often indirectly impacts throughput and flow completion time. Latency 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 queuing in network buffers has often been observed as a significant factor.
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 in the network to determine not only the path RTT, but also to measure 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 deployment and 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 [RFC8290] [RFC8289] [RFC8033] often cannot scale across all possible deployment scenarios.

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 the network (e.g., jitter metrics are often cited when characterising paths supporting real-time traffic). To assess the performance of such applications, it can be necessary to measure 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. Since this impacts transport performance, network tools are needed to detect and measure unwanted/excessive reordering.

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 within deployed infrastructure, and inform decisions about how to progress such mechanisms. Key performance indicators are retransmission rate, packet drop rate, sector utilisation level, a measure of reordering, peak rate, the ECN congestion experienced (CE) marking rate, etc.

Metrics have been defined that evaluate whether a network has maintained packet order on a packet-by-packet basis [RFC4737] and [RFC5236].

Techniques for measuring reordering typically observe packet sequence numbers. 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. As with other measurement, metadata is often needed to understand 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 in the network. A user of summary measurement data needs to trust the source of this data and the method used to generate the summary information.

This information can support network operations, e.g. to inform capacity planning and assist in determining the need for equipment and/or configuration changes by network operators. It can also inform Internet engineering activities by informing the development of new protocols, methodologies, and procedures.

3.1.3. Transport use of Network Layer Header Fields

Information from the transport protocol can be used by a multi-field 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 and by firewalls that use the information to implement access rules (see also section 2.2.2 of [RFC8404]). Network-layer classification methods that rely on a multi-field classifier (e.g. Inferring QoS from the 5-tuple or choice of application protocol) are incompatible with transport protocols that encrypt the transport information. Traffic that cannot be classified, will typically receive a default treatment.

Transport information can also be explicitly set in network-layer header fields that are not encrypted. This can provide information
to enable a different forwarding treatment by the network, 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 could wish to hide 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 network needs of individual sub-flows. There are several ways this could be done:

IP Address: Applications expose the addresses used by endpoints, and this is used in the forwarding decisions in network devices. Address and other protocol information can be used by a Multi-Field (MF) classifier to determine how traffic is treated [RFC2475], and hence the quality of experience for a flow.

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 subflows. 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". To promote privacy, the Flow Label assignment needs to avoid introducing linkability that a network device may observe. Once set, a label can provide information that can help inform network-layer queuing and forwarding [RFC6438](e.g. for Equal Cost Multi-Path, ECMP, routing, and Link Aggregation, LAG) [RFC6294]. [RFC6438] describes considerations when used with IPsec.

Using the Network-Layer Differentiated Services Code Point: Applications can expose their delivery expectations to the network 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 queuing and forwarding, rather than an operator inferring traffic requirements from transport and application headers via a multi-field classifier.

Since the DSCP value can impact the quality of experience for a flow, observations of service performance need to consider this
Using Explicit Congestion Marking: ECN [RFC3168] is a transport mechanism that utilises 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 Congestion Experienced (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 need 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].

When transport headers are concealed, operators will be unable to use this information directly. Careful use of the network layer features can help address provide similar information in the case where the network is unable to inspect transport protocol headers.

3.2. Transport Measurement

The common language between network operators and application/content providers/users is packet transfer performance at a layer that all can view and analyse. For most packets, this has been the transport layer, until the emergence of QUIC, with the obvious exception of Virtual Private Networks (VPNs) and IPsec.

When encryption conceals more layers in each packet, people seeking understanding of the network operation rely more on pattern inferences and other heuristics reliance on pattern inferences and accuracy suffers. For example, the traffic patterns between server and browser are dependent on browser supplier and version, even when the sessions use the same server application (e.g., web e-mail access). It remains to be seen whether more complex inferences can be mastered to produce the same monitoring accuracy (see section 2.1.1 of [RFC8404]).
When measurement datasets are made available by servers or client endpoints, additional metadata, such as the state of the network, is often required 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/device under evaluation [RFC7799].

Packet sampling techniques are used to scale the processing involved in observing packets on high rate links. This exports only the packet header information of (randomly) selected packets. The utility of these measurements depends on the type of bearer and number of mechanisms used by network devices. Simple routers are relatively easy to manage, a device with more complexity demands understanding of the choice of many system parameters. This level of complexity exists when several network methods are combined.

This section discusses topics concerning observation of transport flows, with a focus on transport measurement.

3.2.1. Point of Observation

On-path measurements are particularly useful for locating the source of problems, or to assess the performance of a network segment or a particular device configuration. Often issues can only be understood in the context of the other flows that share a particular path, common network device, 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 understanding implies knowledge of how traffic is assigned to any sub-queues used for flow scheduling, but can also require information about how the traffic dynamics impact active queue management, starvation prevention mechanisms, and circuit-breakers.

Sometimes multiple on-path observation points are needed. By correlating observations of headers at multiple points along the path (e.g., at the ingress and egress of a network segment), an observer can determine the contribution of a portion of the path to an observed metric, to locate a source of delay, jitter, loss, reordering, congestion marking, etc.

3.2.2. Use by Operators to Plan and Provision Networks

Traffic measurements (e.g., traffic volume, loss, latency) is used by operators to help plan deployment of new equipment and configuration in their 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. This measurement information can also be correlated with billing information when this is also collected by an operator.

A network operator supporting traffic that uses transport header encryption might not have access to per-flow measurement data. Trends in aggregate traffic can be observed and can be related to the endpoint addresses being used, but it may be impossible to correlate patterns in measurements with changes in transport protocols (e.g., the impact of changes in introducing a new transport protocol mechanism). This increases the dependency on other indirect sources of information to inform planning and provisioning.

3.2.3. Service Performance Measurement

Traffic measurements (e.g., traffic volume, loss, latency) can be used by various actors to help analyse the performance offered to the users of a network segment, and to inform operational practice.

While active measurements (see section 3.4 of [RFC7799]) may be used within a network, 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 3.2.1). However, passive measurements can rely on observing transport headers which is not possible if those headers are encrypted.

3.2.4. Measuring Transport to Support Network Operations

Information provided by tools observing transport headers can help determine whether mechanisms are needed in the network to prevent flows from acquiring excessive network capacity. Operators can implement operational practices to manage traffic flows (e.g., to prevent flows from acquiring excessive network capacity under severe congestion) by deploying rate-limiters, traffic shaping or network transport circuit breakers [RFC8084].

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 use in the shared Internet. TCP algorithms have been continuously improved over decades and they have reached a level of efficiency and correctness that custom application-layer mechanisms will struggle to easily duplicate [RFC8085].
A standards-compliant TCP stack provides congestion control that may therefore be 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 transport (e.g., TCP and SCTP).

However, when anomalies are detected, tools can interpret the transport protocol header information to help understand the impact of specific transport protocols (or protocol mechanisms) on the other traffic that shares a network. An observation in the network 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 in the face of 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 that contribute to persistent congestion is important to safe operation of network infrastructure, and mechanisms 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].

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 are required to employ mechanisms to prevent congestion collapse, avoid unacceptable contributions to jitter/latency, and to establish an acceptable share of capacity with concurrent traffic [RFC8085].

A network operator needs tools to understand if datagram flows comply with congestion control expectations and therefore whether there is a need to deploy methods such as rate-limiters, transport circuit breakers or other methods to enforce acceptable usage for the offered service.

UDP flows that expose a well-known header by specifying the format of header fields can allow information to 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 the RTP and RTCP header information of real-time flows (see
Section 3.1.2, and the Secure RTP extensions [RFC3711] were explicitly designed to expose header information to enable such observation.

3.3. Use for Network Diagnostics and Troubleshooting

Transport header information can be useful for a variety of operational tasks [RFC8404]: to diagnose network problems, assess network provider performance, evaluate equipment/protocol performance, capacity planning, management of security threats (including denial of service), and responding to user performance questions. Sections 3.1.2 and 5 of [RFC8404] provide further examples. These tasks seldom involve the need to determine the contents of the transport payload, or other application details.

A network operator supporting traffic that uses transport header encryption can see only encrypted transport headers. This prevents deployment of performance measurement tools that rely on transport protocol information. Choosing to encrypt all the information reduces the ability of an operator to observe transport performance and could limit the ability of network operators to trace problems, make appropriate QoS decisions, or response to other queries about the network service. For some this will be blessing, for others it may be a curse. For example, operational performance data about encrypted flows needs to be determined by traffic pattern analysis, rather than relying on traditional tools. This can impact the ability of the operator to respond to faults, it could require reliance on endpoint diagnostic tools or user involvement in diagnosing and troubleshooting unusual use cases or non-trivial problems. A key need here is for tools to provide useful information during network anomalies (e.g., significant reordering, high or intermittent loss).

Measurements can be used to monitor the health of a portion of the Internet, to provide early warning of the need to take action. They can assist in setting buffer sizes, debugging and diagnosing the root causes of faults that concern a particular user’s traffic. They can also be used to support post-mortem investigation after an anomaly to determine the root cause of a problem.

In some cases, measurements could involve active injection of test traffic to perform a measurement. 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 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. An alternative approach is to use in-network techniques that observe transport packet headers added while traffic traverses an operational network to make the measurements. These measurements do not require the cooperation of an endpoint.

In other cases, measurement involves dissecting network traffic flows. The observed transport layer information can help identify whether the link/network tuning is effective and alert to potential problems that can be hard to derive from link or device measurements alone. The design trade-offs for radio networks are often very different from those of wired networks. A radio-based network (e.g., cellular mobile, enterprise WiFi, satellite access/back-haul, point-to-point radio) has the complexity of a subsystem that performs radio resource management, with direct impact on the available capacity, and potentially loss/reordering of packets. The impact of the pattern of loss and congestion, differs for different traffic types, correlation with propagation and interference can all have significant impact on the cost and performance of a provided service. The need for this type of information is expected to increase as operators bring together heterogeneous types of network equipment and seek to deploy opportunistic methods to access radio spectrum.

A flow that conceals its transport header information could imply "don’t touch" to some operators. This could limit a trouble-shooting response to "can’t help, no trouble found".

3.4. Header Compression

Header compression saves link bandwidth by compressing network and transport protocol headers on a per-hop basis. It was widely used with low bandwidth dial-up access links, and still finds application on wireless links that are subject to capacity constraints. Header compression has been specified for use with TCP/IP and RTP/UDP/IP flows [RFC2507], [RFC2508], [RFC4995].

While it is possible to compress only the network layer headers, significant bandwidth savings can be made if both the network and transport layer headers are compressed together as a single unit. The Secure RTP extensions [RFC3711] were explicitly designed to leave the transport protocol headers unencrypted, but authenticated, since support for header compression was considered important. Encrypting the transport protocol headers does not break such header compression, but does cause it to fall back to compressing only the network layer headers, with a significant reduction in efficiency. This may have operational impact.
4. Encryption and Authentication of Transport Headers

End-to-end encryption can be applied at various protocol layers. It can be applied above the transport to encrypt the transport payload. Encryption methods can hide information from an eavesdropper in the network. Encryption can also help protect the privacy of a user, by hiding data relating to user/device identity or location. Neither an integrity check nor encryption methods prevent traffic analysis, and usage needs to reflect that profiling of users, identification of location and fingerprinting of behaviour can take place even on encrypted traffic flows. Any header information that has a clear definition in the protocol’s message format(s), or is implied by that definition, and is not cryptographically confidentiality-protected can be unambiguously interpreted by on-path observers [I-D.trammell-wire-image].

There are several motivations:

- One motive to use encryption is a response to perceptions that the network has become ossified by over-reliance on middleboxes that prevent new protocols and mechanisms from being deployed. This has lead to a perception that there is too much "manipulation" of protocol headers within the network, and that designing to deploy in such networks is preventing transport evolution. In the light of this, a method that authenticates transport headers may help improve the pace of transport development, by eliminating the need to always consider deployed middleboxes [I-D.trammell-plus-abstract-mech], or potentially to only explicitly enable middlebox use for particular paths with particular middleboxes that are deliberately deployed to realise a useful function for the network and/or users[RFC3135].

- Another motivation stems from increased concerns about privacy and surveillance. Some Internet users have valued the ability to protect identity, user location, and defend against traffic analysis, and have used methods such as IPsec Encapsulated Security Payload (ESP), Virtual Private Networks (VPNs) and other encrypted tunnel technologies. Revelations about the use of pervasive surveillance [RFC7624] have, to some extent, eroded trust in the service offered by network operators, and following the Snowden revelation in the USA in 2013 has led to an increased desire for people to employ encryption to avoid unwanted "eavesdropping" on their communications. Concerns have also been voiced about the addition of information to packets by third parties to provide analytics, customization, advertising, cross-site tracking of users, to bill the customer, or to selectively allow or block content. Whatever the reasons, there are now activities in the IETF to design new protocols that could include
some form of transport header encryption (e.g., QUIC [I-D.ietf-quic-transport]).

Authentication methods (that provide integrity checks of protocols fields) have also been specified at the network layer, and this also protects transport header fields. The network layer itself carries protocol header fields that are increasingly used to help forwarding decisions reflect the need of transport protocols, such as the IPv6 Flow Label [RFC6437], DSCP, and ECN fields.

The use of transport layer authentication and encryption exposes a tussle between middlebox vendors, operators, applications developers and users.

- On the one hand, future Internet protocols that enable large-scale 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.

- On the other hand, encryption of transport layer header information has implications for people who are responsible for operating networks and researchers and analysts seeking to understand the dynamics of protocols and traffic patterns.

Whatever the motives, a decision to use pervasive transport header encryption will have implications on the way in which design and evaluation is performed, and which can in turn impact the direction of evolution of the transport protocol stack. While the IETF can specify protocols, the success in actual deployment is often determined by many factors [RFC5218] that are not always clear at the time when protocols are being defined.

The following briefly reviews some security design options for transport protocols. A Survey of Transport Security Protocols [I-D.ietf-taps-transport-security] provides more details concerning commonly used encryption methods at the transport layer.

Authenticating the Transport Protocol Header: Transport layer header information can be authenticated. An integrity check that protects the immutable transport header fields, but can still expose the transport protocol header information in the clear, allowing in-network devices to observe these fields. An integrity check is not able to prevent in-network modification, but can prevent a receiving from accepting changes and avoid impact on the transport protocol operation.

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. TCP-AO may 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 in-network modification. Secure RTP [RFC3711] is another example of a transport protocol that allows header authentication.

**Greasing:** Transport layer header information that is observable can be observed in the network. 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 in-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. This behaviour, known as GREASE (Generate Random Extensions And Sustain Extensibility), is designed to avoid a network device ossifying the use of a specific observable field. Greasing seeks to ease deployment of new methods. It can also prevent in-network devices utilising the information in a transport header, or can make an observation robust to a set of changing values, rather than a specific set of values.

**Encrypting the Transport Payload:** The transport layer payload can be encrypted to protect the content of transport segments. This leaves transport protocol header information in the clear. The integrity of immutable transport header fields could be protected by combining this with an integrity check.

Examples of encrypting the payload include Transport Layer Security (TLS) over TCP [RFC8446] [RFC7525], Datagram TLS (DTLS) over UDP [RFC6347] [RFC7525], Secure RTP [RFC3711], and TCPcrypt [I-D.ietf-tcpinc-tcpcrypt] which permits opportunistic encryption of the TCP transport payload.

**Encrypting the Transport Headers and Payload:** The network layer payload could be encrypted (including the entire transport header and the payload). This method provides confidentiality of the
entire transport packet. It therefore does not expose any transport information to devices in the network, which also prevents modification along a network path.

One example of encryption at the network layer is use of IPsec Encapsulating Security Payload (ESP) [RFC4303] in tunnel mode. This encrypts and authenticates all transport headers, preventing visibility of the transport headers by in-network devices. Some Virtual Private Network (VPN) methods also encrypt these headers.

Selectively Encrypting Transport Headers and Payload: A transport protocol design can encrypt selected header fields, while also choosing to authenticate the entire transport header. This allows specific transport header fields to be made observable by network devices. End-to-end integrity checks can prevent an endpoint from undetected modification of the immutable transport headers.

Mutable fields in the transport header provide opportunities for middleboxes to modify the transport behaviour (e.g., the extended headers described in [I-D.trammell-plus-abstract-mech]). This considers only immutable fields in the transport headers, that is, fields that can be authenticated End-to-End across a path.

An example of a method that encrypts some, but not all, transport 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 are required to use variable format fields. Instead, fields of a specific type ought to always be sent with the same level of confidentiality or integrity protection.

As seen, different transports use encryption to protect their header information to varying degrees. There is, however, a trend towards increased protection with newer transport protocols.

5. Addition of Transport Information to Network-Layer Protocol Headers

Some measurements can be made by adding additional protocol headers carrying operations, administration and management (OAM) information to packets at the ingress to a 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]) and removing the additional header at the egress of the maintenance domain. This approach enables some types of measurements, but 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 required 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. This has the advantage that a single header can support all transport protocols, but there could also be less desirable implications of separating the operation of the transport protocol from the measurement framework.

Another example of a network-layer approach 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 receiving transport endpoints 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 method needs to be explicitly enabled at the sender.

Current measurements suggest it can be undesirable to rely on methods requiring the presence of network options or extension headers. IPv4 network options are often not supported (or are carried on a slower processing path) and some IPv6 networks are also known to drop packets that set an IPv6 header extension (e.g., [RFC7872]). Another disadvantage is that protocols that separately expose header information do not necessarily have an incentive to expose the information that is utilised by the protocol itself, and could manipulate the exposed header information to gain an advantage from the network.

6. Implications of Protecting the Transport Headers

The choice of which fields to expose and which to encrypt is a design choice for the transport protocol. Any selective encryption method requires trading two conflicting goals for a transport protocol designer to decide which header fields to encrypt. Security work typically employs a design technique that seeks to expose only what is needed. This approach provides incentives to not reveal any information that is not necessary for the end-to-end communication. However, there can be performance and operational benefits in exposing selected information to network tools.

This section explores key implications of working with encrypted transport protocols.
6.1. Independent Measurement

Independent observation by multiple actors is important for scientific analysis. Encrypting transport header encryption changes the ability for other actors to collect and independently analyse data. Internet transport protocols employ a set of mechanisms. Some of these need to work in cooperation with the network layer - loss detection and recovery, congestion detection and congestion control, some of these need to work only end-to-end (e.g., parameter negotiation, flow-control).

The majority of present Internet applications use two well-known transport protocols, TCP and UDP. Although TCP represents the majority of current traffic, some real-time applications use UDP, and much of this traffic utilises RTP format headers in the payload of the UDP datagram. Since these protocol headers have been fixed for decades, a range of tools and analysis methods have become common and well-understood.

Protocols that expose the state information used by the transport protocol in their header information (e.g., timestamps used to calculate the RTT, packet numbers used to assess congestion and requests for retransmission) provide an incentive for the sending endpoint to provide correct information, increasing confidence that the observer understands the transport interaction with the network. For example, when TCP is used over an unencrypted network path (i.e., one that does not use IPsec or other encryption below the transport), it implicitly exposes header information that can be used for measurement at any point along the path. This information is necessary for the protocol’s correct operation, therefore there is no incentive for a TCP implementation to put incorrect information in this transport header. A network device can have confidence that the well-known (and ossified) transport information represents the actual state of the endpoints.

When encryption is used to conceal some or all of the transport headers, the transport protocol choose what information to reveal to the network about its internal state, what information to leave encrypted, and what fields to grease to protect against future ossification. Such a transport could be designed, for example, to provide summary data regarding its performance, congestion control state, etc., or to make an explicit measurement signal available. For example, a QUIC endpoint could set the spin bit to reflect to explicitly reveal a session’s RTT (I-D.ietf-quic-spin-exp).

When providing or using such information, it becomes important to consider the privacy of the user and their incentive for providing accurate and detailed information. Protocols that selectively reveal
some transport state or measurement signals are choosing to establish a trust relationship with the network operators. There is no protocol mechanism that can guarantee that the information provided represents the actual transport state of the endpoints, since those endpoints can always send additional information in the encrypted part of the header, to update to replace whatever they reveal. This reduces the ability to independently measure and verify that a protocol is behaving as expected. Some operational uses need the information to contain sufficient detail to understand, and possibly reconstruct, the network traffic pattern for further testing; such operators must gain the trust of transport protocol implementers if they are to correctly reveal such information.

OAM data records [I-D.ietf-ippm-ioam-data] could be embedded into a variety of encapsulation methods at different layers to support the goals of a specific operational domain. OAM-related metadata can support functions such as performance evaluation, path-tracing, path verification information, classification and a diversity of other uses. When encryption is used to conceal some or all of the transport headers, analysis will require coordination between actors at different layers to successfully characterise flows and correlate the performance or behavior of a specific mechanism with the configuration and traffic using operational equipment (e.g. Combining transport and network measurements to explore congestion control dynamics or the implications of active queue management).

For some usage a standardised endpoint-based logging format (e.g., based on Quic-Trace [Quic-Trace]) could offer an alternative to in-network measurement. Such information will have a diversity of uses - examples include developers wishing to debug/understand the transport/application protocols with which they work, to researchers seeking to spot trends, anomalies and to characterise variants of protocols. This use will need to establish the validity and provenance of the logging information (e.g., to establish how and when traces were captured).

However, endpoint logs do not provide equivalent information to in-network measurements. In particular, endpoint logs contain only a part of the information needed to understand the operation of network devices and identify issues such as link performance or capacity sharing between multiple flows. Additional information is needed to determine which equipment/links are used and the configuration of equipment along the network paths being measured.
6.2. Characterising "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.

If "unknown" or "uncharacterised" traffic patterns form a small part of the traffic aggregate passing through a network device or segment of the network, the dynamics of the uncharacterised traffic may not have a significant collateral impact on the performance of other traffic that shares this network segment. Once the proportion of this traffic increases, the need to monitor the traffic and determine if appropriate safety measures need 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. On a shorter timescale, information may also need to be collected to manage denial of service attacks against the infrastructure.

6.3. Accountability and Internet Transport Protocols

Information provided by tools observing transport headers can be used to classify traffic, and to limit the network capacity used by certain flows, as discussed in Section 3.2.4). Equally, operators could use analysis of transport headers and transport flow state to demonstrate that they are not providing differential treatment to certain flows. Obfuscating or hiding this information using encryption may lead operators and maintainers of middleboxes (firewalls, etc.) to seek other methods to classify, and potentially other mechanisms to condition, network traffic.

A lack of data that reduces the level of precision with which flows can be classified also reduces the design space for conditioning mechanisms (e.g., rate limiting, circuit breaker techniques [RFC8084], or blocking of uncharacterised traffic), and this needs to be considered when evaluating the impact of designs for transport encryption [RFC5218].

6.4. Impact on Operational Cost

Many network operators currently utilise observed transport information as a part of their operational practice, and have developed tools and operational practices based around currently deployed transports and their applications. Encryption of the transport information prevents tools from directly observing this
information. A variety of open source and commercial tools have been deployed that utilise this information for a variety of short and long term measurements.

The network will not break just because transport headers are encrypted, although alternative diagnostic and troubleshooting tools would need to be developed and deployed. Introducing a new protocol or application can require these tool chains and practice to be updated, and may in turn impact operational mechanisms, and policies. Each change can introduce associated costs, including the cost of collecting data, and the tooling needed to handle multiple formats (possibly as these co-exist in the network, when measurements need to span time periods during which changes are deployed, or to compare with historical data). These costs are incurred by an operator to manage the service and debug network issues.

At the time of writing, the additional operational cost of using encrypted transports is not yet well understood. Design trade-offs could mitigate these costs by explicitly choosing to expose selected information (e.g., header invariants and the spin-bit in QUIC[I-D.ietf-quic-transport]), the specification of common log formats and development of alternative approaches.

6.5. Impact on Research, Development and Deployment

Evolution and the ability to understand (measure) the impact need to proceed hand-in-hand. Observable transport headers can 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. This helps understanding the interactions between cooperating protocols and network mechanism, the implications of sharing capacity with other traffic and the impact of different patterns of usage. The ability of other stake holders to review transport header traces helps develop insight into performance and traffic contribution of specific variants of a protocol.

In development of new transport protocol mechanisms, attention needs to be paid to the expected scale of deployment. Whatever the mechanism, experience has shown that it is often difficult to correctly implement combination of mechanisms [RFC8085]. 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.
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. There has been recent 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, DCTP, 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.

Concealing 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 practices. 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.

Independently observed data is also important to ensure the health of the research and development communities and can help promote acceptance of proposed specifications by the wider community (e.g., as a method to judge the safety for Internet deployment) and provides valuable input during standardisation. Open standards motivate a desire to include independent observation and evaluation of performance data, which in turn demands control over where and when measurement samples are collected. This requires consideration of the methods used to observe data and the appropriate balance between encrypting all and no transport information.

7. Conclusions

Confidentiality and strong integrity checks have properties that are being incorporated into new protocols and that 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.
The traffic that can be observed by on-path network devices 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 less. The desire to understand the traffic and protocol interactions typically grows as the proportion of traffic increases in volume. The challenges increase when multiple instances of an evolving protocol contribute to the traffic that share network capacity.

An increased pace of evolution therefore needs to be accompanied by methods that can be successfully deployed and used across operational networks. This leads to a need for network operators (at various level (ISPs, enterprises, firewall maintainer, etc) to identify appropriate operational support functions and procedures.

Protocols that change their transport header format (wire format) or their behaviour (e.g., algorithms that are needed to classify and characterise the protocol), will require new tooling to be developed to catch-up with the change. If the currently deployed tools and methods are no longer relevant then it may no longer be possible to correctly measure performance. This can increase the response-time after faults, and can impact the ability to manage the network resulting in traffic causing traffic to be treated inappropriately (e.g., rate limiting because of being incorrectly classified/monitored).

There are benefits in exposing consistent information to the network that avoids traffic being inappropriately classified and then receiving a default treatment by the network. The flow label and DSCP fields provide examples of how transport information can be made available for network-layer decisions. Extension headers could also be used to carry transport information that can inform network-layer decisions. Other information may also be useful to various stakeholder (as described in earlier sections), however this document does not make recommendations about what information should be exposed, to whom it should be observable, or how this will be achieved.

To achieve stable Internet operations the IETF transport community has to date relied heavily on measurement and insights of the network operations community to understand the trade-offs, and to inform selection of appropriate mechanisms, to ensure a safe, reliable, and robust Internet (e.g., [RFC1273],[RFC2914]).

There are trade-offs and implications of increased use of encryption. Transport protocol designers have often ignored the implications of
whether the information in transport header fields can or will be used by in-network devices, and the implications this places on protocol evolution. This motivates a design that provides confidentiality of the header information. It can be expected that a lack of visibility of transport header information can impact the ways that protocols are deployed, standardised, and their operational support. The impact of hiding transport headers therefore needs to be considered in the specification and development of protocols and standards. This has a potential impact on the way in which the IRTF and IETF develop new protocols, specifications, and guidelines:

- Coexistence of Transport and Network Device Protocols/Configuration: Transmission Control Protocol (TCP) is currently the predominant transport protocol used over Internet paths. Its many variants have broadly consistent approaches to avoiding congestion collapse, and to ensuring the stability of the Internet. Increased use of transport layer encryption can overcome ossification, allowing deployment of new transports and different types of congestion control. This flexibility can be beneficial, but it can come at the cost of fragmenting the ecosystem. There is little doubt that developers will try to produce high quality transports for their intended target uses, but it is not yet clear there are sufficient incentives to ensure good practice that benefits the wide diversity of requirements for the Internet community as a whole.

- Supporting Common Specifications: Common open specifications can stimulate engagement by developers, users, and researchers. Increased diversity, and the ability to innovate without public scrutiny, risks point solutions that optimise for specific needs, but accidentally disrupt operations of/in different parts of the network. The social contract that maintains the stability of the Internet relies on accepting common interworking specifications.

- 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, functions, and/or configurations. This can also help understand complex feature interactions. An inability to observe transport protocol information can limit the ability 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 need to be developed.

- Operational Practice: The network operations community relies on being able to understand the pattern and requirements of traffic
passing over the Internet, both in aggregate and at the flow level. These operational practices have developed based on the information available from unencrypted transport headers. The IETF supports this activity by developing operations and management specifications, interface specifications, and associated Best Current Practice (BCP) specifications. Concealing transport header information impacts current practice and demand new specifications.

- Research and Development: Concealing transport 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 need 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 this results in reduced availability of open data, it could eliminate the independent self-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 Groups (ICCRG) and research publications in reviewing new transport mechanisms and assessing the impact of their experimental deployment).

The choice of whether future transport protocols encrypt their protocol headers needs to be taken based not solely on security and privacy considerations, but also taking into account the impact on operations, standards, and research. 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."

As part of a protocol’s design, the community therefore needs to weigh the benefits of ossifying common headers versus the potential demerits of exposing specific information that could be observed along the network path, to ensure network operators, researchers and other stakeholders have appropriate tools to manage their networks and enable stable operation of the Internet as new protocols are deployed. An appropriate balance will emerge over time as real instances of this tension are considered [RFC7258]. This balance between information exposed and information concealed 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 in the various sections of the 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 [I-D.ietf-taps-transport-security].

Confidentiality and strong integrity checks have properties that can also be incorporated into the design of a transport protocol. Integrity checks can protect an endpoint from undetected modification of protocol fields by network devices, whereas encryption and obfuscation or greasing can further prevent these headers being utilised by network devices. Hiding headers can therefore provide the 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 benefits of ossifying common headers, versus the potential demerits of exposing specific information that could be observed along the network path to provide tools to manage new variants of protocols.

A protocol design that uses header encryption can provide confidentiality of some or all of the protocol header information. This prevents an on-path device from knowledge of the header field. It therefore prevents mechanisms being built that directly rely on the information or seek to infer semantics of an exposed header field. Hiding headers can limit the ability to measure and characterise traffic.

Exposed transport headers are sometimes utilised as a part of the information to detect anomalies in network traffic. 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 normalization 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 are sometimes also utilised as 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 may 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 on-path attacks. An additional processing cost can be incurred when decryption needs to be attempted before a receiver is able to discard injected packets.

Open standards motivate a desire for this evaluation to include independent observation and evaluation of performance data, which in turn suggests control over where and when measurement samples are collected. This requires consideration of the appropriate balance between encrypting all and no transport information. 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.

9. IANA Considerations

XX RFC ED - PLEASE REMOVE THIS SECTION XXX

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, and other members of the 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. The opinions expressed and arguments employed reflect only the authors’ view. The European Commission is not responsible for any use that may 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]

[I-D.ietf-ippm-ioam-data]

[I-D.ietf-quic-spin-exp]

[I-D.ietf-quic-transport]

[I-D.ietf-rtcweb-overview]

[I-D.ietf-taps-transport-security]

[I-D.ietf-tcpinc-tcpcrypt]
[I-D.ietf-tsvwg-rtcweb-qos]

[I-D.thomson-quic-grease]

[I-D.trammell-plus-abstract-mech]

[I-D.trammell-wire-image]


[Quic-Trace]
"https:QUIC trace utilities //github.com/google/quic-trace".


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 draft 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.

Authors’ Addresses

Godred Fairhurst
University of Aberdeen
Department of Engineering
Fraser Noble Building
Aberdeen AB24 3UE
Scotland

EMail: 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
URI: https://csperkins.org//
Transport Options for UDP
draft-ietf-tsvwg-udp-options-07.txt

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, except to format it for publication as an RFC or to translate it into languages other than English.

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/ietf/1id-abstracts.txt

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

This Internet-Draft will expire on September 8, 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 (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

Touch Expires September 8, 2019 [Page 1]
Abstract

Transport protocols are extended through the use of transport header options. This document experimentally extends UDP by indicating the location, syntax, and semantics for UDP transport layer options.

Table of Contents

1. Introduction ................................................... 3
2. Conventions used in this document .............................. 3
3. Background ..................................................... 3
4. The UDP Option Area ............................................ 4
5. UDP Options .................................................... 7
   5.1. End of Options List (EOL) .................................. 8
   5.2. No Operation (NOP) ....................................... 9
   5.3. Option Checksum (OCS) ................................... 9
   5.4. Alternate Checksum (ACS) ................................. 11
   5.5. Lite (LITE) .............................................. 11
   5.6. Maximum Segment Size (MSS) ............................. 13
   5.7. Fragmentation (FRAG) .................................... 14
   5.8. Coupling FRAG with LITE ................................. 16
   5.9. Timestamps (TIME) ....................................... 17
   5.10. Authentication and Encryption (AE) .................... 18
6. Echo request (REQ) and echo response (RES) .................... 19
   6.1. Experimental (EXP) ..................................... 19
7. Rules for designing new options ................................. 20
8. Option inclusion and processing ................................ 21
9. UDP API Extensions ............................................. 23
10. Whose options are these? .................................... 23
11. UDP options LITE option vs. UDP-Lite ....................... 24
12. Interactions with Legacy Devices ............................. 25
13. Options in a Stateless, Unreliable Transport Protocol .... 25
14. UDP Option State Caching .................................... 26
15. Updates to RFC 768 ........................................... 26
16. Multicast Considerations ..................................... 26
17. Security Considerations ...................................... 27
18. IANA Considerations .......................................... 27
19. References ................................................... 28
   19.1. Normative References .................................... 28
   19.2. Informative References .................................. 28
20. Acknowledgments .............................................. 30
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 experimental 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", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lowercase uses of these words are not to be interpreted as carrying significance described in RFC 2119.

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. Background

Many protocols include a default header and an area for header options. 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].

These 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 as well. One example of such uses is Substrate Protocol for User Datagrams (SPUD) [Tr16], and this document is intended to provide an out-of-band option area as an alternative to the in-band mechanism currently proposed [Hi15].

Appendix A. Implementation Information...........................31
UDP is one of the most popular protocols that lacks space for options [RFC768]. The UDP header was intended to be a minimal addition to IP, providing only ports and a data checksum for protection. This document experimentally extends UDP to provide a trailer area for options located after the UDP data payload.

4. The UDP Option Area

The UDP transport header includes demultiplexing and service identification (port numbers), a 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 (see also discussion in Section 12).

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:

$$UDP\_Length = IPv4\_Total\_Length - IPv4\_IHL \times 4$$

For IPv6, the IP Payload Length field indicates the datagram 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:

$$UDP\_Length = IPv6\_Payload\_Length - \text{sum(extension header lengths)}$$

In both cases, the space available for the UDP transport protocol data unit is indicated by IP, either completely in the base header (for IPv4) or 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).
In most cases, the IP transport payload and UDP Length point to the same location, indicating that there is no surplus area. It is important to note that this is not a requirement of UDP [RFC768] (discussed further in Section 12). UDP-Lite used the difference in these pointers to indicate the partial coverage of the UDP Checksum, such that the UDP user data, UDP header, and UDP pseudoheader (a subset of the IP header) are covered by the UDP checksum but additional user data in the surplus area is not covered [RFC3828]. This document uses the surplus area for UDP transport options.

The UDP option area is thus defined as the location between the end of the UDP payload and the end of the IP datagram as a trailing options area. This area can occur 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 offset".

UDP options are defined using a TLV (type, length, and optional value) syntax similar to that of TCP [RFC793]. They are typically a minimum of two bytes in length as shown in Figure 4, excepting only the one byte options "No Operation" (NOP) and "End of Options List" (EOL) described below.

```
+--------+--------+
| Kind   | Length  |
+--------+--------+
```

**Figure 4 UDP option default format**

**>>** UDP options MAY occur at any UDP length offset.

**>>** The UDP length MUST be at least as large as the UDP header (8) and no larger than the IP transport payload. Values outside this range MUST be silently discarded as invalid and logged where rate-limiting permits.

Others have considered using values of the UDP Length that is larger than the IP transport payload as an additional type of signal. Using
a value smaller than the IP transport payload is expected to be backward compatible with existing UDP implementations, i.e., to deliver the UDP Length of user data to the application and silently ignore the additional surplus area data. Using a value larger than the IP transport payload would either be considered malformed (and be silently dropped) or could cause buffer overruns, and so is not considered silently and safely backward compatible. Its use is thus out of scope for the extension described in this document.

>> UDP options MUST be interpreted in the order in which they occur in the UDP option area.

5. UDP Options

The following UDP options are currently defined:

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0*</td>
<td>-</td>
<td>End of Options List (EOL)</td>
</tr>
<tr>
<td>1*</td>
<td>-</td>
<td>No operation (NOP)</td>
</tr>
<tr>
<td>2*</td>
<td>2</td>
<td>Option checksum (OCS)</td>
</tr>
<tr>
<td>3*</td>
<td>4</td>
<td>Alternate checksum (ACS)</td>
</tr>
<tr>
<td>4*</td>
<td>4</td>
<td>Lite (LITE)</td>
</tr>
<tr>
<td>5*</td>
<td>4</td>
<td>Maximum segment size (MSS)</td>
</tr>
<tr>
<td>6*</td>
<td>8/10</td>
<td>Fragmentation (FRAG)</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>Timestamps (TIME)</td>
</tr>
<tr>
<td>8</td>
<td>(varies)</td>
<td>Authentication and Encryption (AE)</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>Request (REQ)</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>Response (RES)</td>
</tr>
<tr>
<td>11-126</td>
<td>(varies)</td>
<td>UNASSIGNED (assignable by IANA)</td>
</tr>
<tr>
<td>127-253</td>
<td></td>
<td>RESERVED</td>
</tr>
<tr>
<td>254</td>
<td>N(&gt;=4)</td>
<td>RFC 3692-style experiments (EXP)</td>
</tr>
<tr>
<td>255</td>
<td></td>
<td>RESERVED</td>
</tr>
</tbody>
</table>

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 "*": EOL, NOP, OCS, ACS, LITE, FRAG, and MSS. This includes both recognizing and being able to generate these options if configured to do so.

All other options (without a "*") MAY be implemented, and their use SHOULD be determined either out-of-band or negotiated.

Receivers MUST silently ignore unknown options. That includes options whose length does not indicate the specified value.
Except for NOP, each option SHOULD NOT occur more than once in a single UDP datagram. If a non-NOP option occurs more than once, a receiver MUST interpret only the first instance of that option and MUST ignore all others.

Only the OCS and AE options depend on the contents of the option area. AE is always computed as if the AE hash and OCS checksum are zero; OCS is always computed as if the OCS checksum is zero and after the AE hash has been computed. Future options MUST NOT be defined as having a value dependent on the contents of the option area. Otherwise, interactions between those values, OCS, and AE could be unpredictable.

Receivers cannot treat unexpected option lengths as invalid, as this would unnecessarily limit future revision of options (e.g., defining a new ACS that is defined by having a different length).

Option lengths MUST NOT exceed the IP length of the packet. If this occurs, the packet MUST be treated as malformed and dropped, and the event MAY be logged for diagnostics (logging SHOULD be rate limited).

Required options MUST come before other options. Each required option MUST NOT occur more than once (if they are repeated in a received segment, all except the first MUST be silently ignored).

The requirement that required options come before others is intended to allow for endpoints to implement DOS protection, as discussed further in Section 17.

5.1. End of Options List (EOL)

The End of Options List (EOL) option indicates that there are no more options. It is used to indicate the end of the list of options without needing to pad the options to fill all available option space.

```
+--------+
| Kind=0  |
+--------+
```

Figure 5 UDP EOL option format

When the UDP options do not consume the entire option area, the last non-NOP option SHOULD be EOL (vs. filling the entire option area with NOP values).
All bytes after EOL MUST be ignored by UDP option processing. As a result, there can only ever be one EOL option (even if other bytes were zero, they are ignored).

5.2. No Operation (NOP)

The No Operation (NOP) option is a one byte placeholder, intended to be used as padding, e.g., to align multi-byte options along 16-bit or 32-bit boundaries.

If options longer than one byte are used, NOP options SHOULD be used at the beginning of the UDP options area to achieve alignment as would be more efficient for active (i.e., non-NOP) options.

Segments SHOULD NOT use more than three consecutive NOPs. NOPs are intended to assist with alignment, not other padding or fill.

[NOTE: Tom Herbert suggested we declare "more than 3 consecutive NOPs" a fatal error to reduce the potential of using NOPs as a DOS attack, but IMO there are other equivalent ways (e.g., using RESERVED or other UNASSIGNED values) and the "no more than 3" creates its own DOS vulnerability]

5.3. Option Checksum (OCS)

The Option Checksum (OCS) is conventional Internet checksum that covers all of the UDP options. The primary purpose of OCS is to detect non-standard (i.e., non-option) uses of the options area.

OCS is calculated by computing the Internet checksum options area [RFC1071]. OCS protects the option area from errors in a similar way that the UDP checksum protects the UDP user data (when not zero).
>> When present, the option checksum SHOULD occur as early as possible, preceded by only NOP options for alignment and the LITE option if present.

>> The OCS checksum MUST be half-word coordinated with the start of the UDP options area.

This coordination is accomplished by computing the Internet checksum over the UDP options area (including EOL, if present) and then adjusting the result before storing it into the OCS checksum field. If that field is aligned to the start of the options area, then the checksum is inserted as-is, otherwise the checksum bytes are swapped before inserting them into the field.

The adjustment above helps enable that OCS, together with the other options, result in an overall zero ones-complement sum. This feature is intended to potentially help the UDP options traverse devices that incorrectly attempt to checksum the surplus area (as originally proposed as the Checksum Compensation Option, i.e., CCO [Fa18]). Note that this incorrect checksum traversal feature is defeated by the use of LITE, whether alone or with FRAG, because the LITE area is deliberately not covered by OCS. It also is defeated by the use of a zero UDP checksum (i.e., UDP checksum disabled).

OCS covers the UDP option area, including the Lite option (but not LITE data area) as formatted before swapping (or relocation) for transmission (or, equivalently, after the swap/relocation after reception), as the LITE option would occur at the beginning of the original (prior to rearrangement for transmission) or restored (after rearrangement upon reception) UDP option area.

>> If OCS 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 OCS fails, unless the endpoints have negotiated otherwise for this segment’s socket pair.

As a reminder, use of the UDP checksum is optional. When not used, i.e., when the field is zero, that checksum is assumed to be "correct" for the purpose of accepting UDP packets at a receiver.

OCS is intended to check for accidental errors, not for attacks.
5.4. Alternate Checksum (ACS)

The Alternate Checksum (ACS) provides a stronger alternative to the checksum in the UDP header, using a 16-bit CRC of the conventional UDP payload only (excluding the IP pseudoheader, UDP header, and UDP options, and not include the LITE area). Because it does not include the IP pseudoheader or UDP header, it need not be updated by NATs when IP addresses or UDP ports are rewritten. Its purpose is to detect errors that the UDP checksum, when used, might not detect.

CRC-CCITT (polynomial $x^{16} + x^{12} + x^{5} + 1$) has been chosen because of its ubiquity and use in other packet protocols, such as X.25, HDLC, and Bluetooth. The option contains FCS-16 as defined in Appendix C of [RFC1662], except that it is not inverted in the final step and that it is stored in the ACS option in network byte order.

```
+--------+--------+--------+--------+
| Kind=3 | Len=4  |     CRC16sum    |
+--------+--------+--------+--------+
```

Figure 8 UDP ACS option format

When present, the ACS 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 ACS is not used; instead, the option would simply not be included if that were the desired effect).

ACS does not protect the UDP pseudoheader; only the current UDP checksum provides that protection (when used). ACS 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, ACS does not cover the pseudoheader.

5.5. Lite (LITE)

The Lite option (LITE) is intended to provide equivalent capability to the UDP Lite transport protocol [RFC3828]. UDP Lite allows the UDP checksum to cover only a prefix of the UDP data payload, to protect critical information (e.g., application headers) but allow potentially erroneous data to be passed to the user. This feature helps protect application headers but allows for application data errors. Some applications are impacted more by a lack of data than errors in data, e.g., voice and video.

>> When LITE is active, it MUST come first in the UDP options list.
LITE is intended to support the same API as for UDP Lite to allow applications to send and receive data that has a marker indicating the portion protected by the UDP checksum and the portion not protected by the UDP checksum.

LITE includes a 2-byte offset that indicates the length of the portion of the UDP data that is not covered by the UDP checksum.

```
+----------------+----------------+----------------+------------------+
| Kind=4 | Len=4 | Offset     |
+----------------+----------------+----------------+------------------+
```

Figure 9 UDP LITE option format

At the sender, the option is formed using the following steps:

1. Create a LITE option, ordered as the first UDP option (Figure 10).

2. Calculate the location of the start of the options as an absolute offset from the start of the UDP header and place that length in the last two bytes of the LITE option.

3. If the LITE data area is 4 bytes or longer, swap all four bytes of the LITE option with the first 4 bytes of the LITE data area (Figure 11). If the LITE data area is 0-3 bytes long, slide the LITE option to the front of the LITE data area (i.e., placing the 0-3 bytes of LITE data after the LITE option).

```
+---------+--------------+--------------+------------------+
| UDP Hdr |  user data   |  LITE data   |LITE| other opts  |
+---------+--------------+--------------+------------------+
```

Figure 10   LITE option formation - LITE goes first

```
+---------+--------------+--------------+------------------+
| UDP Hdr |  user data   |  LITE data   |LITE| other opts  |
+---------+--------------+--------------+------------------+
    ^^^^          ^^^^           |
--------------------------
```

Figure 11   Before sending swap LITE option and front of LITE data

Expires September 8, 2019
The resulting packet has the format shown in Figure 12. Note that the UDP length now points to the LITE option, and the LITE option points to the start of the option area.

```
+---------+--------------+----------------+------------------+
| UDP Hdr | user data   |LITE| LITE data |Ldat| other opts |
+---------+--------------+----------------+------------------+
```

Figure 12   Lite option as sent

A legacy endpoint receiving this packet will discard the LITE option and everything that follows, including the lite data and remainder of the UDP options. The UDP checksum will protect only the user data, not the LITE option or lite data.

Receiving endpoints capable of processing UDP options will do the following:

1. Process options as usual. This will start at the LITE option.

2. When the LITE option is encountered, record its location as the start of the LITE data area and (if the LITE offset indicates a LITE data length of at least 4 bytes) swap the four bytes there with the four bytes at the location indicated inside the LITE option, which indicates the start of all of the options, including the LITE one (one past the end of the lite data area). If the LITE offset indicates a LITE data area of 0-3 bytes, then slide the LITE option forward that amount and slide the corresponding bytes after the LITE option to where the LITE option originally began. In either case, this restores the format of the option as it was prior to being sent, as per Figure 10.

3. Continue processing the remainder of the options, which are now in the format shown in Figure 11.

The purpose of this swap (or slide) is to support the equivalent of UDP Lite operation together with other UDP options without requiring the entire LITE data area to be moved after the UDP option area.

5.6. Maximum Segment Size (MSS)

The Maximum Segment Size (MSS, Kind = 3) is a 16-bit indicator of the largest UDP segment that can be received. As with the TCP 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].

+--------+--------+--------+--------+
| Kind=5 | Len=4  | MSS size |
+--------+--------+--------+--------+

Figure 13   UDP MSS option format

The UDP MSS option MAY be used 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 to limit the largest reassembled UDP message,
e.g., when EMTU_R is larger than the required minimums (576 for IPv4
[RFC791] and 1500 for IPv6 [RFC8200]).

5.7. Fragmentation (FRAG)

The Fragmentation option (FRAG) supports UDP fragmentation and
reassemble, which can be used to transfer UDP messages larger than
limited by the IP receive MTU (EMTU_R [RFC1122]). It is typically
used with the UDP MSS option 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.

+--------+--------+--------+--------+
| Kind=6 | Len=8  | Frag. Offset |
+--------+--------+--------+--------+
| Identification |
+--------+--------+--------+--------+

Figure 14   UDP non-terminal FRAG option format

The FRAG option also lacks a "more" bit, zeroed for the terminal
fragment of a set. This is possible because the terminal FRAG option
is indicated as a longer, 10-byte variant, which includes an
Internet checksum over the entire reassembled UDP payload (omitting
the IP pseudoheader and UDP header, as well as UDP options), as
shown in Figure 15.

Touch                 Expires September 8, 2019               [Page 14]
The reassembly checksum SHOULD be used, but MAY be unused in the same situations when the UDP checksum is unused (e.g., for transit tunnels or applications that have their own integrity checks [RFC8200]), and by the same mechanism (set the field to 0x0000).

```
+--------+--------+--------+--------+
| Kind=6 | Len=10 | Frag. Offset |
+--------+--------+--------+--------+
| Identification |
+----------------+--------+--------+
| Checksum |
+--------+--------+
```

Figure 15   UDP terminal FRAG option format

During fragmentation, the UDP header checksum of each fragment needs to be recomputed based on each datagram’s pseudoheader.

After reassembly is complete and validated using the checksum of the terminal FRAG option, the UDP header checksum of the resulting datagram needs to be recomputed based on the datagram’s pseudoheader.

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 Len field indicates whether there are more fragments (Len=8) 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.

FRAG needs to be used with extreme care because it will present incorrect datagram boundaries to a legacy receiver, unless encoded as LITE data (see Section 5.8).

A host SHOULD indicate FRAG support by transmitting an unfragmented datagram using the Fragmentation option (e.g., with Offset zero and length 12, i.e., including the checksum area), except when encoded as LITE.
A host MUST NOT transmit a UDP fragment before receiving recent confirmation from the remote host, except when FRAG is encoded as LITE.

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. Implementers are advised to limit the space available for UDP reassembly.

UDP reassembly space SHOULD be limited to reduce the impact of DOS attacks on resource use.

UDP reassembly space limits SHOULD NOT be implemented as an aggregate, 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 options.

Any additional UDP options would follow the FRAG option in the final fragment, and would be included in the reassembled packet. Processing of those options would commence after reassembly.

UDP options MUST NOT follow the FRAG header in non-terminal fragments. Any data following the FRAG header in non-terminal fragments MUST be silently dropped. All other options that apply to a reassembled packet MUST follow the FRAG header in the terminal fragment.

5.8. Coupling FRAG with LITE

FRAG can be coupled with LITE to avoid impacting legacy receivers. Each fragment is sent as LITE un-checksummed data, where each UDP packet contains no legacy-compatible data. Legacy receivers interpret these as zero-length payload 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 packet itself were a signal. The header of such a packet would appear as shown in Figure 16 and Figure 17.
When a packet is reassembled, it appears as a complete LITE data region. The UDP header of the reassembled packet is adjusted accordingly, so that the reassembled region now appears as conventional UDP user data, and processing of the UDP options continues, as with the non-LITE FRAG variant.

5.9. Timestamps (TIME)

The UDP Timestamp option (TIME) exchanges two four-byte 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].

TS Value (TSval) and TS Echo Reply (TSecr) are used in a similar manner to the TCP TS option [RFC7323]. On transmitted segments 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.

TIME SHOULD make this RTT estimate available to the user application.

UDP timestamps are modeled after TCP timestamps and have similar expectations. In particular, they are expected to be:

- Values are monotonic and non-decreasing
- Values should increase according to a typical 'tick' time
- A request is defined as "reply=0" and a reply is defined as both fields being non-zero.
- A receiver should always respond to a request with the highest TSval received, which is not necessarily the most recently received.

5.10. Authentication and Encryption (AE)

The Authentication and Encryption option (AE) is intended to allow UDP to provide a similar type of authentication as the TCP Authentication Option (TCP-AO) [RFC5925]. It uses the same format as specified for TCP-AO, except that it uses a Kind of 8. AE supports NAT traversal in a similar manner as TCP-AO [RFC6978]. AE can also be extended to provide a similar encryption capability as TCP-AO-ENC, in a similar manner [To18ao].

```
+--------+--------+--------+--------+
| Kind=8 |  Len   |     Digest...   |
|--------+--------+--------+--------+
|        Digest (con’t)...        |
+--------+--------+--------+--------+
```

Figure 19   UDP AE option format

Like TCP-AO, AE 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 AE’s Key Derivation Function (KDF) uses zeroes.
as the value for both ISNs. This means that the AE reuses keys when socket pairs are reused, unlike TCP-AO.

AE can be configured to either include or exclude UDP options, the same way as can TCP-AO. When UDP options are covered, the OCS option area checksum and AE hash areas are zeroed before computing the AE 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 option area contents. This is why such dependencies are not permitted except as defined for OCS and UDP-AE.

Similar to TCP-AO-NAT, AE can be configured to support NAT traversal, excluding one or both of the UDP ports [RFC6978].

6. Echo request (REQ) and echo response (RES)

The echo request (REQ, kind=9) and echo response (RES, kind=10) options provide a means for UDP options to be used to provide packet-level acknowledgements. Their use is described as part of the UDP variant of packetization layer path MTU discovery (PLPMTUD) [Fa19]. The options both have the format indicated in Figure 20.

```
+--------+--------+------------------+
|  Kind  | Len=6  |      nonce       |
+--------+--------+------------------+

Figure 20   UDP REQ and RES options format
```

6.1. Experimental (EXP)

The Experimental option (EXP) is reserved for experiments [RFC3692]. It uses a Kind value of 254. 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].
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]).

7. 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. For example, LITE needs to come first if present; new options cannot declare that they need to precede it. The following is a summary of rules for new options and their rationales:

>> New options MUST NOT depend on option space content. Only OCS and AE depend on the content of the options themselves and their order is fixed (on transmission, AE is computed first using a zero-checksum OCS if present, and OCS is computed last before transmission, over the entire option area, including AE).

>> 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; areas that need to remain unmodified include the IP header, IP options, the UDP body, the UDP option area (i.e., other options), and the post-option area.

>> 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:

- >> LITE MUST be first, if present, and MUST be processed when encountered (e.g., even before security options).

- >> LITE is the only option that modifies the UDP body or option areas.
OCS SHOULD be the first option, except in the presence of LITE, in which case it SHOULD be the first option after LITE.

8. 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 mandatory options MUST be processed by receivers, if present (presuming UDP options are supported at that receiver).

>> Non-mandatory options MAY be ignored by receivers, if present, based on API settings.

>> All options MUST be processed by receivers in the order encountered in the options list.

The basic premise is that 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 mandatory options.

Upon receipt, the receiver checks various properties of the UDP packet and its options to decide whether to accept or drop the packet and whether to accept or ignore some its options as follows (in order):
if the UDP checksum fails then
  silently drop (per RFC1122)
if the UDP checksum passes then
  if OCS is present and fails then
    deliver the UDP payload but ignore all options
    (this is required to emulate legacy behavior)
  if OCS is present and passes then
    deliver the UDP payload after parsing
    and processing the rest of the options

(for other options 'OPT' when encountered in sequence):
if both sender and receiver choose to use OPT then
  if OPT passes then
    deliver the UDP payload after parsing
    and processing the rest of the options
  if OPT fails then
    silently drop the packet
  if OPT is not present when received then
    silently drop the packet
if the sender includes OPT
  and the receiver does not indicate OPT is required then
  the receiver accepts all UDP payloads that pass
  the UDP checksum and indicate for each packet
  whether OPT succeeded, but never drop when OPT fails

I.e., all options other than OCS 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 (by API configuration)
should the receiver require it being present and correct.

I.e., for all options other than OCS:
  o if the option is not required by the receiver, then packets
    missing the option are accepted.
  o if the option is required and missing or incorrectly formed,
    silently drop the packet.
  o if the packet is accepted (either because the option is not
    required or because it was required and correct), then pass the
    option with the 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).
9. UDP API Extensions

UDP currently specifies an application programmer interface (API), summarized as follows (with Unix-style command as an example) [RFC768]:

- Method to create new receive ports
  - E.g., bind(handle, recvaddr(optional), recvport)

- Receive, which returns data octets, source port, and source address
  - E.g., recvfrom(handle, srcaddr, srcport, data)

- Send, which specifies data, source and destination addresses, and source and destination ports
  - E.g., sendto(handle, destaddr, destport, data)

This API is extended to support options as follows:

- Extend the method to create receive ports to include receive options that are required. Datagrams not containing these required options MUST be silently dropped and MAY be logged.

- Extend the receive function to indicate the options and their parameters as received with the corresponding received datagram.

- Extend the send function to indicate the options to be added to the corresponding sent datagram.

Examples of API instances for Linux and FreeBSD are provided in Appendix A, to encourage uniform cross-platform implementations.

10. Whose options are these?

UDP options are indicated in an 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 datagrams 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 UDP option area, except as provided by the options selected (e.g., OCS or AE).

11. UDP options LITE option vs. UDP-Lite

UDP-Lite provides partial checksum coverage, so that 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.

The LITE UDP options option supports a similar service to UDP-Lite. The main difference is that UDP-Lite provides the un-checksummed user data to the application by default, whereas the LITE UDP option can safely provide that service only between endpoints that negotiate that capability in advance. An endpoint that does not implement UDP options would silently discard this non-checksummed user data, along with the UDP options as well.

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 separate 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.

12. 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 UDP 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 payload.
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.

(TBD: test with UDP checksum offload and UDP fragmentation offload)

13. 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.
>> UDP options MUST allow for silent failure on first receipt.

>> UDP options that rely on soft-state exchange MUST allow for message reordering and loss.

>> A UDP option MUST be silently optional until confirmed by exchange with an endpoint.

The above requirements prevent using any option that cannot be safely ignored unless that capability has been negotiated with an endpoint in advance for a socket pair. Legacy systems would need to be able to interpret the transport payload fragments as individual transport datagrams.

14. 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 [RFC2140][To19cb]. 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.

[TBD: extend this section to indicate which options MAY vs. MUST NOT be shared and how, e.g., along the lines of To19cb]

15. Updates to RFC 768

This document updates RFC 768 as follows:

- This document defines the meaning of the IP payload area beyond the UDP length but within the IP length.

- This document extends the UDP API to support the use of options.

16. 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.
17. Security Considerations

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 payload 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 Extension option (Section 5.10). 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. Implementations concerned with the potential for this vulnerability MAY implement only the required options and MAY also limit processing of TLVs. 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 three in a row), this limits their DOS impact. Note that when a packet’s options cannot be processed, it MUST be discarded; the packet and its options should always share the same fate.

18. 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 5. Additional values in this registry are to be assigned from the UNASSIGNED values Section 5 in 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 7.

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]. 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 7.

19. References

19.1. Normative References


19.2. Informative References


20. Acknowledgments

This work benefitted from feedback from Bob Briscoe, Ken Calvert, Ted Faber, Gorry Fairhurst, C. M. Heard (including the FRAG/LITE combination), Tom Herbert, and Mark Smith, 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
Appendix A. Implementation Information

The following information is provided to encourage interoperable API implementations.

System-level variables (sysctl):

<table>
<thead>
<tr>
<th>Name</th>
<th>default</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>net.ipv4.udp_opt</td>
<td>0</td>
<td>UDP options available</td>
</tr>
<tr>
<td>net.ipv4.udp_opt_ocs</td>
<td>1</td>
<td>Default include OCS</td>
</tr>
<tr>
<td>net.ipv4.udp_opt_acs</td>
<td>0</td>
<td>Default include ACS</td>
</tr>
<tr>
<td>net.ipv4.udp_opt_lite</td>
<td>0</td>
<td>Default include LITE</td>
</tr>
<tr>
<td>net.ipv4.udp_opt_mss</td>
<td>0</td>
<td>Default include MSS</td>
</tr>
<tr>
<td>net.ipv4.udp_opt_time</td>
<td>0</td>
<td>Default include TIME</td>
</tr>
<tr>
<td>net.ipv4.udp_opt_frag</td>
<td>0</td>
<td>Default include FRAG</td>
</tr>
<tr>
<td>net.ipv4.udp_opt_ae</td>
<td>0</td>
<td>Default include AE</td>
</tr>
</tbody>
</table>

Socket options (sockopt), cached for outgoing datagrams:

<table>
<thead>
<tr>
<th>Name</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP_OPT</td>
<td>Enable UDP options (at all)</td>
</tr>
<tr>
<td>UDP_OCT_OCS</td>
<td>Enable UDP OCS option</td>
</tr>
<tr>
<td>UDP_OCT_ACS</td>
<td>Enable UDP ACS option</td>
</tr>
<tr>
<td>UDP_OCT_LITE</td>
<td>Enable UDP LITE option</td>
</tr>
<tr>
<td>UDP_OCT_MSS</td>
<td>Enable UDP MSS option</td>
</tr>
<tr>
<td>UDP_OCT_TIME</td>
<td>Enable UDP TIME option</td>
</tr>
<tr>
<td>UDP_OCT_FRAG</td>
<td>Enable UDP FRAG option</td>
</tr>
<tr>
<td>UDP_OCT_AE</td>
<td>Enable UDP AE option</td>
</tr>
</tbody>
</table>

Send/sendto parameters:

(TBD - currently using cached parameters)

Connection parameters (per-socketpair cached state, part UCB):

<table>
<thead>
<tr>
<th>Name</th>
<th>Initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>opts_enabled</td>
<td>net.ipv4.udp_opt</td>
</tr>
<tr>
<td>ocs_enabled</td>
<td>net.ipv4.udp_opt_ocs</td>
</tr>
</tbody>
</table>

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):

<table>
<thead>
<tr>
<th>Name</th>
<th>default</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>net.ipv4.udp_opt_junk</td>
<td>0</td>
<td>Default use of junk</td>
</tr>
</tbody>
</table>

Socket options (sockopt):

<table>
<thead>
<tr>
<th>Name</th>
<th>params</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP_JUNK</td>
<td>-</td>
<td>Enable UDP junk option</td>
</tr>
<tr>
<td>UDP_JUNK_VAL</td>
<td>fillval</td>
<td>Value to use as junk fill</td>
</tr>
<tr>
<td>UDP_JUNK_LEN</td>
<td>length</td>
<td>Length of junk payload in bytes</td>
</tr>
</tbody>
</table>

Connection parameters (per-socketpair cached state, part UCB):

<table>
<thead>
<tr>
<th>Name</th>
<th>Initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>junk_enabled</td>
<td>net.ipv4.udp_opt_junk</td>
</tr>
<tr>
<td>junk_value</td>
<td>0xABCD</td>
</tr>
<tr>
<td>junk_len</td>
<td>4</td>
</tr>
</tbody>
</table>
The Some Congestion Experienced ECN Codepoint
draft-morton-.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 4 January 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. Terminology ........................................... 2
2. Introduction ........................................... 2
3. Background ............................................. 2
4. Some Congestion Experienced ......................... 3
5. Examples of use ........................................ 5
   5.1. Codel-type AQMs ................................... 5
   5.2. RED-type AQMs (including PIE) ..................... 6
   5.3. TCP ............................................... 6
   5.4. Other .............................................. 7
6. Related Work .......................................... 7
7. IANA Considerations ................................... 7
8. Security Considerations ............................... 7
9. Acknowledgements ..................................... 7
10. Normative References ................................. 8
11. Informative References ............................... 8
Authors’ Addresses ....................................... 9

1. 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] and [RFC8174] when, and only when, they appear in all
capitals, as shown here.

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
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:

<table>
<thead>
<tr>
<th>Binary</th>
<th>Keyword</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Not-ECT (Not ECN-Capable Transport)</td>
<td>[RFC3168]</td>
</tr>
<tr>
<td>01</td>
<td>SCE (Some Congestion Experienced)</td>
<td>[This Internet-draft]</td>
</tr>
<tr>
<td>10</td>
<td>ECT (ECN-Capable Transport)</td>
<td>[RFC3168]</td>
</tr>
<tr>
<td>11</td>
<td>CE (Congestion Experienced)</td>
<td>[RFC3168]</td>
</tr>
</tbody>
</table>
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 with respect to congestion control, 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:

<table>
<thead>
<tr>
<th>Not-ECT</th>
<th>Not-ECT or DROP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECT</td>
<td>ECT or SCE or CE or DROP</td>
</tr>
<tr>
<td>SCE</td>
<td>SCE or CE or DROP</td>
</tr>
<tr>
<td>CE</td>
<td>CE or DROP</td>
</tr>
</tbody>
</table>

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 some mixture of ECT and SCE codepoints indicates that some degree of queuing delay exists at 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 ensure RTT-fair convergence with single-queue SCE AQMs, transports SHOULD stabilise at lower SCE-mark ratios for higher BDPs, and MAY reduce their response to CE marks IFF they are responding to SCE signals received at around the same time (eg. within 1-2 RTTs) in the same flow.
To maximise the benefit of SCE, middleboxes SHOULD produce SCE markings sooner than they produce CE markings, when the level of congestion increases. Single-queue AQMs MAY choose to prefer fairness between SCE and non-SCE flows, by instead beginning SCE marking at the lowest threshold of CE marking.

5. Examples of use

5.1. Codel-type AQMs

A simple and natural way to implement SCE in a Codel-type AQM is to mark all ECT packets as SCE if they are over half the Codel target sojourn time, and not marked CE by Codel itself. This threshold function does not necessarily produce the best performance, but is very easy to implement and provides useful information to SCE-aware flows, often sufficient to avoid receiving CE marks whilst still efficiently using available capacity.

For a more sophisticated approach avoiding even small-scale oscillation, a stochastic ramp function may be implemented with 100% marking at the Codel target, falling to 0% marking at or above zero sojourn time. The lower point of the ramp should be chosen so that SCE is not accidentally signalled due to CPU scheduling latencies or serialisation delays of single packets. Absent rigorous analysis of these factors, setting the lower limit at half the Codel target should be safe in many cases.

The default configuration of Codel is 100ms interval, 5ms target. A typical ramp function for these parameters might cease marking below 2.5ms sojourn time, increase marking probability linearly to 100% at 5ms, and mark at 100% for sojourn times above 5ms (in which CE marking is also possible).

In single-queue AQMs, the above strategy will result in SCE flows yielding to pressure from non-SCE flows, since CE marks do not occur until SCE marking has reached 100%. A balance between smooth SCE behaviour and fairness versus non-SCE traffic can be found by having the marking ramp cross the Codel target at some lower SCE marking rate, perhaps even 0%. A two-part ramp, reaching 1/sqrt(X) at the Codel target (for some chosen X, a cwnd at which the crossover between smoothness and fairness occurs) and ramping up more steeply thereafter, has been implemented successfully for experimentation.

Flow-isolating AQMs should avoid signalling SCE to flows classified as "sparse" in order to encourage the fastest possible convergence to the fair share.
For the avoidance of doubt, a decision to mark CE or to drop a packet always takes precedence over SCE marking.

5.2. RED-type AQMs (including PIE)

There are several reasonable methods of producing SCE signals in a RED-type AQM.

The simplest would be a threshold function, giving a hard boundary in queue depth between 0% and 100% SCE marking. This could be a sensible option for limited hardware implementations. The threshold should be set below the point at which a growing queue might trigger CE marking or packet drops.

Another option would be to implement a second marking probability function, occupying a queue-depth space just below that occupied by the main marking probability function. This should be arranged so that high marking rates (ideally 100%) are achieved at or before the point at which CE marking or packet drops begin.

For PIE specifically, a second marking probability function could be added with the same parameters as the main marking probability function, except for a lower QDELAY_REF value. This would result in the SCE marking probability remaining strictly higher than the CE marking probability for ECT flows.

5.3. TCP

Some mechanism should be defined to feed back SCE signals to the sender explicitly. Details of this are left to future I-Ds covering TCP in detail; use could be made of the redundant NS bit in the TCP header, which was formerly associated with ECT(1) in the Nonce Sum specification.

Alternatively, SCE can potentially be handled entirely by the receiver, and thus be entirely independent of any of the dozens of [RFC3168] compliant congestion control algorithms on the sender side. This would be done by adjusting the Receive Window, which has been a standard part of TCP from its inception. This alternative therefore requires the minimum amount of protocol changes on the wire.

The recommended response to each single segment marked with SCE is to reduce cwnd by an amortised 1/sqrt(cwnd) segments. Other responses, such as the 1/cwnd from DCTCP, are also acceptable but may perform less well.
5.4. Other

New transports under development, such as QUIC, may implement a fine-grained signal back to the sender based on SCE. QUIC itself appears to have this sort of feedback already (counting ECT(0), ECT(1) and CE packets received), and the data should be made available for congestion control.

6. Related Work

[RFC8087][RFC7567][RFC7928][RFC8290][RFC8289][RFC8033][RFC8034]

7. IANA Considerations

There are no IANA considerations.

8. Security Considerations

An adversary could inappropriately set SCE marks at middleboxes he controls to slow down SCE-aware flows, eventually reaching a minimum congestion window. However, the same threat already exists with respect to inappropriately setting CE marks on normal ECN flows, and this would have a greater impact per mark. Therefore no new threat is exposed by SCE in practice.

An adversary could also simply ignore SCE marks at the receiver, or ignore SCE information fed back from the receiver to the sender, in an attempt to gain some advantage in throughput. Again, the same could be said about ignoring CE marks, so no truly new threat is exposed. Additionally, correctly implemented SCE detection may actually improve long-term goodput compared to ignoring SCE.

An adversary could erase congestion information by converting SCE marks to ECT or Not-ECT codepoints, thus hiding it from the receiver. This has equivalent effects to ignoring SCE signals at the receiver. An identical threat already exists for erasing congestion information from CE marked packets, and may be mitigated by AQMs switching to dropping packets from flows observed to be non-responsive to CE.

9. Acknowledgements

Thanks to Dave Taht for his contributions to the SCE effort, and his work on writing the original draft-morton-taht-sce-00 that was submitted for IETF/104 on which this draft is based.

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. Normative References


11. Informative References


[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC


Authors’ Addresses

Jonathan Morton
Kokkonranta 21
FI-31520 Pitkajarvi
Finland

Phone: +358 44 927 2377
Email: chromatix99@gmail.com

Rodney W. Grimes (editor)
Redacted
Portland, OR 97217
United States

Email: rgrimes@freebsd.org
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.

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 8, 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
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.

```
0                   1                   2                   3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 0    | Res |R|I|U|B|E|           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              TSN                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Stream Identifier      |     Stream Sequence Number    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Payload Protocol Identifier                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                           User Data                           /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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.
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.
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:
Table 1

The parameter format is the following:

| Parameter Type | Parameter Name                        |
|----------------+--------------------------------------|
| 0x8100         | Retransmit Bit Supported (RBIT-SUPPORTED) |

Table 1

The parameter format is the following:

```
+----------------+-------------------------------------------+
| Parameter Type | Parameter Name                            |
|----------------+-------------------------------------------+
| 0x8100         | Retransmit Bit Supported (RBIT-SUPPORTED) |
+----------------+-------------------------------------------+
```

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:

- 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).

- 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.

- 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.

- 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

- 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.

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. 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.

6. Socket API Considerations

This document does not address any changes to the socket API defined in [RFC6458].

Proshin
Expires December 8, 2019
[Page 8]
7. Acknowledgements

TBD

8. 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:

<table>
<thead>
<tr>
<th>Chunk Flag Value</th>
<th>Chunk Flag Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>E bit</td>
<td>[RFC4960]</td>
</tr>
<tr>
<td>0x02</td>
<td>B bit</td>
<td>[RFC4960]</td>
</tr>
<tr>
<td>0x04</td>
<td>U bit</td>
<td>[RFC4960]</td>
</tr>
<tr>
<td>0x08</td>
<td>I bit</td>
<td>[RFC7053]</td>
</tr>
<tr>
<td>0x10</td>
<td>R bit</td>
<td>RFCXXXX</td>
</tr>
<tr>
<td>0x20</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>0x40</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>0x80</td>
<td>Unassigned</td>
<td></td>
</tr>
</tbody>
</table>

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:
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:

<table>
<thead>
<tr>
<th>Chunk Flag Value</th>
<th>Chunk Flag Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>R bit</td>
<td>RFCXXXX</td>
</tr>
<tr>
<td>0x02</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>0x04</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>0x08</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>0x10</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>0x20</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>0x40</td>
<td>Unassigned</td>
<td></td>
</tr>
<tr>
<td>0x80</td>
<td>Unassigned</td>
<td></td>
</tr>
</tbody>
</table>

Table 3

9. Security Considerations

This document does not introduce any additional security considerations in addition to the ones described in [RFC4960] and [RFC8260].

10. References
10.1. Normative References


10.2. Informative References


Author’s Address

Maksim Proshin
Ericsson
Kistavaegen 25
Stockholm 164 80
Sweden

Email: mproshin@tieto.mera.ru
Identifying and Handling Non Queue Building Flows in a Bottleneck Link
draft-white-tsvwg-nqb-02

Abstract

This draft proposes the definition of a standardized DiffServ code point (DSCP) to identify Non-Queue-Building flows (for example: interactive voice and video, gaming, machine to machine applications), along with a Per-Hop-Behavior (PHB) that provides a separate queue for such flows.

The purpose of such a marking scheme is to enable networks to provide and utilize queues that are optimized to provide low latency and low loss for such Non-Queue-Building flows (e.g. shallow buffers, optimized media access parameters, etc.).

This marking scheme and PHB has been developed primarily for use by access network segments, where queuing delays and queuing loss caused by Queue-Building protocols are manifested. In particular, applications to cable broadband links and mobile network radio and core segments are discussed.

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 30, 2019.
1. Introduction

The vast majority of packets that are carried by 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 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 traditional congestion-controlled applications, there are a variety of relatively low data rate applications that do not materially contribute to queueing delay and loss, 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 and queuing them separately in order that both classes can deliver optimal quality of experience for their applications.

A couple of preconditions need to be satisfied before we can move on from the status quo. First, the packets must be efficiently identified so that they can be quickly assigned to the "right" queue. This is especially important with the rising popularity of encrypted and multiplexed transports, which has the potential of making deep inspection infeasible. Second, the signal must be such that malicious or badly configured nodes can’t abuse it.

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. These applications do not make use of network buffers, but nonetheless can be subjected to packet delay and delay variation as a result of sharing a network buffer with those that do make use of them. Many of these applications are negatively affected by excessive packet delay and delay variation. Such applications are ideal candidates to be queued separately from the capacity-seeking applications that are the cause of queue buildup, latency and loss.
These Non-queue-building (NQB) flows are typically UDP flows that 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 or QUIC, with BBR or other TCP congestion controllers.

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 higher-speed links, but QB on lower-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, and why wouldn’t all applications mark their packets as NQB?

As an answer the last question, it is 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, network elements that support differentiating NQB traffic SHOULD support a "queue protection" function that can identify QB flows that are mismarked as NQB, and reclassify those flows/packets to the QB queue. This benefits the reclassified flow by giving it access to a large buffer (and thus lower packet loss rate), and benefits 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-layer 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 (EF) DSCP, the NQB PHB would differ from the Expedited Forwarding PHB in several important ways.

- 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.

- The node supporting the NQB PHB makes no guarantees on latency or data rate for NQB marked flows, but instead aims to provide a bound on queuing delay for as many such marked flows as it can, and shed load when needed.
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 (0b101110) in the NQB queue. The choice of the 0x2A codepoint (0b101010) for NQB would conveniently allow a node to select these two codepoints using a single mask pattern of 0b101x10.

6. End-to-end Support

In contrast to the existing standard DSCPs, which are typically only meaningful within a DiffServ Domain (e.g. an AS), this DSCP would be intended for end-to-end usage across the Internet. Some network operators bleach the Diffserv field on ingress into their network [Custura], and in some cases apply their own DSCP for internal usage. Networks that support the NQB PHB SHOULD preserve the NQB DSCP when forwarding via an interconnect.

7. Relationship to L4S

The dual-queue mechanism described in this draft is intended to be compatible with [I-D.ietf-tsvwg-l4s-arch].

8. Use Cases

8.1. DOCSIS Access Networks

Residential cable 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.

To support the NQB PHB, cable broadband services MUST be configured to provide a separate queue for NQB traffic that shares the service rate shaping configuration with the queue for QB traffic.

8.2. Mobile Networks

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.

To support the NQB PHB, the mobile network MUST be 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 SHOULD be routed through the dedicated low-latency EPS bearer. A packet that has no associated NQB marking SHOULD be routed through the default EPS bearer.

8.3. WiFi Networks

WiFi networking equipment compliant with 802.11e generally supports either four or eight transmit queues and four sets of associated CSMA parameters that are used to enable differentiated media access characteristics. Implementations typically utilize the IP DSCP field to select a transmit queue.

As discussed in [RFC8325], most implementations use a default DSCP to User Priority mapping that utilizes the most significant three bits of the DiffServ Field to select 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").

Systems that utilize [RFC8325], SHOULD map the 0x2A codepoint to UP_6 in the "Voice" Access Category.

9. 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 relative latency-sensitivity and importance), associating packets to priority levels (configuring and maintaining 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. See also [Claffy] for an exhaustive treatment of the argument.

Flow queueing (FQ) 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 some undesirable effects of the Deficit Round Robin (DRR) scheduling. The DRR scheduler enforces that each non-sparse flow gets an equal fraction of link bandwidth, which causes problems with flows with VPNs and other tunnels, exhibits poor behavior with less-aggressive congestion control algorithms, e.g. LEDBAT [RFC6817], and could exhibit poor behavior with RTP Media Congestion Avoidance Techniques (RMCAT) [I-D.ietf-rmcat-cc-requirements]. 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 defined in this document 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][Estan] 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 mice 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.

Other dual-queue approaches have been proposed, including some that pair a shallow buffer with a deep buffer, similar to what is described in this draft. One such design is the "RD" mechanism in [Podlesny] which proposes that applications select either high rate or low delay, with one queue (the high-rate queue) being given a large buffer and a higher scheduling weight, and the other queue (the low-delay queue) being given a short buffer and lower scheduling weight. This approach is somewhat similar to the NQB PHB, in regards to allowing the application to select between a deep buffer and a shallow one, but it places unnecessary restrictions on the scheduling between the two queues, and doesn’t differentiate traffic based on behavior. Further, the approach doesn’t provide any safety valve to prevent malicious or misconfigured flows from causing excessive packet loss in the low delay queue. Similarly, the "Loss-Latency Tradeoff" approach described in [I-D.fossati-tsvwg-lola] posits that applications should choose between a queue that provides low latency and potentially high loss (i.e. a shallow buffer), and one that provides low loss and potentially high latency (i.e. a deep buffer). This approach misses that both queuing latency and queuing loss are primarily byproducts of application sending behavior, and by properly segregating applications, no trade-off needs to be made.

10. Acknowledgements

Thanks to Bob Briscoe, Greg Skinner, Dave Taht, Toke Hoeiland-Joergensen and Luca Muscariello for their review comments.

11. IANA Considerations

This draft proposes the registration of a standardized DSCP = 0x2A to denote Non-Queue-Building behavior.
12. 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.

13. Informative References


Jesup, R. and Z. Sarker, "Congestion Control Requirements for Interactive Real-Time Media", draft-ietf-rmcat-cc-requirements-09 (work in progress), December 2014.


Authors’ Addresses

Greg White
CableLabs
Email: g.white@cablelabs.com

Thomas Fossati
ARM
Email: Thomas.Fossati@arm.com