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

Abstract

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

Status of This Memo

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

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."
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)            |
+-----------------------------------+   8 bytes \  
| UDP like arranged DCCP ext. Header |
|-----------------------------------+   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.
### Figure 2: The extended DCCP Header with Long Sequence Numbers

```
<table>
<thead>
<tr>
<th>Res</th>
<th>Type</th>
<th>X</th>
<th>Reserved</th>
<th>Sequence Number (high bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td>Sequence Number (low bits)</td>
</tr>
</tbody>
</table>
```

Figure 2: The extended DCCP Header with Long Sequence Numbers

\[\text{[RFC4340]}\]

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

```
<table>
<thead>
<tr>
<th>Res</th>
<th>Type</th>
<th>X</th>
<th>Sequence Number (low bits)</th>
</tr>
</thead>
</table>
```

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

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

### 3.3. UDP header

```
| Res | Type | X | Length | Checksum |
```

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 extented 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-DCCP 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


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
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.lhwxz-hybrid-access-network-architecture][I-D.muley-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            |
|  Application  |            |   + - - - - - - - + - - - - - - + |
|    DCCP      |            |     Subflow (DCCP) | Subflow (DCCP) |
|    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                    Address A2                    Address B2
-----------------------------  -----------------------------
                   (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>
<thead>
<tr>
<th>Feature</th>
<th>TCP</th>
<th>DCCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full-Duplex</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Connection-Oriented</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Header option space</td>
<td>40 bytes</td>
<td>&lt; 1008 bytes or PMTU</td>
</tr>
<tr>
<td>Data transfer</td>
<td>reliable</td>
<td>unreliable</td>
</tr>
<tr>
<td>Packet-loss handling</td>
<td>re-transmission</td>
<td>report only</td>
</tr>
<tr>
<td>Ordered data delivery</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Sequence numbers</td>
<td>one per byte</td>
<td>one per PDU</td>
</tr>
<tr>
<td>Flow control</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Congestion control</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>ECN support</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Selective ACK</td>
<td>yes</td>
<td>depends on congestion control</td>
</tr>
<tr>
<td>Fix message boundaries</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Path MTU discovery</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Fragmentation</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>SYN flood protection</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Half-open connections</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

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]
[I-D.muley-network-based-bonding-hybrid-access]
Muley, P., Henderickx, W., Geng, L., Liu, H., Cardullo, L., Newton, J., Seo, S., Draznin, S., and B. Patil,


Authors’ Addresses
A multipath framework for UDP traffic over heterogeneous access networks
draft-amend-tsvwg-multipath-framework-mpd ccp-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:
o IP compatibility: A multiaccess framework shall be able to transport IP packets and not make any assumptions on which transport protocol is encapsulated.

o Support for unreliable traffic: A multiaccess framework should provide support for transporting unreliable traffic, such as QUIC or UDP based flows. Therefore, unreliable transmission should be supported.

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

o Support for flexible congestion control: A multiaccess framework should support flexible congestion control, including the disabling of the congestion control, if the inner traffic is known to be congestion controlled.

o Support for flexible packet scheduling: A multiaccess framework should support different packet scheduling mechanisms, which should be configurable from the control plane. Examples are cheapest path first, or other more sophisticated schedulers.

o Lightweight: A multiaccess framework should be lightweight in computational resources and limit the encapsulation overhead.

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

3. IP compatible multipath framework based on MP-DCCP

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

```
| Device | Middlebox 1 | Middlebox 2 | Device |
+--------+------------+------------+--------+
| Sender | MP-DCCP entry | MP-DCCP exit | Receiver |
+--------+------------+------------+--------+
```

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., Seo, S., Draznin, S., and B. Patil,
"Network based Bonding solution for Hybrid Access", draft-
muley-network-based-bonding-hybrid-access-03 (work in
progress), October 2018.

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

Abstract

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

Status of This Memo

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

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

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

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

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must
The Low Latency Low Loss Scalable throughput (L4S) Internet service [I-D.ietf-tsvwg-l4s-arch] provides a new Internet service that could eventually replace best efforts, but with ultra-low queuing delay and loss. A structure called the Dual-Queue Coupled AQM manages to provide the L4S service alongside a second queue for Classic Internet traffic, but without prejudging the bandwidth allocations between them. L4S is orthogonal to allocation of bandwidth, so it can be complemented by various bandwidth allocation approaches without prejudging which one.
The Differentiated Services (Diffserv) architecture [RFC2475] provides for various service classes, some defined globally, others defined locally per network domain. Certain of these service classes offer low latency and low loss, as well as differentiated allocation of bandwidth.

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

1.1. Terminology

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

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

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

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

Pure L4S: L4S without unresponsive traffic.

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


DualQ: Abbreviation for Dual-Queue Coupled AQM [I-D.ietf-tsvwg-aqm-dualq-coupled], which is not a specific AQM, but a framework for coupling two AQMs in order to provide L4S.
service while doing no harm to 'Classic' traffic from traditional sources.

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

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

1.2. Document Roadmap

(ToDo)

2. Architectural Comparison of L4S and Diffserv

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

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

2.1. Overlaps between L4S and Diffserv

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

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

2.2. Differences between L4S and Diffserv

Bandwidth allocation: L4S is orthogonal to allocation of bandwidth, so it can be complemented by various bandwidth allocation approaches without prejudgeting which one. In contrast, with Diffserv it was never possible to completely separate control of latency and loss from allocation of bandwidth. The only bandwidth-related aspect of L4S is that it ensures that the capacity seeking behaviour of end-systems can scale with increasing flow rate.
Differentiation vs. General improvement: Diffserv concerns give and take of bandwidth, latency and loss between traffic classes. In contrast, the separation of L4S from Classic traffic in separate queues concerns incremental deployment of a general improvement in latency and loss, without taking from the other queue.

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

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

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

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

3. Low Latency Diffserv Classes within a DualQ Bandwidth Pool

The experimental Dual-Queue Coupled AQM [I-D.ietf-tsvwg-agm-dualq-coupled] consists of a pair of queues. One provides a low latency low loss service but both have full access to
the same pool of bandwidth. When Diffserv was defined no mechanism like this was available that could provide low latency without also requiring bandwidth controls. All Diffserv’s mechanisms for low latency and low loss use some form of priority over bandwidth, then apply a bandwidth constraint to prevent the lower priority traffic from being starved.

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

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

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

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

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

Table 1 lists the Diffserv service classes that have been allocated global use Diffserv codepoints (DSCPs) from Pool 1. They are
described in RFC 4594 ([RFC4594] and its updates ([RFC5865] and
[I-D.ietf-tsvwg-le-phb] so far). An operator that only deploys a
DualQ ([I-D.ietf-tsvwg-qaq-dualq-coupled] but not the relevant
Diffserv PHBs could classify those with an ‘L’ in the ‘Coupled Queue’
column (or local use DSCPs with similar characteristics) into its L
queue, irrespective of the setting of the setting of the ECN field.

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

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

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

Notes for Table 1:

(1): Reserved by RFC 2474 ([RFC2474]).

(2): Superficially, CS5 is a candidate for classification into the
L queue irrespective of its ECN field, given application
signalling is bursty but usually lightweight. However, at
least one major equipment vendor uses CS5 by default to
indicate unresponsive broadcast video traffic (to which RFC
4594 allocates CS3).

(3): Voice-Admit ([RFC5865]) could be given priority over Expedited
Forwarding (EF) ([RFC3246]).
The Real-Time Interactive and Broadcast Video service classes (or any equivalent local-use classes) are intended for inelastic traffic. Therefore they would not be expected to mark themselves as ECN-capable. If they did they would be claiming to be elastic and therefore eligible for classification into the L queue (subject to any policing). These classes should not be classified into the L queue on the basis of DSCP alone, because high bandwidth unresponsive traffic with potentially variable rate is not compatible with the L4S service.

High Throughput Data (or any equivalent local-use class) might use the L4S service because of its support for scalable congestion control.

[I-D.ietf-tsvwg-le-phb] updates RFC 4594 to deprecate using CS1 for Lower Effort (LE).

If a packet is marked LE and ECT(1) and the operator has solely provided a DualQ, this recommends that the packet is classified into the L queue. This could result in LE traffic competing for bandwidth with other classes of traffic in the L queue, but at least it should not harm the latency of other traffic. This is because the ECT(1) marking means the source "MUST" use a scalable congestion control [I-D.ietf-tsvwg-ecn-l4s-id], but the LE marking only means it "SHOULD" use an LBE congestion control [I-D.ietf-tsvwg-le-phb].

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

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

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

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

1. to ensure that a particularly demanding application (e.g. a virtual reality session) survives even if excess traffic overloads the remainder of the site’s bandwidth;

2. to give guaranteed low latency to a particular application (e.g. industrial process control), if the statistically assured low latency of the L queue is insufficiently stable;

3. to provide a bandwidth scavenger service that will have no effect on any other applications at the site, but will scavenge any unused bandwidth, for instance to transfer backups or large data sets.

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

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

- either taking a queue within an existing Diffserv hierarchy and splitting it into L4S and Classic queues;

- or building a queuing hierarchy around a pre-existing dual L4S/Classic queue.

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

The three examples represent the three main ways that this queue-like ‘atom’ can be included in a hierarchy of other queues. Without loss
of generality only one other queue complements the DualQ in each case, but it would be straightforward to extend the examples with more queues.

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

4.1. DualQ Complemented by an Assured Bandwidth Service

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

```
  -----------------+--+
 Assured b/w       |  |-----------.
  -----------------+--+            
        \             Weighted 
         \     w\-.scheduler
             \---++---+++ (   )--'
             |   L      .->||---.         /'-'
             | DualQ |  -------/---++   c\.-.    /
             | b/w  <  ----+--
             | pool  |  ----+-------+        Conditional
             |        |   C  |   \   |---'    priority
             |        |   ----+-------+        scheduler
```

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

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

- The outputs of the AQMs in the two queues (L and C) are coupled together so that L4S sources leave enough space for C packets so that all 'standard' flows get roughly equal throughput;

- A scheduler recombines the outputs of the two queues, giving conditional priority to L packets (the condition prevents starvation of the C queue if any L traffic misbehaves).

A weighted scheduler, e.g. weighted round robin (WRR), is used to combine the outputs of the assured bandwidth queue and the DualQ. It is configured with weight \( w \) for the assured bandwidth queue. Then, packets requesting assured bandwidth will have priority access to fraction \( w \) of the link capacity. However, whenever the assured
bandwidth queue is idle or under-utilized, the DualQ can borrow the balance of the bandwidth. Likewise the assured bandwidth queue can borrow more than fraction $w$ if the DualQ under-utilizes its remaining share.

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

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

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

---

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

---

The symmetry of Figure 2 reveals that both DualQs actually have assured bandwidth. Nonetheless, the label ‘Assured bandwidth’ is only really meaningful from a per-application perspective if the
traffic classified into that DualQ is limited to a small number of application sessions at any one time.

4.2. DualQ Complemented by a Guaranteed Low Latency Service

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

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

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

In a traditional Diffserv architecture, the token bucket would be deployed at the ingress network edge, to limit traffic at each entry point. Alternatively, the token bucket could be deployed directly in front of the queue, where it would only limit the total traffic from
all entry points to the network. For an access link into a network, these two alternative would amount to the same thing.

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

4.3. DualQ Complemented by a Scavenger Service

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

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

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

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

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

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

```
+------------++
|  L1         |   |
|  L2         |   |
|  L3    .->  |-----/       |
\  b/w  <       ( Coupling    (   )--->
|   C  | \
\  ----+-------+        scheduler

Figure 5: Coupling the Classic AQM to Multiple L4S AQMs
```

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

6. Applicability of Coupled AQM to Global Diffserv PHBs

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

In this draft so far, various queuing structures have been described in terms of the way they separate bandwidth and latency. Operators with existing Diffserv deployments may put the question the other way round and ask whether the DualQ Coupled AQM can be used to isolate low latency traffic within the bandwidth allocated to one of the standardized Diffserv PHBs. For instance:
Bandwidth has been allocated to Network Control traffic, but some BGP speakers have been upgraded to a low latency Scalable TCP while others still use Classic TCP. However it’s not possible to predict how much bandwidth one or the other needs at any one time. So it would be useful to isolate the low latency BGP and all the control signalling from the delay caused by the legacy BGP speaker, without having to decide how to carve up the Network Control bandwidth.

Bandwidth has been allocated to Assured Forwarding (AF) traffic but it all shares the same WRED queue and therefore all suffers the same delay. So it would be useful to isolate the AF traffic that supports low latency congestion control from the rest. However, again, it is not possible to predict how many flows of each type there will be at any one time.

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

For convenience, the ‘Mechanism’ column also answers the question "How do PHBs for the global-use DSCPs map to the scenarios in this draft?"
<table>
<thead>
<tr>
<th>Service Class Name</th>
<th>DSCP Name</th>
<th>AQM?</th>
<th>Mechanism</th>
<th>Latency Separation?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Control</td>
<td>CS7</td>
<td>Y&amp;N</td>
<td>Figure 1 or Figure 2</td>
<td>Y</td>
</tr>
<tr>
<td>Network Control</td>
<td>CS6</td>
<td>Y&amp;N</td>
<td>Figure 1 or Figure 2</td>
<td>Y</td>
</tr>
<tr>
<td>OAM</td>
<td>CS2</td>
<td>Y&amp;N</td>
<td>Figure 1 or Figure 2</td>
<td>Y</td>
</tr>
<tr>
<td>Signalling</td>
<td>CS5</td>
<td>N</td>
<td>Figure 1</td>
<td>N</td>
</tr>
<tr>
<td>Telephony</td>
<td>EF</td>
<td>N</td>
<td>Section 4.2</td>
<td>N</td>
</tr>
<tr>
<td>RFC 5865</td>
<td>VA</td>
<td>N</td>
<td>Section 4.2</td>
<td>N</td>
</tr>
<tr>
<td>R-T</td>
<td>CS4</td>
<td>N</td>
<td>Figure 2</td>
<td>Y</td>
</tr>
<tr>
<td>Interactive MM</td>
<td>AF4n</td>
<td>Y</td>
<td>Section 5</td>
<td>Y</td>
</tr>
<tr>
<td>Conference</td>
<td>CS3</td>
<td>N</td>
<td>Figure 2</td>
<td>N</td>
</tr>
<tr>
<td>Video</td>
<td>AF3n</td>
<td>Y</td>
<td>Section 5</td>
<td>Y</td>
</tr>
<tr>
<td>MM Streaming</td>
<td>AF2n</td>
<td>Y</td>
<td>Section 5</td>
<td>Y</td>
</tr>
<tr>
<td>Low Latency Data</td>
<td>AF1n</td>
<td>Y</td>
<td>Section 5</td>
<td>Y</td>
</tr>
<tr>
<td>High Thru'put Data</td>
<td>BE/DF/CS0</td>
<td>Y</td>
<td>Section 3</td>
<td>Y</td>
</tr>
<tr>
<td>Low Priority Data</td>
<td>LE</td>
<td>Y</td>
<td>Section 4.3 n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>

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

7. Best Practice for Classification and Marking

7.1. Never Re-Mark a DSCP

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

7.2. Classification Order
7.2.1. Classification Order: Problem

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

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

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

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

7.2.2. Classification Order: Solutions

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

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

8. Policing and Traffic Conditioning

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

This specification contains no IANA considerations.

10. Security Considerations

{ToDo}

11. Comments Solicited

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

12. Acknowledgements

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

13. References

13.1. Normative References


13.2. Informative References

[I-D.ietf-tsvwg-agm-dualq-coupled]

[I-D.ietf-tsvwg-ecn-l4s-id]

[I-D.ietf-tsvwg-l4s-arch]
[I-D.ietf-tsvwg-le-phb]
Bless, R., "A Lower Effort Per-Hop Behavior (LE PHB)",
draft-ietf-tsvwg-le-phb-06 (work in progress), October 2018.

[RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black,
"Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers",
RFC 2474, DOI 10.17487/RFC2474, December 1998,

[RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z.,
and W. Weiss, "An Architecture for Differentiated Services",
RFC 2475, DOI 10.17487/RFC2475, December 1998,

[RFC2597] Heinanen, J., Baker, F., Weiss, W., and J. Wroclawski,
"Assured Forwarding PHB Group",
RFC 2597, DOI 10.17487/RFC2597, June 1999,

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black,
"The Addition of Explicit Congestion Notification (ECN) to IP",
RFC 3168, DOI 10.17487/RFC3168, September 2001,

[RFC3246] Davie, B., Charny, A., Bennet, J., Benson, K., Le Boudec, J.,
Courtney, W., Davari, S., Firoiu, V., and D. Stiliadis,
"An Expedited Forwarding PHB (Per-Hop Behavior)",
RFC 3246, DOI 10.17487/RFC3246, March 2002,

RFC 3260, DOI 10.17487/RFC3260, April 2002,

[RFC4594] Babiarz, J., Chan, K., and F. Baker,
"Configuration Guidelines for DiffServ Service Classes",
RFC 4594, DOI 10.17487/RFC4594, August 2006,

[RFC5681] Allman, M., Paxson, V., and E. Blanton,
"TCP Congestion Control",
RFC 5681, DOI 10.17487/RFC5681, September 2009,
Appendix A. Open Issues

- The Abstract promises "in which cases one can stand alone without needing the other". That’s in the text, but not highlighted as such.

- Document Roadmap TBA

- Mapping to 802.11 user priorities (or LTE QCIs)? Not strictly within the scope, but perhaps desirable to add, or at least to mention how L4S (experimental) would affect RFC8325 which gives (standards track) mappings between Diffserv and 802.11.

- Identify L4S-friendly rate policers

- Comparison between L4S policing and Diffserv traffic conditioning is TBA

- Security Considerations are TBA (largely depends on the previous bullet)

Author’s Address

Bob Briscoe
CableLabs
UK

Email: ietf@bobbriscoe.net
URI: http://bobbriscoe.net/
Explicit Congestion Notification (ECN) andCongestion Feedback
Using the Network Service Header (NSH)
<draft-eastlake-sfc-nsh-ecn-support-03.txt>

Abstract

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

Status of This Memo

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

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

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

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

The list of current Internet-Drafts can be accessed at
http://www.ietf.org/1id-abstracts.html. The list of Internet-Draft
Shadow Directories can be accessed at
Table of Contents

1. Introduction ............................................ 3
   1.1 NSH Background ...................................... 3
   1.2 ECN Background ...................................... 5
   1.3 Tunnel Congestion Feedback Background ............... 5
   1.4 Conventions Used in This Document .................. 7

2. The NSH ECN Field ....................................... 8

3. ECN Support in the NSH ................................ 10
   3.1 At The Ingress ...................................... 11
   3.2 At Transit Nodes .................................... 12
       3.2.1 At NSH Transit Nodes ............................ 12
       3.2.2 At an SF/Proxy .................................. 13
       3.2.3 At Other Forwarding Nodes ...................... 13
   3.3 At Exit/Egress ...................................... 13
   3.4 Conservation of Packets ............................. 14

4. Tunnel Congestion Feedback Support .................. 15

5. IANA Considerations ................................... 16

6. Security Considerations ................................ 17

7. Acknowledgements ...................................... 17

Normative References .................................. 18
Informative References ................................. 19

Authors’ Addresses ..................................... 20
1. Introduction

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

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

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

1.1 NSH Background

The Service Function Chaining (SFC [RFC7665]) architecture calls for the encapsulation of traffic within a service function chaining domain with a Network Service Header (NSH [RFC8300]) added by the "Classifier" (ingress node) on entry to the domain and the NSH being removed on exit from the domain at the egress node. The NSH is used to control the path of a packet in an SFC domain. The NSH is a natural place, in a domain where traffic is NSH encapsulated, to note congestion, avoiding possible confusion due, for example, to changes in the outer transport header in different parts of the domain.
Figure 1. Example SFC Path Forwarding Nodes

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

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

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

1.3 Tunnel Congestion Feedback Background

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

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

1. (Traffic throttling (policing), where the downstream traffic flowing out of the ingress node is limited to reduce or eliminate congestion.

2. Upstream congestion feedback, where the ingress node sends messages upstream to or towards the ultimate traffic source, a function that can throttle traffic generation/transmission.

3. Traffic re-direction, where the ingress node configures the NSH of some future traffic so that it avoids congested paths. Great care must be taken to avoid (a) significant re-ordering of traffic in flows that it is desirable to keep in order and (b) oscillation/instability in traffic paths due to alternate congestion of previously idle paths and the idling of previously congested paths. For example, it is preferable to classify
traffic into flows of a sufficiently coarse granularity that the flows are long lived and use a stable path per flow sending only newly appearing flows on apparently uncongested paths.

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

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

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

Acronyms:

AQM - Active Queue Management [RFC7567]
CE - Congestion Experienced [RFC3168]
downstream - The direction from ingress to egress
ECN - Explicit Congestion Notification [RFC3168]
ECT - ECN Capable Transport [RFC3168]
IPFIX - IP Flow Information Export [RFC7011]
Not-ECT - Not ECN-Capable Transport [RFC3168]
NSH - Network Service Header [RFC8300]
SF - Service Function [RFC7665]
SFC - Service Function Chaining [RFC7665]
SFF - Service Function Forwarder [RFC7665] - A type of node that forwards based on the NSH.
TLV - Type Length Value
upstream - The direction from egress to ingress
2. The NSH ECN Field

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

```
<table>
<thead>
<tr>
<th>Transport Encapsulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Service Header (NSH)</td>
</tr>
<tr>
<td>Base Header</td>
</tr>
<tr>
<td>Service Path Header</td>
</tr>
<tr>
<td>Metadata (Context Header(s))</td>
</tr>
<tr>
<td>Original Packet / Frame</td>
</tr>
</tbody>
</table>
```

Figure 3. Data Encapsulation with the NSH

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

```
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
+-----------------------------------------------+
|Ver|O|U|    TTL    |   Length  |U|U|U|U|MD Type| Next Protocol |
+-----------------------------------------------+
^ ^
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NSH ECN field</td>
</tr>
<tr>
<td>-------</td>
</tr>
</tbody>
</table>
```

Figure 4: NSH Base Header

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

Table 1 shows the meaning of the code points in the NSH ECN field. These have the same meaning as the ECN field code points in the IPv4 or IPv6 header as defined in [RFC3168].
<table>
<thead>
<tr>
<th>Binary</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Not-ECT</td>
<td>Not ECN-Capable Transport</td>
</tr>
<tr>
<td>01</td>
<td>ECT(1)</td>
<td>ECN-Capable Transport</td>
</tr>
<tr>
<td>10</td>
<td>ECT(0)</td>
<td>ECN-Capable Transport</td>
</tr>
<tr>
<td>11</td>
<td>CE</td>
<td>Congestion Experienced</td>
</tr>
</tbody>
</table>

Table 1. ECN Field Code Points
3. ECN Support in the NSH

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

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

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

ECN Propagation:

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

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

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

ECN Marking:

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

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>Incoming Header (also equal to departing Inner Header)</th>
<th>Departing NSH and Outer Headers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not-ECT</td>
<td>ECT(0)</td>
</tr>
<tr>
<td>ECT(0)</td>
<td>ECT(0)</td>
</tr>
<tr>
<td>ECT(1)</td>
<td>ECT(1)</td>
</tr>
<tr>
<td>CE</td>
<td>CE</td>
</tr>
</tbody>
</table>

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

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

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

```
+-----------------+
|   Outer Header  |
+-----------------+
|       NSH       |
+-----------------+
|   Inner Header  |
+-----------------+
|     Payload     |
+-----------------+
```

Figure 5. Packet in Transit

3.2.1 At NSH Transit Nodes

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

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

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

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

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

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

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

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

3.2.3 At Other Forwarding Nodes

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

3.3 At Exit/Egress

First, any actions are taken based on Congestion Experienced such as forwarding statistics back to the ingress (see Section 4). If the packet being carried inside the NSH is IP, when the NSH is removed the NSH ECN field MUST be combined with IP ECN field as specified in Table 3 that was extracted from [RFC6040]. This requirement applies to all egress nodes for the domain in which NSH is being used to route traffic.
<table>
<thead>
<tr>
<th>Arriving Inner Header</th>
<th>Arriving Outer Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not-ECT</td>
<td>ECT(0)</td>
</tr>
<tr>
<td>ECT(0)</td>
<td>Not-ECT</td>
</tr>
<tr>
<td>ECT(1)</td>
<td>ECT(1)</td>
</tr>
<tr>
<td>CE</td>
<td>CE</td>
</tr>
</tbody>
</table>

Table 3. Exit ECN Fields Merger

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

3.4 Conservation of Packets

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

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>tbd(16-17)</td>
<td>NSH ECN</td>
<td>[this document]</td>
</tr>
</tbody>
</table>
6. Security Considerations

For general NSH security considerations, see [RFC8300].

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

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

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

7. Acknowledgements

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

    Joel Halpern, Tal Mizrahi, Xinpeng Wei
Normative References


Informative References


Authors’ Addresses

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

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

Bob Briscoe
Independent
UK

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

Andrew G. Malis
Huawei Technologies

Email: agmalis@gmail.com
Copyright and IPR Provisions

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

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

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

Status of This Memo

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

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

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

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

Copyright Notice

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

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

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

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

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

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

3. Middlebox Pathologies

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

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

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

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

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

The design of the CCO seeks to increase UDP Options compatibility with middleboxes and other existing network equipment, while at the same time providing error detection on UDP Options area in the same...
way that the UDP Checksum provides an integrity check for the UDP Header and UDP Payload.

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

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

```
+---------+--------+------------+
| Kind=xx | Len=4  | Checksum   |
+---------+--------+------------+
     1 byte  1 byte    2 bytes
```

Figure 2: UDP CCO Option Format

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

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

Figure 3 shows the bytes on which CCO is computed and how, when present, the unpaired byte at the start and/or at end of Options area are included in the sum.
When this CCO checksum and the UDP Options field are covered by the UDP checksum calculation [RFC0768], the resulting UDP checksum value is numerically the same as when the UDP checksum calculation is calculated over only the UDP Payload. That is, the result returned by both checksum computations Figure 1 coincide.

4.1. Calculating the CCO

The CCO can be present at any position within the Options space, the checksum field of the CCO MUST be aligned on a 2 byte boundary. This condition can be achieved by placing a NOP Option before the CCO in the case the number of bytes preceding the CCO (UDP Payload + UDP Options placed before CCO) is odd (see Figure 4).
When calculated in this way, the CCO value is initialized to zero and the checksum is calculated over the UDP Options and the pseudo-header: the one’s complement of the result is then stored in the CCO field.

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

4.2. Validating CCO

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>UDP Length:</th>
<th>Even</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preceding UDP Options:</td>
<td>MSS (kind 5, len 4, val 0x5c0)</td>
</tr>
<tr>
<td>Following UDP Options:</td>
<td>None</td>
</tr>
<tr>
<td>NOP Padding before CCO:</td>
<td>No</td>
</tr>
<tr>
<td>Total UDP Options Length:</td>
<td>8</td>
</tr>
<tr>
<td>UDP Options bytes 0/1:</td>
<td>0504</td>
</tr>
<tr>
<td>UDP Options bytes 2/3:</td>
<td>05c0</td>
</tr>
<tr>
<td>UDP Options bytes 4/5:</td>
<td>cc04</td>
</tr>
<tr>
<td>UDP Options bytes 6/7:</td>
<td>0008</td>
</tr>
</tbody>
</table>

Sum: d6d0

CCO: 292f

Figure 5: Checksum calculations

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

Interaction with other UDP Options

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

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

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

5. Acknowledgements

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

6. IANA Considerations

This memo includes no requests to IANA
7. Security Considerations

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

8. Normative References

[I-D.ietf-tsvwg-udp-options]


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.

Authors' Addresses
Godred Fairhurst  
University of Aberdeen  
School of Engineering  
Fraser Noble Building  
Aberdeen AB24 3UE  
UK  
Email: gorry@erg.abdn.ac.uk

Tom Jones  
University of Aberdeen  
School of Engineering  
Fraser Noble Building  
Aberdeen AB24 3UE  
UK  
Email: tom@erg.abdn.ac.uk

Raffaele Zullo  
University of Aberdeen  
School of Engineering  
Fraser Noble Building  
Aberdeen AB24 3UE  
UK  
Email: raffaele@erg.abdn.ac.uk
A Loss-Latency Trade-off Signal for the Mobile Network

draft-fossati-tsvwg-lola-00

Abstract

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

Status of This Memo

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

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

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

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

Copyright Notice

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

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

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

2. Terminology

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

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

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

4. Marking

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

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

5. Relationship to a Mobile DiffServ Domain

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

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

6. On Remarking and Bleaching

NQB markings SHOULD be preserved when forwarding via an interconnect.

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

7. IANA Considerations

This document makes no request to IANA.

8. Security Considerations

Internet applications cannot benefit from wrongly indicating low-latency as they have to pay the expense of high loss as a trade-off. Hence there is no incentive for Internet applications to set the wrong code-point.
The NQB signal is not integrity protected and could be flipped by an on-path attacker. This might negatively affect the QoS of the tampered flow.

9. Privacy Considerations

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

10. Acknowledgments

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

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

11. References

11.1. Normative References

[I-D.white-tsvwg-nqb]
White, G., "Identifying and Handling Non Queue Building Flows in a Bottleneck Link", draft-white-tsvwg-nqb-00 (work in progress), October 2018.


11.2. Informative References


Authors' Addresses

Thomas Fossati
Nokia
Email: thomas.fossati@nokia.com

Gorry Fairhurst
University of Aberdeen
Email: gorry@erg.abdn.ac.uk

Pedro A. Aranda Gutierrez
Universidad Carlos III de Madrid
Email: paranda@it.uc3m.es

Mirja Kuehlewind
ETH Zurich
Email: mirja.kuehlewind@tik.ee.ethz.ch
Abstract

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

Status of this Memo

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

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

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

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

Copyright and License Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.
This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

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

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

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

There are two use cases of USH:

1) Protocol trailer (section 2.1)

2) Protocol header or Extended UDP Header (section 2.2)

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

2 UDP Surplus Header format

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

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Padding (0 to 3 bytes)              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Type       |   TSLength    |         Checksum              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
˜                      Type Specific Data                       ˜
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The fields are:

- Padding: Aligns the UDP Surplus Header to four bytes. The number of padding bytes required is: 3 - ((udp_length - 1) % 4), where the udp_length is the length of the UDP datagram as specified in the UDP Length field. Padding bytes MUST be set to zero on transmission, and MUST be verified to be zero when received.
o Type: Indicates the format of the UDP surplus space and how the Type Specific Data is interpreted. Defined Type values are:
  o 0: Reserved
  o 1: UDP options
  o 2-127: Reserved
  o 128-255: Available for private use or experimentation

o TSLength: Length of the type specific data in units of four byte words. The length of the type specific data is thus zero to 1020 bytes.

o Checksum: The standard one’s complement checksum that covers the UDP surplus area. The coverage starts from the first byte of Padding, or the Type field if no padding is present, through the end of the IP packet. If the number of Padding bytes is odd then a zero byte is logically prepended to surplus area for the checksum calculation.

o Type Specific Data: Variable length data that is considered part of the UDP Surplus Header. This data is interpreted per the value of the Type field.

2.1 Protocol trailer format

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

```
+----------+
|          |
|          |
| UDP header |
+----------+
|          |
|          |
| UDP data |
+----------+
\          /  
Surplus space  | USH base header |
\----------/  \----------------/-
                | Type specific data |
```

T. Herbert  Expires January 9, 2020
The packet format of UDP Surplus Header as a protocol trailer is:

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Source port            |      Destination port         | |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ UDP
|           Length              |         Checksum              | |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+/
|                                                               |
˜                           UDP data                            ˜
|                                                               |
+               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |
|               |                      Padding                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+/ |
|     Type      |   TSLength    |         Checksum              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ |
|                                                               |
˜                      Type Specific Data                       ˜ USH
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ /
```

Notes:

- The offset of the UDP Surplus Header from the start of the UDP header, including possible padding for the USH, is equal the UDP Length.

- The number of padding bytes is 3 - ((udp_length - 1) % 4), where udp_length is equal to the UDP Length field. The offset of the Type field of the USH is 4 * ((udp_length - 1) / 4 + 1).

- If the size of the USH header (four plus four times TSLength) is less than the size of the UDP surplus space in a packet, then the USH is considered to be malformed (see section 3.2).

- The UDP checksum covers the UDP header and UDP data. The USH checksum covers the entire UDP surplus space.

- A legacy receiver, one that does not understand the UDP Surplus Header, will ignore the contents of the UDP surplus space and process the UDP data as normal. Protocol data that cannot correctly be ignored by a receiver, such as the fragmentation option in the [UDPOPT], MUST NOT be in a surplus space trailer.
2.2 Protocol header format (Extended UDP header)

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

```
+--------------------------------+  | Extended UDP header |
|       | UDP header                   |  |
| Surplus | UDP space header             |  |
| space   | Type specific data           |  |
|         | UDP data                     |
```

The packet format containing an extended UDP header is:

```
+--------------------------------+  |---------------------------------|
|        Source port            |      Destination port            | |
|--------------------------------+  |---------------------------------|
|           Length              |         Checksum                  | |
|--------------------------------+  |---------------------------------|
|     Type      |   TSLength    |         Checksum                  | |
|--------------------------------+  |---------------------------------|
|                                    | USH                                 |
|                                    |                                    |
```

Notes:

- Since the UDP header is aligned and a multiple of four bytes, no padding for USH is necessary.

- The UDP length is fixed to be eight so that all bytes beyond the UDP header are contained in the surplus space.

- The UDP checksum covers the eight bytes of UDP header and the checksum pseudo header. The USH checksum covers the entire surplus space which includes the UDP Surplus Header and UDP data.
- The UDP data length is the IP payload length minus the size of the UDP header and the size of the UDP Surplus Header. That is:

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

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

3 Operation

3.1 Sender operation

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

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

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

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

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

1) Set the Checksum field to zero. Compute the standard one’s complement two byte checksum starting from the Type field through the end of the IP packet (end of the surplus space). If the length of the surplus space is odd then a zero byte is logically appended for the purposes of the calculation.
2) Set the value of the Checksum field to the bitwise "not" of the checksum computed in the previous step.

3.2 Receiver operation

The processing for a UDP packet with surplus space is:

1) Check for minimum length to contain a UDP Surplus Header. If the UDP surplus space length is less than 3 - ((udp_length - 1) % 4) + 4, then the UDP Surplus Header is considered invalid.

2) Check padding bytes. If the UDP Length is not a multiple of four bytes then verify that the padding bytes following the UDP payload are set to zero. The required number of padding bytes is 3 - ((udp_length - 1) % 4). If the padding bytes are not zero, the UDP Surplus Header is considered invalid.

3) Check the TSLength field. If the length determined from the TSLength field plus the starting offset of the Type Specific Data exceeds the length of the IP packet then the UDP Surplus Header is considered invalid.

4) Verify the checksum. Compute the one’s complement checksum starting from the Type field through the end of the IP packet (the end of the surplus area). If the result of the computation is checksum zero (˜0 or -0) then the checksum is verified. If the checksum is not verified then the UDP Surplus Header is considered invalid.

5) Check the Type. If the Type is unknown to the receiver then the surplus header is considered invalid.

6) Process the Type Specific Data per the Type in the UDP Surplus Header. If an error condition is encountered in the course of processing the Type Specific Data then the receiver SHOULD consider that the UDP Surplus Header is invalid.

7) In the protocol trailer use case, if there are additional bytes beyond the UDP Surplus Header, a receiver SHOULD ignore those bytes (with the exception that the excess bytes MUST be included in the USH Checksum computation).

8) If the UDP Surplus Header is validated and processed, deliver the UDP data to the application.

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

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

3.2.1 Error handling

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

- In the protocol trailer case (UDP Length greater than eight), the UDP surplus area SHOULD be ignored per protocol processing convention. An implementation MAY allow configuration that would discard such packets. An implementation MUST either process the surplus space or ignore the whole space. In particular, the UDP Surplus Header MUST NOT be partially processed lest that leads to indeterminate results of processing an accepted packet.

- In the case of a protocol header (a UDP packet having exactly a length of eight), the receiver SHOULD discard packets with malformed UDP surplus space or UDP Surplus Header. A receiver MAY deliver the packet to the application in the unlikely scenario that the application applies semantics to zero length UDP datagrams and there is the possibility that the surplus space is a legacy use case (i.e. the sender set surplus space but doesn’t use the UDP Surplus Header format).

4 Motivation

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

- While the UDP surplus area was implicitly created by [RFC768], the space was never specifically reserved by IETF action. Prescribing a format enables interoperable and backwards compatible use of this space within the context of defined protocol specifications.

- A common header allows different uses and extensibility of the UDP surplus space within a common framework. This is achieved by inclusion of a Type field and Type Specific Data in the UDP Surplus Header. For instance, legacy uses of surplus space could be adapted to use the format and brought into conformance.

- Since the UDP surplus space was never reserved, there is a possibility that the UDP surplus space is already being used by
some implementations. Disambiguating these "legacy" use cases from a newly defined standard format is essential. The required Checksum field, and to a lesser extent the Type and TSLength fields, help disambiguate uses of the surplus area from legacy or accidental uses of the surplus area. Use of the extended UDP header format also reduces the chances of misinterpreting legacy uses.

- The USH checksum is checksum offload friendly. See appendix A for discussion on checksum offload and USH.

- The required checksum in the UDP Surplus Header properly compensates for those devices that incorrectly compute UDP checksum over the length of the IP payload as opposed to just the UDP length.

- A fixed checksum, as opposed to placing a checksum in options, avoids the problem that a checksum can’t protect against corruption of the type field for the option containing the checksum.

- Protocols headers, such as those used in the Extended UDP Header format, are more implementation friendly than protocol trailers. See Appendix B for more discussion.

- Maintaining four byte alignment, as is common in IP protocols, is beneficial to implementations on several hardware architectures. See Appendix C for more discussion.
5  Security Considerations

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

6  IANA Considerations

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

7  References

7.1  Normative References

7.2  Informative References

[UDPOPT]  Touch, J., "Transport Options for UDP", draft-ietf-tsvwg-
          udp-options-07
Appendix A: Checksum processing

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

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

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

A.1 Transmit Checksum processing

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

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

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

A.1.1 TX checksum for USH trailer

The recommended procedure for setting checksums when the UDP Surplus Header is a trailer is:
1) On the host set the USH checksum using the normal procedures for setting the checksum (section 3.1).

2) Arrange for the UDP checksum to be offloaded to the device. This is done by indicating the checksum start offset to be the first byte of UDP header, indicating the checksum field offset to be the offset of the UDP checksum field, and initializing the UDP checksum field to the "bitwise not" of the appropriate IP pseudo header.

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

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

A.1.2 TX checksum for USH header

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

1) Set the UDP checksum on the host. This is normal procedures to set the UDP checksum for a UDP datagram with length of eight.

2) Arrange to offload the USH checksum. The USH checksum field is initialized to zero, the offset to start the checksum calculation is set to the offset of the Type field in the USH, and the checksum field offset is set to the offset of the USH checksum field.

A.2 Receive Checksum handling

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

A.2.1 Simultaneous verification

If a device provides a checksum complete value and the UDP checksum
is set, then both the UDP checksum and USH checksum can be simultaneously verified:

1) Pull up checksum to start of the UDP header. That is the checksum complete value is computed from the start of the UDP header through the end of the IP packet.

2) Verify the UDP checksum taking into account the pseudo header. If the UDP checksum is verified, then the USH checksum is also verified.

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

A.2.2 RX checksum for USH trailer

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

1) Compute the one’s complement checksum across the UDP surplus space. If checksum zero is the result, then the USH checksum is verified.

2) Perform one’s complement subtraction of the value derived in step 1) from the checksum complete value. The result is the checksum complete value across just the UDP header and UDP data.

3) Compute the IP pseudo header for the UDP checksum and one’s complement add the result to that of step 2). If the result is checksum zero then the UDP checksum is verified.

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

A.2.3 RX checksum for USH header

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

1) Compute the one’s complement checksum across the UDP header.

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

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

Appendix B: Protocol headers versus versus protocol trailers

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

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

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

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

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

Appendix C: Protocol field alignment
This appendix is informational and does not constitute a normative part of this document.

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

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

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

Author’s Address

Tom Herbert
Intel
Santa Clara, CA
USA

Email: tom@quantonium.net
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

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 in the appendices. In many cases the framework simplifies the basic control algorithm, and requires little extra processing. Therefore it is believed the Coupled AQM would be applicable and easy to deploy in all types of buffers; buffers in cost-reduced mass-market residential equipment; buffers in end-system stacks; buffers in carrier-scale equipment including remote access servers, routers, firewalls and Ethernet switches; buffers in network interface cards, buffers in virtualized network appliances, hypervisors, and so on.

For the public Internet, nearly all the benefit will typically be achieved by deploying the Coupled AQM into either end of the access link between a ‘site’ and the Internet, which is invariably the bottleneck. 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 watchings videos over the same 40Mb/s downstream broadband link, latency was so low that the football picture appeared to stick to the user’s finger on the 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:

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

(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 [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

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

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

$$p' = \frac{p_{CL}}{k}.$$

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

$$p_{CL} = k*p',$$

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:

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

which has also been found to work very well in practice.
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 Dual Queue 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*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:
o Optional packet classifier(s) to use in addition to the ECN field (see Section 2.3);

o 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 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.

o 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.

o Coupling factor, k;

o 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...
packet is \( t \text{shift} \) greater than that of an L packet. \( t \text{shift} \) could be expressed as a multiple of the typical RTT rather than as an absolute delay.

- The maximum Classic ECN marking probability, \( p_{C_{\text{max}}} \), 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'etre 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 queueing 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]  
CableLabs, "MAC and Upper Layer Protocols Interface (MULPI) Specification, CM-SP-MULPIv3.1", Data-Over-Cable Service Interface Specifications DOCSIS(R) 3.1 Version i17 or later, January 2019, <https://specification-search.cablelabs.com/CM-SP-MULPIv3.1>


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:   limit = MAX_LINK_RATE * 250 ms % Dual buffer size
4:   k = 2 % Coupling factor
5:   % NOT SHOWN % scheduler-dependent weight or equival’t parameter
6:   % PI2 AQM parameters
7:   RTT_max = 100 ms % Worst case RTT expected
8:   RTT_typ = 15 ms % Typical RTT
9:   % PI2 constants derived from above PI2 parameters
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:  % L4S ramp AQM parameters
16:  minTh = 475 us % L4S min marking threshold in time units
17:  range = 525 us % Range of L4S ramp in time units
18:  Th_len = 2 * MTU % Min L4S marking threshold in bytes
19:  % L4S constants incl. those derived from other parameters
20:  p_Lmax = 1 % Max L4S marking prob
21:  floor = Th_len / MIN_LINK_RATE
22:  if (minTh < floor) {
23:     % Shift ramp so minTh >= serialization time of 2 MTU
24:     minTh = floor
25:  }
26:  maxTh = minTh+range % L4S max marking threshold in time units
27:   }
28: }

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 ) { % Scheduler chooses lq
4:        lq.dequeue(pkt) % Native L4S AQM
5:        p'_L = laqm(lq.time())
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:             continue % continue to the top of the while loop
15:           }
16:           mark(pkt) % squared mark
17:         } % packet loss
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 queueing delay, which avoids the complexity of smoothing, but also avoids embedding a worst-case RTT of smoothing delay in the network (see Section 2.1).

- The ramp rises linearly directly from 0 to 1, not to an intermediate value of $p'_L$ as RED would, because there is no need to keep ECN marking probability low.

- Marking does not have to be randomized. Determinism is used instead of randomness; to reduce the delay necessary to smooth out the noise of randomness from the signal.

The ramp function requires two configuration parameters, the minimum threshold ($\text{minTh}$) and the width of the ramp ($\text{range}$), 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 $0.17^*\text{RTT_typ}$, 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 $\text{RTT_typ}/30$ is recommended.

A minimum marking threshold parameter ($\text{Th_len}$) 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) { % Returns native L4S AQM probability
2:    if (qdelay >= maxTh)
3:      return 1
4:    else if (qdelay > minTh)
5:      return (qdelay - minTh)/range % Divide could use a bit-shift
6:    else
7:      return 0
8:  }
```

Figure 5: Example Pseudocode for the Native L4S AQM

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

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

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
differences in queueing delay into changes in probability (assuming probability has a value from 0 to 1).

alpha and beta determine how much p’ ought to change after each update interval (Tupdate). For smaller Tupdate, p’ should change by the same amount per second, but in finer more frequent steps. So alpha depends on Tupdate (see line 14 of the initialization function in Figure 2). It is best to update p’ as frequently as possible, but Tupdate will probably be constrained by hardware performance. As shown in line 13, the update interval should be 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 Tupdate=16ms (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 * Tupdate / 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 Tupdate=16 ms and RTT_max = 100 ms, they result in alpha = 0.16; beta = 3.2 (discrepancies are due to rounding). These defaults have been verified with a wide range of link rates, target delays and a range of traffic models with mixed and similar RTTs, short and long flows, etc.

In corner cases, p’ can overflow the range [0,1] so the resulting value of p’ has to be bounded (omitted from the pseudocode). Then, as already explained, the coupled and Classic probabilities are derived from the new p’ in lines 4 and 5 of Figure 6 as p_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 Tupdate dependent on p’. Instead, in PI2, alpha and beta are independent of p’ because the squaring applied to Classic traffic
tunes them inherently. This is explained in [PI2], which also explains why this more principled approach removes the need for most of the heuristics that had to be added to PIE.

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.

```
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)
8b:       } else {                              % overload saturation
8c:         if ( p_C > rand() ) {            % probability p_C = p'^2
8e:           drop(pkt)      % revert to Classic drop due to overload
8f:           continue        % continue to the top of the while loop
8g:         }
8h:       if ( 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 mark
14:         continue % continue to the top of the while loop
15:       }
16:     mark(pkt)                                  % squared mark
17:   }
18: }
19: return(pkt)                      % return the packet and stop
20: }
21: return(NULL)                             % no packet to dequeue
22: }
```

Figure 7: Example Dequeue Pseudocode for DualQ Coupled PI2 AQM
(Including 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, biased against the Classic traffic by a time-shift of tshift. To implement time-shifted FIFO, the scheduler() function in line 3 of the dequeue code would simply be implemented as the scheduler() function at the bottom of Figure 10 in Appendix B. For the public Internet a good value for tshift is 50ms. For private networks with smaller diameter, about 4*target would be reasonable. TS-FIFO is a very simple scheduler, but complexity might need to be added to address some deficiencies (which is why it is not recommended over WRR):

  * TS-FIFO does not fully isolate latency in the L4S queue from uncontrolled bursts in the Classic queue;
  
  * TS-FIFO is only appropriate if time-stamping of packets is feasible;
  
  * Even if time-stamping is supported, the sojourn time of the head packet is always stale. For instance, if a burst arrives at an empty queue, the sojourn time will only measure the delay...

Figure 8: Example PI-Update Pseudocode for DualQ Coupled PI2 AQM (Including Overload Code)
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.
cred_params_init(...) {
  % Set input parameter defaults
  % DualQ Coupled framework parameters
  limit = MAX_LINK_RATE * 250 ms  % Dual buffer size
  k' = 1                        % Coupling factor as a power of 2
  tshift = 50 ms                % Time shift of TS-FIFO scheduler
  % Constants derived from Classic AQM parameters
  k = 2^k'                     % Coupling factor from Equation (1)
  % Classic AQM parameters
  g_C = 5            % EWMA smoothing parameter as a power of 1/2
  S_C = -1          % Classic ramp scaling factor as a power of 2
  minTh = 500 ms    % No Classic drop/mark below this queue delay
  % Constants derived from Classic AQM parameters
  gamma = 2^(-g_C)                     % EWMA smoothing parameter
  range_C = 2^S_C                         % Range of Classic ramp
  % L4S AQM parameters
  T = 1 ms             % Queue delay threshold for native L4S AQM
  % Constants derived from above parameters
  S_L = S_C - k'        % L4S ramp scaling factor as a power of 2
  range_L = 2^S_L                             % Range of L4S ramp
}

Figure 9: Example Header Pseudocode for DualQ Coupled Curvy RED AQM
cred_dequeue(lq, cq, pkt) {  
% Couples L4S & Classic queues
   while ( lq.len() + cq.len() > 0 ) {
      if ( scheduler() == lq ) {
        lq.dequeue(pkt)                            % L4S scheduled
        p_CL = (cq.time() - minTh) / range_L
        if (  ( lq.time() > T )  
          OR ( p_CL > maxrand(U) ) )
          mark(pkt)
      } else {
        cq.dequeue(pkt)                        % Classic scheduled
        Q_C = gamma * qc.time() + (1-gamma) * Q_C  % Classic Q EWMA
        sqrt_p_C = (Q_C - minTh) / range_C
        if ( sqrt_p_C > maxrand(2*U) ) {
          if ( (ecn(pkt) == 0)  {            % ECN field = not-ECT
            drop(pkt)                    % Squared drop, redo loop
            continue       % continue to the top of the while loop
          }  
          mark(pkt)
        }
      }
     }
   return(pkt)                % return the packet and stop here
  }  
  return(NULL)                            % no packet to dequeue
}

maxrand(u) {                % return the max of u random numbers
   maxr=0
   while (u-- > 0)
     maxr = max(maxr, rand())                   % 0 <= rand() < 1
   return(maxr)
}

scheduler() {
   if ( lq.time() + tshift >= cq.time() )
     return lq;
   else
     return cq;
}

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 to be dequeued, then returns control to the caller, so that it does not block while the dequeue is in progress. While making the dequeue decision, it also makes the AQM decisions on dropping or marking. The alternative of applying the AQMs at enqueue would shift some processing from the lower layer to the upper layer.
critical time when each packet is dequeued. However, it would also add a whole queue of delay to the control signals, making the control loop very sloppy.

The code is written assuming the AQMs are applied on dequeue (Note 1). All the dequeue code is contained within a large while loop so that if it decides to drop a packet, it will continue until it selects a packet to schedule. If both queues are empty, the routine returns NULL at line 20. Line 3 of the dequeue pseudocode is where the conditional priority scheduler chooses between the L4S queue (lq) and the Classic queue (cq). The time-shifted FIFO scheduler is shown at lines 28-33, which would be suitable if simplicity is paramount (see Note 2).

Within each queue, the decision whether to forward, drop or mark is taken as follows (to simplify the explanation, it is assumed that U=1):

L4S: If the test at line 3 determines there is an L4S packet to dequeue, the tests at lines 5b and 5c determine whether to mark it. The first is a simple test of whether the L4S queue delay (lq.time()) is greater than a step threshold T (Note 3). The second test is similar to the random ECN marking in RED, but with the following differences: i) marking depends on queuing time, not bytes, in order to scale for any link rate without being reconfigured; ii) marking of the L4S queue 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 \( \text{sqrt}_p_C \). 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.

Figure 11: Optimised Example Dequeue Pseudocode for Coupled DualQ AQM using Integer Arithmetic

The two ranges, range_L and range_C are expressed as powers of 2 so that division can be implemented as a right bit-shift (>>) in lines 5 and 10 of the integer variant of the pseudocode (Figure 11).

For the integer variant of the pseudocode, an integer version of the rand() function used at line 25 of the maxrand(function) in Figure 10 would be arranged to return an integer in the range \(0 \leq \text{maxrand()} < 2^{32}\) (not shown). This would scale up all the floating point probabilities in the range \([0,1]\) by \(2^{32}\).

Queuing delays are also scaled up by \(2^{32}\), but in two stages: i) In 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

<table>
<thead>
<tr>
<th>RTT_C / RTT_L</th>
<th>Reno</th>
<th>Cubic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>k'=1</td>
<td>k'=0</td>
</tr>
<tr>
<td>2</td>
<td>k'=2</td>
<td>k'=1</td>
</tr>
<tr>
<td>3</td>
<td>k'=2</td>
<td>k'=2</td>
</tr>
<tr>
<td>4</td>
<td>k'=3</td>
<td>k'=2</td>
</tr>
<tr>
<td>5</td>
<td>k'=3</td>
<td>k'=3</td>
</tr>
</tbody>
</table>

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]:

\[ 2^{k'} = 1.64 \left( \frac{RTT_{reno}}{RTT_{dc}} \right) \quad (2) \]
\[ 2^{k'} = 1.19 \left( \frac{RTT_{cubic}}{RTT_{dc}} \right) \quad (3) \]

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 point.

Fairhurst, et al. Expires 7 December 2019
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 PMTU ......................... 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 directed 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 end point. 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.
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
The method described in this document for datagram PLs, which is an extension to Classical PMTU Discovery.

Probes serve to determine 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 cannot 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 mechanism:

* 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
quoted packet - this is also necessary for the PTB message to be passed to the corresponding PL.

The validation SHOULD utilize information that it is not simple for an off-path attacker to determine [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:

\[ \text{MIN}_\text{PMTU} < \text{PTB}_\text{SIZE} < \text{BASE}_\text{PMTU} \]

* 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 \text{BASE}_\text{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 \text{BASE}_\text{PMTU}.

\[ \text{PTB}_\text{SIZE} = \text{PLPMTU} \]

* Completes the search for a larger PLPMTU.

\[ \text{PTB}_\text{SIZE} > \text{PROBED}_\text{SIZE} \]

* 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 \leq 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-date 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             v            Actual PMTU
                      |            PROBED_SIZE
                      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
  +----+       v
  |      Connectivity and BASE_PMTU confirmed +----+
  v       Consistent connectivity and BASE_PMTU confirmed

PLPMTU confirmation failed

     +----+       Search completed
     +----+       v
     v
     +----+ 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 PMTU changes or when the method (for whatever reason) makes a suboptimal choice for the PLPMTU.

A full implementation of DPLPMTUD provides an algorithm enabling the DPLPMTUD sender to increase the PLPMTU following a change in the characteristics of the path, such as when a link is reconfigured with a larger MTU, or when there is a change in the set of links traversed by an end-to-end flow (e.g., after a routing or path fail-over decision).

5.2. State Machine

A state machine for DPLPMTUD is depicted in Figure 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.
Figure 4: State machine for Datagram PLPMTUD

The following states are defined:

DISABLED: The DISABLED state is the initial state before probing has started. It is also entered from any other state, when the PL indicates loss of connectivity.
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 SEARCHCOMPLETE 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 DPLPMTUD 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
[I-D.ietf-quic-transport]


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 evaluated and recommendations to be considered

* Section added to align with work proposed in the QUIC WG.

Working Group draft -02:

* The draft was updated based on feedback from the WG, and a detailed review by Magnus Westerlund.

* The document updates RFC 4821.

* Requirements list updated.

* Added more explicit discussion of a simpler black-hole detection mode.

* This draft includes reorganisation of the section on IETF protocols.

* Added more discussion of implementation within an application.

* Added text on flapping paths.

* Replaced ’effective MTU’ with new term PLPMTU.

Working Group draft -03:
* Updated figures
* Added more discussion on blackhole detection
* Added figure describing just blackhole detection
* Added figure relating MPS sizes

Working Group draft -04:
* Described phases and named these consistently.
* Corrected transition from confirmation directly to the search phase (Base has been checked).
* Redrawn state diagrams.
* Renamed BASE_MTU to BASE_PMTU (because it is a base for the PMTU).
* Clarified Error state.
* Clarified suspending DPLPMTUD.
* Verified normative text in requirements section.
* Removed duplicate text.
* Changed all text to refer to /packet probe/probe packet/
  /validation/verification/ added term /Probe Confirmation/ and
  clarified BlackHole detection.

Working Group draft -05:
* Updated security considerations.
* Feedback after speaking with Joe Touch helped improve UDP-Options
  description.

Working Group draft -06:
* Updated description of ICMP issues in section 1.1
* Update to description of QUIC.

Working group draft -07:
* Moved description of the PTB processing method from the PTB
  requirements section.
* Clarified what is performed in the PTB validation check.
* Updated security consideration to explain PTB security without needing to read the rest of the document.
* Reformatted state machine diagram

Working group draft -08:
* Moved to rfcxml v3+
* Rendered diagrams to svg in html version.
* Removed Appendix A. Event-driven state changes.
* Removed section on DPLPMTUD with UDP Options.
* Shortened the description of phases.

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
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.
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 decapsulation, as in the pipe model of [RFC2983], the special semantics of the ECN field would be lost. Similarly, if the DSCP is changed at the boundary between Diffserv domains, the special ECN semantics would also be lost. This is an important implication of the localized scope of most Diffserv arrangements. In this document, correct propagation of traffic class information is assumed, while what ‘correct’ means and how it is achieved is covered elsewhere (e.g. RFC 2983) and is outside the scope of the present document.

The guidelines in this document do ensure that common encapsulation and decapsulation rules are sufficiently generic to cover cases where ECT(1) is used instead of ECT(0) to identify alternative ECN semantics (as in L4S [I-D.ietf-tsvwg-ecn-l4s-id]) and where ECN marking algorithms use ECT(1) to encode 3 severity levels into the ECN field (e.g. PCN [RFC6660]) rather than the default of 2. All these different semantics for the ECN field work because it has been
possible to define common default decapsulation rules that allow for all cases.

Note that the guidelines in this document do not necessarily require the subnet wire protocol to be changed to add support for congestion notification. For instance, the Feed-Up-and-Forward Mode (Section 3.2) and the Null Mode (Section 3.4) do not. Another way to add congestion notification without consuming header space in the subnet protocol might be to use a parallel control plane protocol.

This document focuses on the congestion notification interface between IP and lower layer or tunnel protocols that can encapsulate IP, where the term ‘IP’ includes v4 or v6, unicast, multicast or anycast. However, it is likely that the guidelines will also be useful when a lower layer protocol or tunnel encapsulates itself, e.g. Ethernet MAC in MAC ([IEEE802.1Q]; previously 802.1ah) or when it encapsulates other protocols. In the feed-backward mode, propagation of congestion signals for multicast and anycast packets is out-of-scope (because the complexity would make it unlikely to be attempted).

2. Terminology

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

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
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. A
corollary of the above is that a lower layer congestion
notification protocol:

3. SHOULD be able to distinguish ECN-PDUs from Not-ECN-PDUs.

Note that there is no need for all interior nodes within a subnet to
be able to mark congestion explicitly. A mix of ECN and drop signals
from different nodes is fine. However, if _any_ interior nodes might
generate ECN markings, guideline 2 above says that all relevant
egress node(s) SHOULD be able to propagate those markings up to the
higher layer.

In IP, if the ECN field in each PDU is cleared to the Not-ECT (not
ECN-capable transport) codepoint, it indicates that the L4 transport
will not understand congestion markings. A congested buffer must not
mark these Not-ECT PDUs, and therefore drops them instead.

The mechanism a lower layer uses to distinguish the ECN-capability of
PDUs need not mimic that of IP. The above guidelines merely say that
the lower layer system, as a whole, should achieve the same outcome.
For instance, ECN-capable feedback loops might use PDUs that are
identified by a particular set of labels or tags. Alternatively,
logical link protocols that use flow state might determine whether a
PDU can be congestion marked by checking for ECN-support in the flow
state. Other protocols might depend on out-of-band control signals.

The per-domain checking of ECN support in MPLS [RFC5129] is a good
example of a way to avoid sending congestion markings to L4
transports that will not understand them, without using any header
space in the subnet protocol.

In MPLS, header space is extremely limited, therefore RFC5129 does
not provide a field in the MPLS header to indicate whether the PDU is
an ECN-PDU or a Not-ECN-PDU. Instead, interior nodes in a domain are
allowed to set explicit congestion indications without checking
whether the PDU is destined for a L4 transport that will understand
them. Nonetheless, this is made safe by requiring that the network
operator upgrades all decapsulating edges of a whole domain at once,
as soon as even one switch within the domain is configured to mark
rather than drop during congestion. Therefore, any edge node that
might decapsulate a packet will be capable of checking whether the
higher layer transport is ECN-capable. When decapsulating a CE-
marked packet, if the decapsulator discovers that the higher layer
(inner header) indicates the transport is not ECN-capable, it drops
the packet--effectively on behalf of the earlier congested node (see
Decapsulation Guideline 1 in Section 4.4).
It was only appropriate to define such an incremental deployment strategy because MPLS is targeted solely at professional operators, who can be expected to ensure that a whole subnetwork is consistently configured. This strategy might not be appropriate for other link technologies targeted at zero-configuration deployment or deployment by the general public (e.g., Ethernet). For such 'plug-and-play' environments it will be necessary to invent a failsafe approach that ensures congestion markings will never fall into black holes, no matter how inconsistently a system is put together. Alternatively, congestion notification relying on correct system configuration could be confined to flavours of Ethernet intended only for professional network operators, such as Provider Backbone Bridges (PBB [IEEE802.1Q]; previously 802.1ah).

ECN support in TRILL [I-D.ietf-trill-ecn-support] provides a good example of how to add ECN to a lower layer protocol without relying on careful and consistent operator configuration. TRILL provides an extension header word with space for flags of different categories depending on whether logic to understand the extension is critical. The congestion experienced marking has been defined as a 'critical ingress-to-egress' flag. So if a transit RBridge sets this flag and an egress RBridge does not have any logic to process it, it will drop it; which is the desired default action anyway. Therefore TRILL 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 taken to ensure that the field used is specified as mutable during transit. Otherwise interior nodes signalling congestion would invalidate any authentication protocol applied to the lower layer header--by altering a header field that had been assumed as immutable.

The redesign of protocols that encapsulate IP in order to propagate congestion signals between layers raises potential signal integrity concerns. Experimental or proposed approaches exist for assuring the end-to-end integrity of in-band congestion signals, e.g.:

- 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 Romascun 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-rfc6040update-shim

* Wire Protocol Design: Indication of ECN Support: Added TRILL as an example of a well-design protocol that does not need an indication of ECN support in the wire protocol.

* Encapsulation Guidelines: In the case of a Not-ECN-PDU with a CE outer, replaced SHOULD be dropped, with explanations of when SHOULD or MUST are appropriate.

* Feed-Up-and-Forward Mode: Explained examples more carefully, referred to PDCP and cited UTRAN spec as well as E-UTRAN.

* Updated references.

* Marked open issues as resolved, but did not delete Open Issues Appendix (yet).

From ietf-04 to ietf-05:

* Explained why tightly coupled shim headers only "SHOULD" comply with RFC 6040, not "MUST".

* Updated references

From ietf-03 to ietf-04:

* Addressed Richard Scheffenegger’s review comments: primarily editorial corrections, and addition of examples for clarity.

From ietf-02 to ietf-03:

* Updated references, ad cited RFC4774.

From ietf-01 to ietf-02:

* Added Section for guidelines that are applicable in all cases.

* Updated references.

From ietf-00 to ietf-01: Updated references.
From briscoe-04 to ietf-00: Changed filename following tsvwg adoption.

From briscoe-03 to 04:

* Re-arranged the introduction to describe the purpose of the document first before introducing ECN in more depth. And clarified the introduction throughout.

* Added applicability to 3GPP TS 36.300.

From briscoe-02 to 03:

* Scope section:
  + Added dependence on correct propagation of traffic class information
  + For the feed-forward 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:
Internet-Draft     ECN Semantics for Low Queuing Delay         July 2019

o  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;

o  changes the status of the experimental ECN nonce [RFC3540] to
historic;

o  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:

o  it SHOULD survive end-to-end between source and destination
applications: across the boundary between host and network,
between interconnected networks, and through middleboxes;

o  it SHOULD be common to IPv4 and IPv6 and transport-agnostic;

o  it SHOULD be incrementally deployable;

o  it SHOULD enable an AQM to classify packets encapsulated by outer
    IP or lower-layer headers;

o  it SHOULD consume minimal extra codepoints;

o  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
round trip. 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

\[ p_C \sim \left(\frac{p_L}{k}\right)^2 \]
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].
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

[Alizadeh-stability]


[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]


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 queueing 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 supersed 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 wil 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;
B. 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);

C. 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.
<table>
<thead>
<tr>
<th>Issue</th>
<th>DSCP + ECN</th>
<th>ECN</th>
<th>ECT(1) + CE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>initial</td>
<td>initial</td>
<td>initial</td>
</tr>
<tr>
<td>lower layers</td>
<td>N . .</td>
<td>. . Y</td>
<td>N . .</td>
</tr>
</tbody>
</table>

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/
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.
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 $0 < \text{code rate} \leq 1$. A code rate close to 1 indicates that a small number of repair symbols have been produced during the encoding process.

Encoding Symbol: Unit of data generated by the encoding process. With systematic codes, source symbols are part of the encoding symbols.
Packet Erasure Channel: A communication path where packets are either lost (e.g., in our case, by a congested router, or because the number of transmission errors exceeds the correction capabilities of the physical-layer code) or received. When a packet is received, it is assumed that this packet is not corrupted (i.e., in our case, the bit-errors, if any, are fixed by the physical-layer code and therefore hidden to the upper layers).

Repair Symbol: Encoding symbol that is not a source symbol.

Source Block: Group of ADUs that are to be FEC protected as a single block. This notion is restricted to Block FEC Codes.

Source Symbol: Unit of data used during the encoding process.

Systematic Code: FEC code in which the source symbols are part of the encoding symbols.

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

3. Summary of Architecture Overview

The architecture of [RFC6363], Section 3, equally applies to this FECFRAME extension and is not repeated here. However, we provide hereafter a quick summary to facilitate the understanding of this document to readers not familiar with the concepts and terminology.
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.
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) queueing 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.
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 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
enough paced and low enough rate not to build a queue.  For
instance, VoIP, low rate datagrams to sync online games,
relatively low rate application-limited traffic, DNS, LDAP, etc.
This traffic would need to be tagged with specific identifiers,
e.g. a low latency DiffServ Codepoint such as Expedited
Forwarding (EF [RFC3246]), Non-Queue-Building (NQB
[I-D.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 unscaleable' 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-tswg-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.

- Private networks of heterogeneous data centres, where there is no single administrator that can arrange for all the simultaneous changes to senders, receivers and network needed to deploy DCTCP:
  - a set of private data centres interconnected over a wide area with separate administrations, but within the same company
  - a set of data centres operated by separate companies interconnected by a community of interest network (e.g. for the finance sector)
  - multi-tenant (cloud) data centres where tenants choose their operating system stack (Infrastructure as a Service - IaaS)

- Different types of transport (or application) congestion control:
  - elastic (TCP/SCTP);
  - real-time (RTP, RMCAT);
  - query (DNS/LDAP).

- Where low delay quality of service is required, but without inspecting or intervening above the IP layer [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.

- 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-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>DCTCP (existing)</td>
<td>DualQ AQM downstream</td>
</tr>
<tr>
<td></td>
<td>WorkS DOWNSTREAM FOR CONTROLLED DEPLOYMENTS/TRIALS</td>
<td></td>
</tr>
<tr>
<td>2 TCP Prague</td>
<td>AccECN (already in progress:DCTCP/BBR)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FULLY WORKS DOWNSTREAM</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>DualQ AQM upstream</td>
<td>TCP Prague</td>
</tr>
<tr>
<td></td>
<td>FULLY WORKS UPSTREAM AND DOWNSTREAM</td>
<td></td>
</tr>
</tbody>
</table>

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].
The TCP authentication option (TCP-AO [RFC5925]) can be used to detect tampering with TCP congestion feedback.

The ECN Nonce [RFC3540] was proposed to detect tampering with congestion feedback, but it has been reclassified as historic [RFC8311].

Appendix C.1 of [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


[I-D.briscoe-docsis-q-protection]

[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.cardwell-iccrg-bbr-congestion-control]

[I-D.ietf-quic-transport]

[I-D.ietf-tcpm-accurate-ecn]

[I-D.ietf-tcpm-generalized-ecn]

[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]

Smith, K., "Network management of encrypted traffic", draft-smith-encrypted-traffic-management-05 (work in progress), May 2016.


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 a 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],</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[I-D.stewart-tsvwg-sctpecn].</td>
</tr>
<tr>
<td></td>
<td>SCALABLE TRANSPORT -</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SAFETY ADDITIONS</td>
<td></td>
</tr>
<tr>
<td>4-1</td>
<td>Fall back to Reno/Cubic on loss</td>
<td>[I-D.ietf-tsvwg-ecn-l4s-id] S.2.3,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[RFC8257]</td>
</tr>
<tr>
<td>4-2</td>
<td>Fall back to Reno/Cubic if classic</td>
<td>[I-D.ietf-tsvwg-ecn-l4s-id] S.2.3</td>
</tr>
<tr>
<td></td>
<td>ECN bottleneck detected</td>
<td></td>
</tr>
<tr>
<td>4-3</td>
<td>Reduce RTT-dependence</td>
<td>[I-D.ietf-tsvwg-ecn-l4s-id] S.2.3</td>
</tr>
<tr>
<td>4-4</td>
<td>Scaling TCP’s</td>
<td>[I-D.ietf-tsvwg-ecn-l4s-id] S.2.3,</td>
</tr>
<tr>
<td></td>
<td>Congestion Window for Small Round</td>
<td>[TCP-sub-mss-w]</td>
</tr>
<tr>
<td></td>
<td>Trip Times</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SCALABLE TRANSPORT -</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PERFORMANCE ENHANCEMENTS</td>
<td></td>
</tr>
<tr>
<td>5-1</td>
<td>Setting ECT in TCP</td>
<td>[I-D.ietf-tcpm-generalized-ecn]</td>
</tr>
<tr>
<td>5-2</td>
<td>Faster-than-additive increase</td>
<td>[I-D.ietf-tsvwg-ecn-l4s-id] (Appx A.2.2)</td>
</tr>
<tr>
<td>5-3</td>
<td>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>
<td>TCP</td>
</tr>
<tr>
<td>----</td>
<td>--------</td>
<td>-----</td>
</tr>
<tr>
<td>0</td>
<td>tsvwg</td>
<td>Y</td>
</tr>
<tr>
<td>1</td>
<td>tsvwg</td>
<td>n/a</td>
</tr>
<tr>
<td>2</td>
<td>tsvwg</td>
<td>n/a</td>
</tr>
<tr>
<td>3</td>
<td>tcpm</td>
<td>Y</td>
</tr>
<tr>
<td>4-1</td>
<td>tcpm</td>
<td>Y</td>
</tr>
<tr>
<td>4-2</td>
<td>tcpm/iccrg?</td>
<td>Y</td>
</tr>
<tr>
<td>4-3</td>
<td>tcpm/iccrg?</td>
<td>Y</td>
</tr>
<tr>
<td>4-4</td>
<td>tcpm</td>
<td>Y</td>
</tr>
<tr>
<td>5-1</td>
<td>tcpm</td>
<td>Y</td>
</tr>
<tr>
<td>5-2</td>
<td>tcpm/iccrg?</td>
<td>Y</td>
</tr>
<tr>
<td>5-3</td>
<td>tcpm/iccrg?</td>
<td>Y</td>
</tr>
</tbody>
</table>

Authors’ Addresses

Bob Briscoe (editor)
CableLabs
UK

Email: ietf@bobbriscoe.net
URI: http://bobbriscoe.net/
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. Remarketing to other DSCP(PS)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][RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Applicability

A Lower Effort PHB is applicable for many applications that otherwise use best-effort delivery. More specifically, it is suitable for traffic and services that can tolerate strongly varying throughput
for their data flows, especially periods of very low throughput or even starvation (i.e., long interruptions due to significant or even complete packet loss). Therefore, an application sending an LE marked flow needs to be able to tolerate short or (even very) long interruptions due to the presence of severe congestion conditions during the transmission of the flow. Thus, there ought to be an expectation that packets of the LE PHB could be excessively delayed or dropped when any other traffic is present. It is application-dependent when a lack of progress is considered being a failure (e.g., if a transport connection fails due to timing out, the application may try several times to re-establish the transport connection in order to resume the application session before finally giving up). The LE PHB is suitable for sending traffic of low urgency across a Differentiated Services (DS) domain or DS region.

Just like best-effort traffic, LE traffic SHOULD be congestion controlled (i.e., use a congestion controlled transport or implement an appropriate congestion control method [RFC2914] [RFC8085]). Since LE traffic could be starved completely for a longer period of time, transport protocols or applications (and their related congestion control mechanisms) SHOULD be able to detect and react to such a starvation situation. An appropriate reaction would be to resume the transfer instead of aborting it, i.e., an LE optimized transport ought to use appropriate retry strategies (e.g., exponential back-off with an upper bound) as well as corresponding retry and timeout limits in order to avoid the loss of the connection due to the mentioned starvation periods. While it is desirable to achieve a quick resumption of the transfer as soon as resources become available again, it may be difficult to achieve this in practice. In lack of a transport protocol and congestion control that are adapted to LE, applications can also use existing common transport protocols and implement session resumption by trying to re-establish failed connections. Congestion control is not only useful to let the flows within the LE behavior aggregate adapt to the available bandwidth that may be highly fluctuating, but is also essential if LE traffic is mapped to the default PHB in DS domains that do not support LE. In this case, use of background transport protocols, e.g., similar to LEDBAT [RFC6817], is expedient.

Use of the LE PHB might assist a network operator in moving certain kinds of traffic or users to off-peak times. Furthermore, packets can be designated for the LE PHB when the goal is to protect all other packet traffic from competition with the LE aggregate while not completely banning LE traffic from the network. An LE PHB SHOULD NOT be used for a customer’s "normal Internet" traffic and packets SHOULD NOT be "downgraded" to the LE PHB instead of being dropped, particularly when the packets are unauthorized traffic. The LE PHB
is expected to have applicability in networks that have at least some unused capacity at certain periods.

The LE PHB allows networks to protect themselves from selected types of traffic as a complement to giving preferential treatment to other selected traffic aggregates. LE ought not to be used for the general case of downgraded traffic, but could be used by design, e.g., to protect an internal network from untrusted external traffic sources. In this case there is no way for attackers to preempt internal (non LE) traffic by flooding. Another use case in this regard is forwarding of multicast traffic from untrusted sources. Multicast forwarding is currently enabled within domains only for specific sources within a domain, but not for sources from anywhere in the Internet. A major problem is that multicast routing creates traffic sources at (mostly) unpredictable branching points within a domain, potentially leading to congestion and packet loss. In the case of multicast traffic packets from untrusted sources are forwarded as LE traffic, they will not harm traffic from non-LE behavior aggregates. A further related use case is mentioned in [RFC3754]: preliminary forwarding of non-admitted multicast traffic.

There is no intrinsic reason to limit the applicability of the LE PHB to any particular application or type of traffic. It is intended as an additional traffic engineering tool for network administrators. For instance, it can be used to fill protection capacity of transmission links that is otherwise unused. Some network providers keep link utilization below 50% to ensure that all traffic is forwarded without loss after rerouting caused by a link failure (cf. Section 6 of [RFC3439]). LE marked traffic can utilize the normally unused capacity and will be preempted automatically in case of link failure when 100% of the link capacity is required for all other traffic. Ideally, applications mark their packets as LE traffic, since they know the urgency of flows. Since LE traffic may be starved for longer periods of time it is probably less suitable for real-time and interactive applications.

Example uses for the LE PHB:

- 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.
o For backup traffic or non-time critical synchronization or mirroring traffic.

o For content distribution transfers between caches.

o For preloading or prefetching objects from web sites.

o For network news and other "bulk mail" of the Internet.

o For "downgraded" traffic from some other PHB when this does not violate the operational objectives of the other PHB.

o For multicast traffic from untrusted (e.g., non-local) sources.

4. PHB Description

The LE PHB is defined in relation to the default PHB (best-effort). A packet forwarded with the LE PHB SHOULD have lower precedence than packets forwarded with the default PHB, i.e., in the case of congestion, LE marked traffic SHOULD be dropped prior to dropping any default PHB traffic. Ideally, LE packets would be forwarded only when no packet with any other PHB is awaiting transmission. This means that in case of link resource contention LE traffic can be starved completely, which may not be always desired by the network operator's policy. The used scheduler to implement the LE PHB may reflect this policy accordingly.

A straightforward implementation could be a simple priority scheduler serving the default PHB queue with higher priority than the lower-effort PHB queue. Alternative implementations may use scheduling algorithms that assign a very small weight to the LE class. This, however, could sometimes cause better service for LE packets compared to BE packets in cases when the BE share is fully utilized and the LE share not.

If a dedicated LE queue is not available, an active queue management mechanism within a common BE/LE queue could also be used. This could drop all arriving LE packets as soon as certain queue length or sojourn time thresholds are exceeded.

Since congestion control is also useful within the LE traffic class, Explicit Congestion Notification (ECN) [RFC3168] SHOULD be used for LE packets, too. More specifically, an LE implementation SHOULD also apply CE marking for ECT marked packets and transport protocols used for LE SHOULD support and employ ECN. For more information on the benefits of using ECN see [RFC8087].
5. Traffic Conditioning Actions

If possible, packets SHOULD be pre-marked in DS-aware end systems by applications due to their specific knowledge about the particular precedence of packets. There is no incentive for DS domains to distrust this initial marking, because letting LE traffic enter a DS domain causes no harm. Thus, any policing such as limiting the rate of LE traffic is not necessary at the DS boundary.

As for most other PHBs an initial classification and marking can be also performed at the first DS boundary node according to the DS domain’s own policies (e.g., as protection measure against untrusted sources). However, non-LE traffic (e.g., BE traffic) SHOULD NOT be remarked to LE. Remarking traffic from another PHB results in that traffic being "downgraded". This changes the way the network treats this traffic and it is important not to violate the operational objectives of the original PHB. See also remarks with respect to downgrading in Section 3 and Section 8.

6. Recommended DS Codepoint

The RECOMMENDED codepoint for the LE PHB is ‘000001’.

Earlier specifications [RFC4594] recommended to use CS1 as codepoint (as mentioned in [RFC3662]). This is problematic since it may cause a priority inversion in Diffserv domains that treat CS1 as originally proposed in [RFC2474], resulting in forwarding LE packets with higher precedence than BE packets. Existing implementations SHOULD transition to use the unambiguous LE codepoint ‘000001’ whenever possible.

This particular codepoint was chosen due to measurements on the currently observable DSCP remarking behavior in the Internet [ietf99-secchi]. Since some network domains set the former IP precedence bits to zero, it is possible that some other standardized DSCPs get mapped to the LE PHB DSCP if it were taken from the DSCP standards action pool 1 (xxxxx0).

7. Deployment Considerations

In order to enable LE support, DS nodes typically only need

- 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. Remarketing to other DSCPs/PHBs

"DSCP bleaching", i.e., setting the DSCP to ‘000000’ (default PHB) is NOT RECOMMENDED for this PHB. This may cause effects that are in contrast to the original intent in protecting BE traffic from LE traffic (no harm property). In the case that a DS domain does not support the LE PHB, its nodes SHOULD treat LE marked packets with the default PHB instead (by mapping the LE DSCP to the default PHB), but they SHOULD do so without 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., forwarded as best-effort traffic) they get a better forwarding treatment than expected. For some applications and services, it is favorable if the transmission is finished earlier than expected. However, in some cases it may be against the original intention of the LE PHB user to strictly send the traffic only if otherwise unused resources are available. In the case that LE traffic is mapped to the default PHB, LE traffic may compete with BE traffic for the same resources and thus adversely affect the original BE aggregate. Applications that want to ensure the lower precedence compared to BE traffic even in such cases SHOULD use additionally a corresponding Lower-than-Best-Effort transport protocol [RFC6297], e.g., LEDBAT [RFC6817].

A DS domain that still uses DSCP CS1 for marking LE traffic (including Low Priority-Data as defined in [RFC4594] or the old definition in [RFC3662]) SHOULD remark traffic to the LE DSCP ‘000001’ at the egress to the next DS domain. This increases the probability that the DSCP is preserved end-to-end, whereas a CS1 marked packet may be remarked by the default DSCP if the next domain is applying Diffserv-Interconnection [RFC8100].

9. Multicast Considerations

Basically, the multicast considerations in [RFC3754] apply. However, using the Lower Effort PHB for multicast requires paying special attention to the way how packets get replicated inside routers. Due to multicast packet replication, resource contention may actually occur even before a packet is forwarded to its output port and in the worst case, these forwarding resources are missing for higher prioritized multicast or even unicast packets.

Several forward error correction coding schemes such as fountain codes (e.g., [RFC5053]) allow reliable data delivery even in environments with a potential high amount of packet loss in transmission. When used for example over satellite links or other broadcast media, this means that receivers that lose 80% of packets in transmission simply need 5 times as long to receive the complete data than those receivers experiencing no loss (without any receiver feedback required).

Superficially viewed, it may sound very attractive to use IP multicast with the LE PHB to build this type of opportunistic reliable distribution in IP networks, but it can only be usefully
deployed with routers that do not experience forwarding/replication resource starvation when a large amount of packets (virtually) need to be replicated to links where the LE queue is full.

Thus, packet replication of LE marked packets should consider the situation at the respective output links: it is a waste of internal forwarding resources if a packet is replicated to output links that have no resources left for LE forwarding. In those cases a packet would have been replicated just to be dropped immediately after finding a filled LE queue at the respective output port. Such behavior could be avoided for example by using a conditional internal packet replication: a packet would then only be replicated in case the output link is not fully used. This conditional replication, however, is probably not widely implemented.

While the resource contention problem caused by multicast packet replication is also true for other Diffserv PHBs, LE forwarding is special, because often it is assumed that LE packets only get forwarded in case of available resources at the output ports. The previously mentioned redundancy data traffic could nicely use the varying available residual bandwidth being utilized by LE PHB, but only if the specific requirements stated above for conditional replication in the internal implementation of the network devices are considered.

10. The Update to RFC 4594

[RFC4594] recommended to use CS1 as codepoint in section 4.10, whereas CS1 was defined in [RFC2474] to have a higher precedence than CS0, i.e., the default PHB. Consequently, Diffserv domains implementing CS1 according to [RFC2474] will cause a priority inversion for LE packets that contradicts with the original purpose of LE. Therefore, every occurrence of the CS1 DSCP is replaced by the LE DSCP.

Changes:

- 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."
This document removes the following line of RFC 4594, Section 4.10: "The RECOMMENDED DSCP marking is CS1 (Class Selector 1)." and replaces this with the following text: "The RECOMMENDED DSCP marking is LE (Lower Effort), which replaces the prior recommendation for CS1 (Class Selector 1) marking."

11. The Update to RFC 8325

Section 4.2.10 of RFC 8325 [RFC8325] specifies "[RFC3662] and [RFC4594] both recommend Low-Priority Data be marked CS1 DSCP." which is updated to "[RFC3662] recommends that Low-Priority Data be marked CS1 DSCP. [RFC4594] as updated by [RFCXXXX] recommends Low-Priority Data be marked LE DSCP."

This document removes the following paragraph of RFC 8325, Section 4.2.10 because this document makes the anticipated change: "Note: This marking recommendation may change in the future, as [LE-PHB] defines a Lower Effort (LE) PHB for Low-Priority Data traffic and recommends an additional DSCP for this traffic."

Section 4.2.10 of RFC 8325 [RFC8325] specifies "therefore, it is RECOMMENDED to map Low-Priority Data traffic marked CS1 DSCP to UP 1" which is updated to "therefore, it is RECOMMENDED to map Low-Priority Data traffic marked with LE DSCP or legacy CS1 DSCP to UP 1"

This update to RFC 8325 replaces the following entry from figure 1:

<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>
<tbody>
<tr>
<td>Low-Priority Data (legacy)</td>
<td>CS1</td>
<td>RFC 3662</td>
<td>1</td>
<td>AC_BK (Background)</td>
</tr>
</tbody>
</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></td>
<td>LE (1)</td>
<td>DF</td>
<td>AF42, AF43</td>
<td>AF41, AF42</td>
</tr>
<tr>
<td></td>
<td>LE (1)</td>
<td>DF</td>
<td>AF32, AF33</td>
<td>AF31, AF32</td>
</tr>
<tr>
<td></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


<table>
<thead>
<tr>
<th>Reference</th>
<th>Title</th>
<th>Status</th>
<th>Date</th>
<th>URL</th>
</tr>
</thead>
</table>
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 Section 12 and to delete it afterwards.
o Incorporated Mirja’s and Benjamin’s suggestions.

o Editorial suggested by Gorry: In case => In the case that

Changes in Version 09:

o Incorporated comments from IETF Last Call:
  * from Olivier Bonaventure: added a bit of text for session resumption and congestion control aspects as well as ECN usage.
  * from Kyle Rose: Revised privacy considerations text in Security Considerations Section

Changes in Version 08:

o revised two sentences as suggested by Spencer Dawkins

Changes in Version 07:

o revised some text for clarification according to comments from Spencer Dawkins

Changes in Version 06:

o added Multicast Considerations section with input from Toerless Eckert

o incorporated suggestions by David Black with respect to better reflect legacy CS1 handling

Changes in Version 05:

o added scavenger service class into abstract

o added some more history

o added reference for "Myth of Over-Provisioning" in RFC3439 and references to presentations w.r.t. codepoint choices

o added text to update draft-ietf-tsvwg-rtcweb-qos

o revised text on congestion control in case of remarking to BE

o added reference to DSCP measurement talk @IETF99

o small typo fixes
Changes in Version 04:

- 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
Abstract

The Stream Control Transmission Protocol (SCTP) provides a reliable communications channel between two end-hosts in many ways similar to the Transmission Control Protocol (TCP). With the widespread deployment of Network Address Translators (NAT), specialized code has been added to NAT 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.

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 ............................................. 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>Support</td>
<td>No Support</td>
<td>Limited</td>
</tr>
<tr>
<td>Support</td>
<td>No Support</td>
<td>Support</td>
<td>None</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
+--------+ Address      +--------+ Address
|  SCTP  | ------+ NAT ------| Internet | ------+ SCTP  |
|endpoint|      |      |      |       |      |      |    |
|    A   |      |      |      |      |      |      |    |
+--------+      |      |      |      |      |      |      |
| Internal Port | VTag |     |      |      |      |      |    |
|      |      |      |      |      |      |      |    |
```

Figure 1: Basic network setup

A variation of this case is shown below, i.e., multiple NAT devices in a single path:
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:

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.

```
+--------+          +-----+           /        \\           +--------+
| Host A | <------> | NAT | <------> | Internet | <------> | Host B |
\-----\          \-----\           \      /           \-----\|
Priv-Addr:Int-Port ------> Ext-Addr:Ext-Port
                           Ext-VTag
```

Translate To:

```
Pub-Addr:Int-Port ------> Ext-Addr:Ext-Port
                        Ext-VTag
```

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.

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

<table>
<thead>
<tr>
<th>Type = 6</th>
<th>Reserved</th>
<th>M</th>
<th>T</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

The ERROR chunk defined in [RFC4960] is extended to add the new ‘M-bit’. The M-bit indicates to the receiver of the ERROR chunk that the chunk was not generated by the peer SCTP endpoint, but instead by a middle box.

[NOTE to RFC-Editor:
Assignment of M-bit to be confirmed by IANA.
]

5.2.  New Error Causes

This section defines the new error causes added by this document.

5.2.1.  VTag and Port Number Collision Error Cause

Cause Code: 2 bytes (unsigned integer)
This field holds the IANA defined cause code for the 'VTag and Port Number Collision' Error Cause. IANA is requested to assign the value 0x00B0 for this cause code.

Cause Length: 2 bytes (unsigned integer)
This field holds the length in bytes of the error cause. The value MUST be the length of the Cause-Specific Information plus 4.

Chunk: variable length
The Cause-Specific Information is filled with the chunk that caused this error. This can be an INIT, INIT-ACK, or ASCONF chunk. Note that if the entire chunk will not fit in the ERROR chunk or ABORT chunk being sent then the bytes that do not fit are truncated.

[NOTE to RFC-Editor:
Assignment of cause code to be confirmed by IANA.
]

5.2.2. Missing State Error Cause

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

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.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Type = 0xC007         |         Length = 4            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Parameter Type = 0xC008                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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 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.
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 can not 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.

```
init[Initiate-Tag = 1234]
10.0.0.1:1 --------> 203.0.113.1:2
Ext-VTtag = 0
```

A NAT entry is created, the source address is substituted and the packet is sent on:
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
```

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>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>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.
Internet-Draft  SCTP NAT Support  July 2019

+--------+  /--\--\   /-|Router 1| \  \\
| Host | <-----> | NAT | <-> | Internet | == | Host |  \\
+------+         +-----+      /        \/    +--------+  \\
|   A  |         |  Int  |  Int   |   Ext    |   Ext  |    Priv   |
+------+                       \--/\--/      \-|Router 2|-/  +------+
+--------+  \------+

INIT-ACK[Initiate-tag = 5678, IP-Addr = 203.0.113.129]
192.0.2.1:1 <-------------------------- 203.0.113.1:2
Int-VTag = 1234

NAT does need to change the table for second address:

+---------+--------+----------+--------+-----------+
| NAT     |  Int    |  Int    |   Ext   |   Ext    |    Priv   |
| VTag    |  Port   |   VTag  |   Port  |   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.
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.
**INIT**[Initiate-Tag = 1234]
10.0.0.1:1 --------> 203.0.113.1:2
Ext-VTag = 0

NAT 1 creates entry:

<table>
<thead>
<tr>
<th>VTag</th>
<th>Port</th>
<th>VTag</th>
<th>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
ExtVTag = 0

Host B includes its second address in the INIT-ACK, which results in two NAT entries in NAT 1.
INIT-ACK[Initiate-Tag = 5678, IP-Addr = 203.0.113.129]
192.0.2.1:1 <----------------------- 203.0.113.1:2
Int-VTag = 1234

NAT 1 does not need to update the table for second address:

<table>
<thead>
<tr>
<th>NAT 1</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>1234</td>
<td>1</td>
<td>5678</td>
<td>2</td>
<td>10.0.0.1</td>
<td></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.
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:
NAT 2
+---------+--------+----------+--------+-----------+
<table>
<thead>
<tr>
<th>VTag</th>
<th>Port</th>
<th>VTag</th>
<th>Port</th>
<th>Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>1</td>
<td>5678</td>
<td>2</td>
<td>10.1.0.1</td>
</tr>
</tbody>
</table>
+---------+--------+----------+--------+-----------+

ASCONF [ADD-IP,Int-VTag=1234, Ext-VTag = 5678]
192.0.2.129:1 -----------------> 203.0.113.129:2
Ext-VTag = 5678

ASCONF-ACK
192.0.2.129:1 <------------------ 203.0.113.129:2
Int-VTag = 1234

ASCONF-ACK
10.1.0.1:1 <----- 203.0.113.129:2
Int-VTag = 1234

7.4. NAT Loses Its State

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.

+--------+              +-----+         /        \
| Host A | <----------> | NAT | <----> | Internet | <----> | Host B |
+--------+              +-----+         \        /         +--------+
\--/\--/

DATA
10.0.0.1:1 ----------> 203.0.113.1:2
Ext-VTag = 5678

The NAT device cannot find entry for the association. It sends ERROR message with the M-Bit set and the cause "NAT state missing".
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.

```
+---------+--------+----------+--------+-----------+
| NAT     |  Int   |  Int     |   Ext   |   Ext     |
| VTag    | Port   | VTag     | Port   |  Addr     |
| 1234    | 1      | 5678     | 2      | 10.0.0.1  |
+---------+--------+----------+--------+-----------+
```

```
ASCONF [ADD-IP,DELETE-IP,Int-VTag=1234, Ext-VTag = 5678]
10.0.0.1:1 -------------> 203.0.113.129:2
Ext-VTag = 5678
```

Host B adds the new source address and deletes all former entries.

```
+---------+--------+----------+--------+-----------+
| NAT     |  Int   |  Int     |   Ext   |   Ext     |
| VTag    | Port   | VTag     | Port   |  Addr     |
+---------+--------+----------+--------+-----------+
```

```
ASCONF [ADD-IP,DELETE-IP,Int-VTag=1234, Ext-VTag = 5678]
192.0.2.2:1 ----------------> 203.0.113.129:2
Ext-VTag = 5678
```
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:

<table>
<thead>
<tr>
<th>NAT A</th>
<th>Int</th>
<th>Int</th>
<th>Ext</th>
<th>Ext</th>
<th>Priv</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VTag</td>
<td>Port</td>
<td>VTag</td>
<td>Port</td>
<td>Addr</td>
</tr>
<tr>
<td>1234</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>10.0.0.1</td>
<td></td>
</tr>
</tbody>
</table>

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.

NAT B processes INIT. As the outgoing INIT of Host A has already created an entry, the entry is found and updated:
VTag != Int-VTag, but Ext-VTag == 0, find entry.

<table>
<thead>
<tr>
<th>NAT A</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>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:

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 is a compilation of issues found since the publication of RFC4960 in September 2007 based on experience with implementing, testing, and using SCTP along with the suggested fixes. This document provides deltas to RFC4960 and is organized in a time ordered way. The issues are listed in the order they were brought up. Because some text is changed several times the last delta in the text is the one which should be applied. In addition to the delta a description of the problem and the details of the solution are also provided.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 25, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents
Table of Contents

1. Introduction ................................................. 3
2. Conventions .................................................. 4
3. Corrections to RFC 4960 ....................................... 4
   3.1. Path Error Counter Threshold Handling .................. 4
   3.2. Upper Layer Protocol Shutdown Request Handling ........ 5
   3.3. Registration of New Chunk Types ........................ 6
   3.4. Variable Parameters for INIT Chunks ..................... 7
   3.5. CRC32c Sample Code on 64-bit Platforms ................. 8
   3.6. Endpoint Failure Detection ............................ 9
   3.7. Data Transmission Rules ............................... 10
   3.8. T1-Cookie Timer ....................................... 11
   3.9. Miscellaneous Typos .................................... 12
   3.10. CRC32c Sample Code .................................... 19
   3.11. partial_bytes_acked after T3-rtx Expiration ........... 20
   3.12. Order of Adjustments of partial_bytes_acked and cwnd ... 21
   3.13. HEARTBEAT ACK and the association error counter ....... 22
   3.15. Transmittal in Fast Recovery .......................... 24
   3.16. Initial Value of ssthresh ............................. 25
   3.17. Automatically Confirmed Addresses ..................... 26
   3.18. Only One Packet after Retransmission Timeout .......... 27
   3.19. INIT ACK Path for INIT in COOKIE-WAIT State .......... 28
   3.20. Zero Window Probing and Unreachable Primary Path ...... 29
   3.21. Normative Language in Section 10 ........................ 30
   3.22. Increase of partial_bytes_acked in Congestion Avoidance 33
   3.23. Inconsistency in Notifications Handling ................ 34
   3.24. SACK.Delay Not Listed as a Protocol Parameter .......... 40
   3.25. Processing of Chunks in an Incoming SCTP Packet ........ 42
   3.26. CWND Increase in Congestion Avoidance Phase .......... 43
   3.27. Refresh of cwnd and ssthresh after Idle Period ....... 46
   3.28. Window Updates After Receiver Window Opens Up ....... 47
   3.29. Path of DATA and Reply Chunks ........................ 48
   3.30. Outstanding Data, Flightsize and Data In Flight Key Terms 50
   3.31. CWND Degradation due to Max.Burst ..................... 52
   3.32. Reduction of RTO.Initial ............................. 53
   3.33. Ordering of Bundled SACK and ERROR Chunks .............. 55
   3.34. Undefined Parameter Returned by RECEIVE Primitive .... 56
   3.35. DSCP Changes ........................................ 57
1. Introduction

This document contains a compilation of all defects found up until
the publication of this document for [RFC4960] specifying the Stream
Control Transmission Protocol (SCTP). These defects may be of an
editorial or technical nature. This document may be thought of as a
companion document to be used in the implementation of SCTP to
clarify errors in the original SCTP document.

This document provides a history of the changes that will be compiled
into a BIS document for [RFC4960]. It is structured similar to
[RFC4460].

Each error will be detailed within this document in the form of:

- The problem description,
- The text quoted from [RFC4960],
- The replacement text that should be placed into an upcoming BIS
document,
- A description of the solution.

Note that when reading this document one must use care to assure that
a field or item is not updated further on within the document. Since
this document is a historical record of the sequential changes that
have been found necessary at various inter-op events and through discussion on the list, the last delta in the text is the one which should be applied.

2. Conventions

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

3. Corrections to RFC 4960

[NOTE to RFC-Editor:

References to obsoleted RFCs are in OLD TEXT sections and have the corresponding references to the obsoleting RFCs in the NEW TEXT sections. In addition to this, there are some references to the obsoleted [RFC2960], which are intended.

]

3.1. Path Error Counter Threshold Handling

3.1.1. Description of the Problem

The handling of the 'Path.Max.Retrans' parameter is described in Section 8.2 and Section 8.3 of [RFC4960] in an inconsistent way. Whereas Section 8.2 describes that a path is marked inactive when the path error counter exceeds the threshold, Section 8.3 says the path is marked inactive when the path error counter reaches the threshold.

This issue was reported as an Errata for [RFC4960] with Errata ID 1440.

3.1.2. Text Changes to the Document
Old text: (Section 8.3)

When the value of this counter reaches the protocol parameter 'Path.Max.Retrans', the endpoint should mark the corresponding destination address as inactive if it is not so marked, and may also optionally report to the upper layer the change of reachability of this destination address. After this, the endpoint should continue HEARTBEAT on this destination address but should stop increasing the counter.

New text: (Section 8.3)

When the value of this counter exceeds the protocol parameter 'Path.Max.Retrans', the endpoint SHOULD mark the corresponding destination address as inactive if it is not so marked, and MAY also optionally report to the upper layer the change of reachability of this destination address. After this, the endpoint SHOULD continue HEARTBEAT on this destination address but SHOULD stop increasing the counter.

This text has been modified by multiple errata. It is further updated in Section 3.23.

3.1.3. Solution Description

The intended state change should happen when the threshold is exceeded.

3.2. Upper Layer Protocol Shutdown Request Handling

3.2.1. Description of the Problem

Section 9.2 of [RFC4960] describes the handling of received SHUTDOWN chunks in the SHUTDOWN-RECEIVED state instead of the handling of shutdown requests from its upper layer in this state.

This issue was reported as an Errata for [RFC4960] with Errata ID 1574.

3.2.2. Text Changes to the Document
Once an endpoint has reached the SHUTDOWN-RECEIVED state, it MUST NOT send a SHUTDOWN in response to a ULP request, and should discard subsequent SHUTDOWN chunks.

Once an endpoint has reached the SHUTDOWN-RECEIVED state, it MUST ignore ULP shutdown requests, but MUST continue responding to SHUTDOWN chunks from its peer.

This text is in final form, and is not further updated in this document.

3.2.3. Solution Description

The text never intended the SCTP endpoint to ignore SHUTDOWN chunks from its peer. If it did, the endpoints could never gracefully terminate associations in some cases.

3.3. Registration of New Chunk Types

3.3.1. Description of the Problem

Section 14.1 of [RFC4960] should deal with new chunk types, however, the text refers to parameter types.

This issue was reported as an Errata for [RFC4960] with Errata ID 2592.

3.3.2. Text Changes to the Document
The assignment of new chunk parameter type codes is done through an IETF Consensus action, as defined in [RFC2434]. Documentation of the chunk parameter MUST contain the following information:

This text has been modified by multiple errata. It is further updated in Section 3.43.

3.3.3. Solution Description

Refer to chunk types as intended and change reference to [RFC8126].

3.4. Variable Parameters for INIT Chunks

3.4.1. Description of the Problem

Newlines in wrong places break the layout of the table of variable parameters for the INIT chunk in Section 3.3.2 of [RFC4960].

This issue was reported as an Errata for [RFC4960] with Errata ID 3291 and Errata ID 3804.

3.4.2. Text Changes to the Document
Old text: (Section 3.3.2)

<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>IPv6 Address (Note 1)</td>
</tr>
<tr>
<td>Cookie Preservative</td>
<td>Optional</td>
<td>6</td>
<td>Cookie Preservative</td>
</tr>
<tr>
<td>Reserved for ECN Capable (Note 2)</td>
<td>Optional</td>
<td>9</td>
<td>Reserved for ECN Capable (Note 2)</td>
</tr>
<tr>
<td>Host Name Address (Note 3)</td>
<td>Optional</td>
<td>32768 (0x8000)</td>
<td>Host Name Address (Note 3)</td>
</tr>
<tr>
<td>Supported Address Types (Note 4)</td>
<td>Optional</td>
<td>11</td>
<td>Supported Address Types (Note 4)</td>
</tr>
</tbody>
</table>

New text: (Section 3.3.2)

<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>IPv6 Address (Note 1)</td>
</tr>
<tr>
<td>Cookie Preservative</td>
<td>Optional</td>
<td>6</td>
<td>Cookie Preservative</td>
</tr>
<tr>
<td>Reserved for ECN Capable (Note 2)</td>
<td>Optional</td>
<td>9</td>
<td>Reserved for ECN Capable (Note 2)</td>
</tr>
<tr>
<td>Host Name Address (Note 3)</td>
<td>Optional</td>
<td>32768 (0x8000)</td>
<td>Host Name Address (Note 3)</td>
</tr>
<tr>
<td>Supported Address Types (Note 4)</td>
<td>Optional</td>
<td>11</td>
<td>Supported Address Types (Note 4)</td>
</tr>
</tbody>
</table>

This text is in final form, and is not further updated in this document.

3.4.3. Solution Description

Fix the formatting of the table.

3.5. CRC32c Sample Code on 64-bit Platforms

3.5.1. Description of the Problem

The sample code for computing the CRC32c provided in [RFC4960] assumes that a variable of type unsigned long uses 32 bits. This is not true on some 64-bit platforms (for example the ones using LP64).

This issue was reported as an Errata for [RFC4960] with Errata ID 3423.

3.5.2. Text Changes to the Document
unsigned long
generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    unsigned long crc32 = 0xffffffffL;

This text has been modified by multiple errata. It is further updated in Section 3.10 and in Section 3.46.

3.5.3. Solution Description

Use 0xffffffffL instead of ˜0L which gives the same value on platforms using 32 bits or 64 bits for variables of type unsigned long.

3.6. Endpoint Failure Detection

3.6.1. Description of the Problem

The handling of the association error counter defined in Section 8.1 of [RFC4960] can result in an association failure even if the path used for data transmission is available, but idle.

This issue was reported as an Errata for [RFC4960] with Errata ID 3788.

3.6.2. Text Changes to the Document
An endpoint shall keep a counter on the total number of consecutive retransmissions to its peer (this includes retransmissions to all the destination transport addresses of the peer if it is multi-homed), including unacknowledged HEARTBEAT chunks.

An endpoint SHOULD keep a counter on the total number of consecutive retransmissions to its peer (this includes data retransmissions to all the destination transport addresses of the peer if it is multi-homed), including the number of unacknowledged HEARTBEAT chunks observed on the path which is currently used for data transfer. Unacknowledged HEARTBEAT chunks observed on paths different from the path currently used for data transfer SHOULD NOT increment the association error counter, as this could lead to association closure even if the path which is currently used for data transfer is available (but idle).

This text has been modified by multiple errata. It is further updated in Section 3.23.

3.6.3. Solution Description

A more refined handling for the association error counter is defined.

3.7. Data Transmission Rules

3.7.1. Description of the Problem

When integrating the changes to Section 6.1 A) of [RFC2960] as described in Section 2.15.2 of [RFC4460] some text was duplicated and became the final paragraph of Section 6.1 A) of [RFC4960].

This issue was reported as an Errata for [RFC4960] with Errata ID 4071.

3.7.2. Text Changes to the Document
The sender MUST also have an algorithm for sending new DATA chunks to avoid silly window syndrome (SWS) as described in [RFC0813]. The algorithm can be similar to the one described in Section 4.2.3.4 of [RFC1122].

However, regardless of the value of rwnd (including if it is 0), the data sender can always have one DATA chunk in flight to the receiver if allowed by cwnd (see rule B below). This rule allows the sender to probe for a change in rwnd that the sender missed due to the SACK having been lost in transit from the data receiver to the data sender.

This text is in final form, and is not further updated in this document.

3.7.3. Solution Description

Last paragraph of Section 6.1 A) removed as intended in Section 2.15.2 of [RFC4460].

3.8. T1-Cookie Timer

3.8.1. Description of the Problem

Figure 4 of [RFC4960] illustrates the SCTP association setup. However, it incorrectly shows that the T1-init timer is used in the COOKIE-ECHOED state whereas the T1-cookie timer should have been used instead.

This issue was reported as an Errata for [RFC4960] with Errata ID 4400.
3.8.2. Text Changes to the Document

---------
Old text: (Section 5.1.6, Figure 4)
---------

COOKIE ECHO [Cookie_Z] ------\
(Start T1-init timer) \       
(Enter COOKIE-ECHOED state) \---> (build TCB enter ESTABLISHED state)
                                            /---- COOKIE-ACK
                                            /
(Cancel T1-init timer, <-----/ Enter ESTABLISHED state)

---------
New text: (Section 5.1.6, Figure 4)
---------

COOKIE ECHO [Cookie_Z] ------\
(Start T1-cookie timer) \     
(Enter COOKIE-ECHOED state) \---> (build TCB enter ESTABLISHED state)
                                            /---- COOKIE-ACK
                                            /
(Cancel T1-cookie timer, <---/ Enter ESTABLISHED state)

This text has been modified by multiple errata. It is further updated in Section 3.9.

3.8.3. Solution Description

Change the figure such that the T1-cookie timer is used instead of the T1-init timer.

3.9. Miscellaneous Typos

3.9.1. Description of the Problem

While processing [RFC4960] some typos were not caught.

One typo was reported as an Errata for [RFC4960] with Errata ID 5003.
3.9.2. Text Changes to the Document

---------
Old text: (Section 1.6)
---------
Transmission Sequence Numbers wrap around when they reach $2^{32} - 1$. That is, the next TSN a DATA chunk MUST use after transmitting TSN = $2^{32} - 1$ is TSN = 0.

---------
New text: (Section 1.6)
---------
Transmission Sequence Numbers wrap around when they reach $2^{32} - 1$. That is, the next TSN a DATA chunk MUST use after transmitting TSN = $2^{32} - 1$ is TSN = 0.

This text is in final form, and is not further updated in this document.

---------
Old text: (Section 3.3.10.9)
---------
No User Data: This error cause is returned to the originator of a DATA chunk if a received DATA chunk has no user data.

---------
New text: (Section 3.3.10.9)
---------
No User Data: This error cause is returned to the originator of a DATA chunk if a received DATA chunk has no user data.

This text is in final form, and is not further updated in this document.
Old text: (Section 6.7, Figure 9)

Endpoint A                                    Endpoint Z
{App sends 3 messages; strm 0} DATA [TSN=6, Strm=0, Seq=2] ----------
-----> (ack delayed) (Start T3-rtx timer)
DATA [TSN=7, Strm=0, Seq=3] ----------> X (lost)
DATA [TSN=8, Strm=0, Seq=4] ---------------> (gap detected, immediately send ack)
            /----- SACK [TSN Ack=6, Block=1, Start=2, End=2]
<-----/ (remove 6 from out-queue, and mark 7 as "1" missing report)

New text: (Section 6.7, Figure 9)

Endpoint A                                    Endpoint Z
{App sends 3 messages; strm 0}
DATA [TSN=6, Strm=0, Seq=2] ---------------> (ack delayed)
(Start T3-rtx timer)
DATA [TSN=7, Strm=0, Seq=3] ----------> X (lost)
DATA [TSN=8, Strm=0, Seq=4] ---------------> (gap detected, immediately send ack)
            /----- SACK [TSN Ack=6, Block=1, Start=2, End=2]
<-----/ (remove 6 from out-queue, and mark 7 as "1" missing report)

This text is in final form, and is not further updated in this document.
An endpoint bundles chunks by simply including multiple chunks in one outbound SCTP packet. The total size of the resultant IP datagram, including the SCTP packet and IP headers, MUST be less than or equal to the current PMTU.

This text is in final form, and is not further updated in this document.

o Receive Unacknowledged Message

Format: RECEIVE_UNACKED(data retrieval id, buffer address, buffer size, [,stream id] [, stream sequence number] [,partial flag] [,payload protocol-id])

This text is in final form, and is not further updated in this document.
M) Set Protocol Parameters

Format: SETPROTOCOLPARAMETERS(association id,
[,destination transport address,]
protocol parameter list)

---

New text: (Section 10.1 M)
---

M) Set Protocol Parameters

Format: SETPROTOCOLPARAMETERS(association id,
[destination transport address,]
protocol parameter list)

This text is in final form, and is not further updated in this document.

---

Old text: (Appendix C)
---

ICMP2) An implementation MAY ignore all ICMPv6 messages where the type field is not "Destination Unreachable", "Parameter Problem", or "Packet Too Big".

---

New text: (Appendix C)
---

ICMP2) An implementation MAY ignore all ICMPv6 messages where the type field is not "Destination Unreachable", "Parameter Problem", or "Packet Too Big".

This text is in final form, and is not further updated in this document.
ICMP7) If the ICMP message is either a v6 "Packet Too Big" or a v4 "Fragmentation Needed", an implementation MAY process this information as defined for PATH MTU discovery.

This text is in final form, and is not further updated in this document.

2) For the receiver of the COOKIE ECHO, the only CONFIRMED address is the one to which the INIT-ACK was sent.

This text is in final form, and is not further updated in this document.
Old text: (Section 5.1.6, Figure 4)

COOKIE ECHO [Cookie_Z] ------\
(Start T1-init timer)    \   \----> (build TCB enter ESTABLISHED state)
(Enter COOKIE-ECHOED state) \----> (build TCB enter ESTABLISHED state)

(Cancel T1-init timer, <-----/)
Enter ESTABLISHED state

New text: (Section 5.1.6, Figure 4)

COOKIE ECHO [Cookie_Z] ------\
(Start T1-cookie timer)    \   \----> (build TCB enter ESTABLISHED state)
(Enter COOKIE-ECHOED state) \----> (build TCB enter ESTABLISHED state)

(Cancel T1-cookie timer, <---/)
Enter ESTABLISHED state

This text has been modified by multiple errata. It includes modifications from Section 3.8. It is in final form, and is not further updated in this document.

Old text: (Section 5.2.5)

5.2.5. Handle Duplicate COOKIE-ACK.

New text: (Section 5.2.5)

5.2.5. Handle Duplicate COOKIE ACK.

This text is in final form, and is not further updated in this document.
By default, an SCTP endpoint SHOULD monitor the reachability of the idle destination transport address(es) of its peer by sending a HEARTBEAT chunk periodically to the destination transport address(es). HEARTBEAT sending MAY begin upon reaching the ESTABLISHED state and is discontinued after sending either SHUTDOWN or SHUTDOWN-ACK. A receiver of a HEARTBEAT MUST respond to a HEARTBEAT with a HEARTBEAT-ACK after entering the COOKIE-ECHOED state (INIT sender) or the ESTABLISHED state (INIT receiver), up until reaching the SHUTDOWN-SENT state (SHUTDOWN sender) or the SHUTDOWN-ACK-SENT state (SHUTDOWN receiver).

This text is in final form, and is not further updated in this document.

3.9.3. Solution Description

Typos fixed.

3.10. CRC32c Sample Code

3.10.1. Description of the Problem

The CRC32c computation is described in Appendix B of [RFC4960]. However, the corresponding sample code and its explanation appears at the end of Appendix C, which deals with ICMP handling.
3.10.2. Text Changes to the Document

Move all of Appendix C starting with the following sentence to the end of Appendix B.

The following non-normative sample code is taken from an open-source CRC generator [WILLIAMS93], using the "mirroring" technique and yielding a lookup table for SCTP CRC32c with 256 entries, each 32 bits wide.

This text has been modified by multiple errata. It includes modifications from Section 3.5. It is further updated in Section 3.46.

3.10.3. Solution Description

Text moved to the appropriate location.

3.11. partial_bytes_acked after T3-rtx Expiration

3.11.1. Description of the Problem

Section 7.2.3 of [RFC4960] explicitly states that partial_bytes_acked should be reset to 0 after packet loss detection from SACK but the same is missed for T3-rtx timer expiration.

3.11.2. Text Changes to the Document

---------
Old text: (Section 7.2.3)
---------

When the T3-rtx timer expires on an address, SCTP should perform slow start by:

\[
\text{ssthresh} = \max\left(\frac{\text{cwnd}}{2}, 4\times \text{MTU}\right)
\]
\[
\text{cwnd} = 1\times \text{MTU}
\]

---------
New text: (Section 7.2.3)
---------

When the T3-rtx timer expires on an address, SCTP SHOULD perform slow start by:

\[
\text{ssthresh} = \max\left(\frac{\text{cwnd}}{2}, 4\times \text{MTU}\right)
\]
\[
\text{cwnd} = 1\times \text{MTU}
\]
\[
\text{partial_bytes_acked} = 0
\]
This text is in final form, and is not further updated in this document.

3.11.3. Solution Description

Specify that partial_bytes_acked should be reset to 0 after T3-rtx timer expiration.

3.12. Order of Adjustments of partial_bytes_acked and cwnd

3.12.1. Description of the Problem

Section 7.2.2 of [RFC4960] likely implies the wrong order of adjustments applied to partial_bytes_acked and cwnd in the congestion avoidance phase.

3.12.2. Text Changes to the Document

---------
Old text: (Section 7.2.2)
---------
---------

- When partial_bytes_acked is equal to or greater than cwnd and before the arrival of the SACK the sender had cwnd or more bytes of data outstanding (i.e., before arrival of the SACK, flightsize was greater than or equal to cwnd), increase cwnd by MTU, and reset partial_bytes_acked to (partial_bytes_acked - cwnd).

---------
New text: (Section 7.2.2)
---------
---------

- When partial_bytes_acked is equal to or greater than cwnd and before the arrival of the SACK the sender had cwnd or more bytes of data outstanding (i.e., before arrival of the SACK, flightsize was greater than or equal to cwnd), partial_bytes_acked is reset to (partial_bytes_acked - cwnd). Next, cwnd is increased by 1*MTU.

This text has been modified by multiple errata. It is further updated in Section 3.26.

3.12.3. Solution Description

The new text defines the exact order of adjustments of partial_bytes_acked and cwnd in the congestion avoidance phase.
3.13. HEARTBEAT ACK and the association error counter

3.13.1. Description of the Problem

Section 8.1 and Section 8.3 of [RFC4960] prescribe that the receiver of a HEARTBEAT ACK must reset the association overall error counter. In some circumstances, e.g. when a router discards DATA chunks but not HEARTBEAT chunks due to the larger size of the DATA chunk, it might be better to not clear the association error counter on reception of the HEARTBEAT ACK and reset it only on reception of the SACK to avoid stalling the association.

3.13.2. Text Changes to the Document

---------
Old text: (Section 8.1)
---------

The counter shall be reset each time a DATA chunk sent to that peer endpoint is acknowledged (by the reception of a SACK) or a HEARTBEAT ACK is received from the peer endpoint.

---------
New text: (Section 8.1)
---------

The counter MUST be reset each time a DATA chunk sent to that peer endpoint is acknowledged (by the reception of a SACK). When a HEARTBEAT ACK is received from the peer endpoint, the counter SHOULD also be reset. The receiver of the HEARTBEAT ACK MAY choose not to clear the counter if there is outstanding data on the association. This allows for handling the possible difference in reachability based on DATA chunks and HEARTBEAT chunks.

This text is in final form, and is not further updated in this document.
Upon the receipt of the HEARTBEAT ACK, the sender of the HEARTBEAT should clear the error counter of the destination transport address to which the HEARTBEAT was sent, and mark the destination transport address as active if it is not so marked. The endpoint may optionally report to the upper layer when an inactive destination address is marked as active due to the reception of the latest HEARTBEAT ACK. The receiver of the HEARTBEAT ACK must also clear the association overall error count as well (as defined in Section 8.1).

This text has been modified by multiple errata. It is further updated in Section 3.23.

3.13.3. Solution Description

The new text provides a possibility to not reset the association overall error counter when a HEARTBEAT ACK is received if there are valid reasons for it.

3.14. Path for Fast Retransmission

3.14.1. Description of the Problem

[RFC4960] clearly describes where to retransmit data that is timed out when the peer is multi-homed but the same is not stated for fast retransmissions.

3.14.2. Text Changes to the Document
Furthermore, when its peer is multi-homed, an endpoint SHOULD try to retransmit a chunk that timed out to an active destination transport address that is different from the last destination address to which the DATA chunk was sent.

When its peer is multi-homed, an endpoint SHOULD send fast retransmissions to the same destination transport address where the original data was sent to. If the primary path has been changed and the original data was sent to the old primary path before the fast retransmit, the implementation MAY send it to the new primary path.

This text is in final form, and is not further updated in this document.

3.14.3. Solution Description

The new text clarifies where to send fast retransmissions.

3.15. Transmittal in Fast Recovery

3.15.1. Description of the Problem

The Fast Retransmit on Gap Reports algorithm intends that only the very first packet may be sent regardless of cwnd in the Fast Recovery phase but rule 3) of [RFC4960], Section 7.2.4, misses this clarification.

3.15.2. Text Changes to the Document
3) Determine how many of the earliest (i.e., lowest TSN) DATA chunks marked for retransmission will fit into a single packet, subject to constraint of the path MTU of the destination transport address to which the packet is being sent. Call this value K. Retransmit those K DATA chunks in a single packet. When a Fast Retransmit is being performed, the sender SHOULD ignore the value of cwnd and SHOULD NOT delay retransmission for this single packet.

This text is in final form, and is not further updated in this document.

3.15.3. Solution Description

The new text explicitly specifies to send only the first packet in the Fast Recovery phase disregarding cwnd limitations.

3.16. Initial Value of ssthresh

3.16.1. Description of the Problem

The initial value of ssthresh should be set arbitrarily high. Using the advertised receiver window of the peer is inappropriate if the peer increases its window after the handshake. Furthermore, use a higher requirements level, since not following the advice may result in performance problems.
3.16.2. Text Changes to the Document

Old text: (Section 7.2.1)

- The initial value of ssthresh MAY be arbitrarily high (for example, implementations MAY use the size of the receiver advertised window).

New text: (Section 7.2.1)

- The initial value of ssthresh SHOULD be arbitrarily high (e.g., the size of the largest possible advertised window).

This text is in final form, and is not further updated in this document.

3.16.3. Solution Description

Use the same value as suggested in [RFC5681], Section 3.1, as an appropriate initial value. Furthermore, use the same requirements level.

3.17. Automatically Confirmed Addresses

3.17.1. Description of the Problem

The Path Verification procedure of [RFC4960] prescribes that any address passed to the sender of the INIT by its upper layer is automatically CONFIRMED. This, however, is unclear if only addresses in the request to initiate association establishment are considered or any addresses provided by the upper layer in any requests (e.g. in 'Set Primary').

3.17.2. Text Changes to the Document
1) Any address passed to the sender of the INIT by its upper layer is automatically considered to be CONFIRMED.

This text is in final form, and is not further updated in this document.

3.17.3. Solution Description

The new text clarifies that only addresses provided by the upper layer in the request to initialize an association are automatically confirmed.

3.18. Only One Packet after Retransmission Timeout

3.18.1. Description of the Problem

[ RFC4960 ] is not completely clear when it describes data transmission after T3-rtx timer expiration. Section 7.2.1 does not specify how many packets are allowed to be sent after T3-rtx timer expiration if more than one packet fit into cwnd. At the same time, Section 7.2.3 has the text without normative language saying that SCTP should ensure that no more than one packet will be in flight after T3-rtx timer expiration until successful acknowledgment. It makes the text inconsistent.

3.18.2. Text Changes to the Document
Old text: (Section 7.2.1)

- The initial cwnd after a retransmission timeout MUST be no more than 1*MTU.

New text: (Section 7.2.1)

- The initial cwnd after a retransmission timeout MUST be no more than 1*MTU and only one packet is allowed to be in flight until successful acknowledgement.

This text is in final form, and is not further updated in this document.

3.18.3. Solution Description

The new text clearly specifies that only one packet is allowed to be sent after T3-rtx timer expiration until successful acknowledgement.

3.19. INIT ACK Path for INIT in COOKIE-WAIT State

3.19.1. Description of the Problem

In case of an INIT received in the COOKIE-WAIT state [RFC4960] prescribes to send an INIT ACK to the same destination address to which the original INIT has been sent. This text does not address the possibility of the upper layer to provide multiple remote IP addresses while requesting the association establishment. If the upper layer has provided multiple IP addresses and only a subset of these addresses are supported by the peer then the destination address of the original INIT may be absent in the incoming INIT and sending INIT ACK to that address is useless.

3.19.2. Text Changes to the Document
Old text: (Section 5.2.1)

Upon receipt of an INIT in the COOKIE-WAIT state, an endpoint MUST respond with an INIT ACK using the same parameters it sent in its original INIT chunk (including its Initiate Tag, unchanged). When responding, the endpoint MUST send the INIT ACK back to the same address that the original INIT (sent by this endpoint) was sent.

New text: (Section 5.2.1)

Upon receipt of an INIT in the COOKIE-WAIT state, an endpoint MUST respond with an INIT ACK using the same parameters it sent in its original INIT chunk (including its Initiate Tag, unchanged). When responding, the following rules MUST be applied:

1) The INIT ACK MUST only be sent to an address passed by the upper layer in the request to initialize the association.

2) The INIT ACK MUST only be sent to an address reported in the incoming INIT.

3) The INIT ACK SHOULD be sent to the source address of the received INIT.

This text is in final form, and is not further updated in this document.

3.19.3. Solution Description

The new text requires sending INIT ACK to a destination address that is passed by the upper layer and reported in the incoming INIT. If the source address of the INIT meets these conditions, sending the INIT ACK to the source address of the INIT is the preferred behavior.

3.20. Zero Window Probing and Unreachable Primary Path

3.20.1. Description of the Problem

Section 6.1 of [RFC4960] states that when sending zero window probes, SCTP should neither increment the association counter nor increment the destination address error counter if it continues to receive new packets from the peer. However, the reception of new packets from the peer does not guarantee the peer’s reachability and, if the destination address becomes unreachable during zero window probing,
SCTP cannot get an updated rwnd until it switches the destination address for probes.

3.20.2. Text Changes to the Document

-------
Old text: (Section 6.1)
-------

If the sender continues to receive new packets from the receiver while doing zero window probing, the unacknowledged window probes should not increment the error counter for the association or any destination transport address. This is because the receiver MAY keep its window closed for an indefinite time. Refer to Section 6.2 on the receiver behavior when it advertises a zero window.

-------
New text: (Section 6.1)
-------

If the sender continues to receive SACKs from the peer while doing zero window probing, the unacknowledged window probes SHOULD NOT increment the error counter for the association or any destination transport address. This is because the receiver could keep its window closed for an indefinite time. Section 6.2 describes the receiver behavior when it advertises a zero window.

This text is in final form, and is not further updated in this document.

3.20.3. Solution Description

The new text clarifies that if the receiver continues to send SACKs, the sender of probes should not increment the error counter of the association and the destination address even if the SACKs do not acknowledge the probes.

3.21. Normative Language in Section 10

3.21.1. Description of the Problem

Section 10 of [RFC4960] is informative and, therefore, normative language such as MUST and MAY cannot be used there. However, there are several places in Section 10 where MUST and MAY are used.
3.21.2. Text Changes to the Document

Old text: (Section 10.1 E))

- no-bundle flag - instructs SCTP not to bundle this user data with other outbound DATA chunks. SCTP MAY still bundle even when this flag is present, when faced with network congestion.

New text: (Section 10.1 E))

- no-bundle flag - instructs SCTP not to bundle this user data with other outbound DATA chunks. SCTP may still bundle even when this flag is present, when faced with network congestion.

This text is in final form, and is not further updated in this document.

Old text: (Section 10.1 G))

- 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 Receive contains a partial delivery of the whole message. When this flag is set, the stream id and Stream Sequence Number MUST accompany this receive. When this flag is set to 0, it indicates that no more deliveries will be received for this Stream Sequence Number.

New text: (Section 10.1 G))

- 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 must accompany this primitive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.
This text is in final form, and is not further updated in this document.

--------
Old text: (Section 10.1 N))
--------

- 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 MUST accompany this receive. When this flag is set to 0, it indicates that no more deliveries will be received for this Stream Sequence Number.

--------
New text: (Section 10.1 N))
--------

- 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 must accompany this primitive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.

This text is in final form, and is not further updated in this document.
Old text: (Section 10.1 O))

- 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 MUST accompany this receive. When this flag is set to 0, it indicates that no more deliveries will be received for this Stream Sequence Number.

New text: (Section 10.1 O))

- 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 must accompany this primitive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.

This text is in final form, and is not further updated in this document.

3.21.3. Solution Description

The normative language is removed from Section 10. In addition, the consistency of the text has been improved.

3.22. Increase of partial_bytes_acked in Congestion Avoidance

3.22.1. Description of the Problem

Two issues have been discovered with the partial_bytes_acked handling described in Section 7.2.2 of [RFC4960]:

- If the Cumulative TSN Ack Point is not advanced but the SACK chunk acknowledges new TSNs in the Gap Ack Blocks, these newly acknowledged TSNs are not considered for partial_bytes_acked although these TSNs were successfully received by the peer.
o Duplicate TSNs are not considered in partial_bytes_ackeds although they confirm that the DATA chunks were successfully received by the peer.

3.22.2. Text Changes to the Document

--------
Old text: (Section 7.2.2)
--------

o Whenever cwnd is greater than ssthresh, upon each SACK arrival that advances the Cumulative TSN Ack Point, increase partial_bytes_acked by the total number of bytes of all new chunks acknowledged in that SACK including chunks acknowledged by the new Cumulative TSN Ack and by Gap Ack Blocks.

--------
New text: (Section 7.2.2)
--------

o Whenever cwnd is greater than ssthresh, upon each SACK arrival, increase partial_bytes_acked by the total number of bytes of all new chunks acknowledged in that SACK including chunks acknowledged by the new Cumulative TSN Ack, by Gap Ack Blocks and by the number of bytes of duplicated chunks reported in Duplicate TSNs.

This text has been modified by multiple errata. It is further updated in Section 3.26.

3.22.3. Solution Description

Now partial_bytes_acked is increased by TSNs reported as duplicated as well as TSNs newly acknowledged in Gap Ack Blocks even if the Cumulative TSN Ack Point is not advanced.

3.23. Inconsistency in Notifications Handling

3.23.1. Description of the Problem

[RFC4960] uses inconsistent normative and non-normative language when describing rules for sending notifications to the upper layer. E.g. Section 8.2 of [RFC4960] says that when a destination address becomes inactive due to an unacknowledged DATA chunk or HEARTBEAT chunk, SCTP SHOULD send a notification to the upper layer while Section 8.3 of [RFC4960] says that when a destination address becomes inactive due to an unacknowledged HEARTBEAT chunk, SCTP may send a notification to the upper layer.
This makes the text inconsistent.

3.23.2. Text Changes to the Document

---------
Old text: (Section 8.1)
---------

An endpoint shall keep a counter on the total number of consecutive retransmissions to its peer (this includes retransmissions to all the destination transport addresses of the peer if it is multi-homed), including unacknowledged HEARTBEAT chunks.

---------
New text: (Section 8.1)
---------

An endpoint SHOULD keep a counter on the total number of consecutive retransmissions to its peer (this includes data retransmissions to all the destination transport addresses of the peer if it is multi-homed), including the number of unacknowledged HEARTBEAT chunks observed on the path which currently is used for data transfer. Unacknowledged HEARTBEAT chunks observed on paths different from the path currently used for data transfer SHOULD NOT increment the association error counter, as this could lead to association closure even if the path which currently is used for data transfer is available (but idle). If the value of this counter exceeds the limit indicated in the protocol parameter 'Association.Max.Retrans', the endpoint SHOULD consider the peer endpoint unreachable and SHALL stop transmitting any more data to it (and thus the association enters the CLOSED state). In addition, the endpoint SHOULD report the failure to the upper layer and optionally report back all outstanding user data remaining in its outbound queue. The association is automatically closed when the peer endpoint becomes unreachable.

This text has been modified by multiple errata. It includes modifications from Section 3.6. It is in final form, and is not further updated in this document.
When an outstanding TSN is acknowledged or a HEARTBEAT sent to that address is acknowledged with a HEARTBEAT ACK, the endpoint shall clear the error counter of the destination transport address to which the DATA chunk was last sent (or HEARTBEAT was sent). When the peer endpoint is multi-homed and the last chunk sent to it was a retransmission to an alternate address, there exists an ambiguity as to whether or not the acknowledgement should be credited to the address of the last chunk sent. However, this ambiguity does not seem to bear any significant consequence to SCTP behavior. If this ambiguity is undesirable, the transmitter may choose not to clear the error counter if the last chunk sent was a retransmission.

This text is in final form, and is not further updated in this document.
Old text: (Section 8.3)

When the value of this counter reaches the protocol parameter 'Path.Max.Retrans', the endpoint should mark the corresponding destination address as inactive if it is not so marked, and may also optionally report to the upper layer the change of reachability of this destination address. After this, the endpoint should continue HEARTBEAT on this destination address but should stop increasing the counter.

New text: (Section 8.3)

When the value of this counter exceeds the protocol parameter 'Path.Max.Retrans', the endpoint SHOULD mark the corresponding destination address as inactive if it is not so marked, and SHOULD also report to the upper layer the change of reachability of this destination address. After this, the endpoint SHOULD continue HEARTBEAT on this destination address but SHOULD stop increasing the counter.

This text has been modified by multiple errata. It includes modifications from Section 3.1. It is in final form, and is not further updated in this document.
Upon the receipt of the HEARTBEAT ACK, the sender of the HEARTBEAT should clear the error counter of the destination transport address to which the HEARTBEAT was sent, and mark the destination transport address as active if it is not so marked. The endpoint may optionally report to the upper layer when an inactive destination address is marked as active due to the reception of the latest HEARTBEAT ACK. The receiver of the HEARTBEAT ACK must also clear the association overall error count as well (as defined in Section 8.1).

This text has been modified by multiple errata. It includes modifications from Section 3.13. It is in final form, and is not further updated in this document.
An endpoint should limit the number of retransmissions of the SHUTDOWN chunk to the protocol parameter ‘Association.Max.Retrans’. If this threshold is exceeded, the endpoint should destroy the TCB and MUST report the peer endpoint unreachable to the upper layer (and thus the association enters the CLOSED state).

An endpoint SHOULD limit the number of retransmissions of the SHUTDOWN chunk to the protocol parameter ‘Association.Max.Retrans’. If this threshold is exceeded, the endpoint SHOULD destroy the TCB and SHOULD report the peer endpoint unreachable to the upper layer (and thus the association enters the CLOSED state).

This text is in final form, and is not further updated in this document.

The sender of the SHUTDOWN ACK should limit the number of retransmissions of the SHUTDOWN ACK chunk to the protocol parameter ‘Association.Max.Retrans’. If this threshold is exceeded, the endpoint should destroy the TCB and may report the peer endpoint unreachable to the upper layer (and thus the association enters the CLOSED state).

The sender of the SHUTDOWN ACK SHOULD limit the number of retransmissions of the SHUTDOWN ACK chunk to the protocol parameter ‘Association.Max.Retrans’. If this threshold is exceeded, the endpoint SHOULD destroy the TCB and SHOULD report the peer endpoint unreachable to the upper layer (and thus the association enters the CLOSED state).

This text is in final form, and is not further updated in this document.
3.23.3. Solution Description

The inconsistencies are removed by using consistently SHOULD.

3.24. SACK.Delay Not Listed as a Protocol Parameter

3.24.1. Description of the Problem

SCTP as specified in [RFC4960] supports delaying SACKs. The timer value for this is a parameter and Section 6.2 of [RFC4960] specifies a default and maximum value for it. However, defining a name for this parameter and listing it in the table of protocol parameters in Section 15 of [RFC4960] is missing.

This issue was reported as an Errata for [RFC4960] with Errata ID 4656.

3.24.2. Text Changes to the Document

--------
Old text: (Section 6.2)
--------

An implementation MUST NOT allow the maximum delay to be configured to be more than 500 ms. In other words, an implementation MAY lower this value below 500 ms but MUST NOT raise it above 500 ms.

--------
New text: (Section 6.2)
--------

An implementation MUST NOT allow the maximum delay (protocol parameter ‘SACK.Delay’) to be configured to be more than 500 ms. In other words, an implementation MAY lower the value of SACK.Delay below 500 ms but MUST NOT raise it above 500 ms.

This text is in final form, and is not further updated in this document.
The following protocol parameters are RECOMMENDED:

- RTO.Initial - 3 seconds
- RTO.Min - 1 second
- RTO.Max - 60 seconds
- Max.Burst - 4
- RTO.Alpha - 1/8
- RTO.Beta - 1/4
- Valid.Cookie.Life - 60 seconds
- Association.Max.Retrans - 10 attempts
- Path.Max.Retrans - 5 attempts (per destination address)
- Max.Init.Retransmits - 8 attempts
- HB.interval - 30 seconds
- HB.Max.Burst - 1

This text has been modified by multiple errata. It is further updated in Section 3.32.

3.24.3. Solution Description

The parameter was given a name and added to the list of protocol parameters.
3.25. Processing of Chunks in an Incoming SCTP Packet

3.25.1. Description of the Problem

There are a few places in [RFC4960] where the receiver of a packet must discard it while processing the chunks of the packet. It is unclear whether the receiver has to rollback state changes already performed while processing the packet or not.

The intention of [RFC4960] is to process an incoming packet chunk by chunk and not to perform any prescreening of chunks in the received packet. Thus, by discarding one chunk the receiver also causes discarding of all further chunks.

3.25.2. Text Changes to the Document

------
Old text: (Section 3.2)
------

00 - Stop processing this SCTP packet and discard it, do not process any further chunks within it.

01 - Stop processing this SCTP packet and discard it, do not process any further chunks within it, and report the unrecognized chunk in an ‘Unrecognized Chunk Type’.

------
New text: (Section 3.2)
------

00 - Stop processing this SCTP packet, discard the unrecognized chunk and all further chunks.

01 - Stop processing this SCTP packet, discard the unrecognized chunk and all further chunks, and report the unrecognized chunk in an ‘Unrecognized Chunk Type’.

This text is in final form, and is not further updated in this document.
It is helpful for some firewalls if they can inspect just the first fragment of a fragmented SCTP packet and unambiguously determine whether it corresponds to an INIT chunk (for further information, please refer to [RFC1858]). Accordingly, we stress the requirements, stated in Section 3.1, that (1) an INIT chunk MUST NOT be bundled with any other chunk in a packet, and (2) a packet containing an INIT chunk MUST have a zero Verification Tag. Furthermore, we require that the receiver of an INIT chunk MUST enforce these rules by silently discarding an arriving packet with an INIT chunk that is bundled with other chunks or has a non-zero verification tag and contains an INIT-chunk.

This text is in final form, and is not further updated in this document.

3.25.3. Solution Description

The new text makes it clear that chunks can be processed from the beginning to the end and no rollback or pre-screening is required.

3.26. CWND Increase in Congestion Avoidance Phase

3.26.1. Description of the Problem

[RFC4960] in Section 7.2.2 prescribes to increase cwnd by 1*MTU per RTT if the sender has cwnd or more bytes of data outstanding to the corresponding address in the Congestion Avoidance phase. However, this is described without normative language. Moreover, Section 7.2.2 includes an algorithm how an implementation can achieve
this but this algorithm is underspecified and actually allows increasing cwnd by more than 1*MTU per RTT.

3.26.2. Text Changes to the Document

--------
Old text: (Section 7.2.2)
--------

When cwnd is greater than ssthresh, cwnd should be incremented by 1*MTU per RTT if the sender has cwnd or more bytes of data outstanding for the corresponding transport address.

--------
New text: (Section 7.2.2)
--------

When cwnd is greater than ssthresh, cwnd SHOULD be incremented by 1*MTU per RTT if the sender has cwnd or more bytes of data outstanding for the corresponding transport address. The basic guidelines for incrementing cwnd during congestion avoidance are:

- SCTP MAY increment cwnd by 1*MTU.
- SCTP SHOULD increment cwnd by one 1*MTU once per RTT when the sender has cwnd or more bytes of data outstanding for the corresponding transport address.
- SCTP MUST NOT increment cwnd by more than 1*MTU per RTT.

This text is in final form, and is not further updated in this document.
Old text: (Section 7.2.2)

- Whenever cwnd is greater than sssthresh, upon each SACK arrival that advances the Cumulative TSN Ack Point, increase partial_bytes_acked by the total number of bytes of all new chunks acknowledged in that SACK including chunks acknowledged by the new Cumulative TSN Ack and by Gap Ack Blocks.

- When partial_bytes_acked is equal to or greater than cwnd and before the arrival of the SACK the sender had cwnd or more bytes of data outstanding (i.e., before arrival of the SACK, flightsize was greater than or equal to cwnd), increase cwnd by MTU, and reset partial_bytes_acked to (partial_bytes_acked - cwnd).

New text: (Section 7.2.2)

- Whenever cwnd is greater than sssthresh, 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.

- When partial_bytes_acked is greater than cwnd and before the arrival of the SACK the sender had less than cwnd bytes of data outstanding (i.e., before arrival of the SACK, flightsize was less than cwnd), reset partial_bytes_acked to cwnd.

- When partial_bytes_acked is equal to or greater than cwnd and before the arrival of the SACK the sender had cwnd or more bytes of data outstanding (i.e., before arrival of the SACK, flightsize was greater than or equal to cwnd), partial_bytes_acked is reset to (partial_bytes_acked - cwnd). Next, cwnd is increased by 1*MTU.

This text has been modified by multiple errata. It includes modifications from Section 3.12 and Section 3.22. It is in final form, and is not further updated in this document.

3.26.3. Solution Description

The basic guidelines for incrementing cwnd during the congestion avoidance phase are added into Section 7.2.2. The guidelines include the normative language and are aligned with [RFC5681].

The algorithm from Section 7.2.2 is improved to not allow increasing cwnd by more than 1*MTU per RTT.

3.27. Refresh of cwnd and ssthresh after Idle Period

3.27.1. Description of the Problem

[RFC4960] prescribes to adjust cwnd per RTO if the endpoint does not transmit data on a given transport address. In addition to that, it prescribes to set cwnd to the initial value after a sufficiently long idle period. The latter is excessive. Moreover, it is unclear what is a sufficiently long idle period.

[RFC4960] doesn’t specify the handling of ssthresh in the idle case. If ssthresh is reduced due to a packet loss, ssthresh is never recovered. So traffic can end up in Congestion Avoidance all the time, resulting in a low sending rate and bad performance. The problem is even more serious for SCTP because in a multi-homed SCTP association traffic that switches back to the previously failed primary path will also lead to the situation where traffic ends up in Congestion Avoidance.

3.27.2. Text Changes to the Document

Old text: (Section 7.2.1)

- The initial cwnd before DATA transmission or after a sufficiently long idle period MUST be set to min(4*MTU, max (2*MTU, 4380 bytes)).

New text: (Section 7.2.1)

- The initial cwnd before DATA transmission MUST be set to min(4*MTU, max (2*MTU, 4380 bytes)).
---
Old text: (Section 7.2.1)
---

- When 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) per RTO.

---
New text: (Section 7.2.1)
---

- 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.

This text is in final form, and is not further updated in this document.

3.27.3. Solution Description

A rule about cwnd adjustment after a sufficiently long idle period is removed.

The text is updated to describe the ssthresh handling. When the idle period is detected, the cwnd value is stored to the ssthresh value.

3.28. Window Updates After Receiver Window Opens Up

3.28.1. Description of the Problem

The sending of SACK chunks for window updates is only indirectly referenced in [RFC4960], Section 6.2, where it is stated that an SCTP receiver must not generate more than one SACK for every incoming packet, other than to update the offered window.

However, the sending of window updates when the receiver window opens up is necessary to avoid performance problems.

3.28.2. Text Changes to the Document
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.

This text is in final form, and is not further updated in this document.

3.28.3. Solution Description

The new text makes clear that additional SACK chunks for window updates should be sent as long as excessive bursts are avoided.

3.29. Path of DATA and Reply Chunks

3.29.1. Description of the Problem

Section 6.4 of [RFC4960] describes the transmission policy for multi-homed SCTP endpoints. However, there are the following issues with it:

- It states that a SACK should be sent to the source address of an incoming DATA. However, it is known that other SACK policies (e.g. sending SACKs always to the primary path) may be more beneficial in some situations.
- Initially it states that an endpoint should always transmit DATA chunks to the primary path. Then it states that the rule for transmittal of reply chunks should also be followed if the endpoint is bundling DATA chunks together with the reply chunk which contradicts with the first statement to always transmit DATA chunks.
chunks to the primary path. Some implementations were having problems with it and sent DATA chunks bundled with reply chunks to a different destination address than the primary path that caused many gaps.

3.29.2. Text Changes to the Document

-----------
Old text: (Section 6.4)
-----------

An endpoint SHOULD transmit reply chunks (e.g., SACK, HEARTBEAT ACK, etc.) to the same destination transport address from which it received the DATA or control chunk to which it is replying. This rule should also be followed if the endpoint is bundling DATA chunks together with the reply chunk.

However, when acknowledging multiple DATA chunks received in packets from different source addresses in a single SACK, the SACK chunk may be transmitted to one of the destination transport addresses from which the DATA or control chunks being acknowledged were received.

-----------
New text: (Section 6.4)
-----------

An endpoint SHOULD transmit reply chunks (e.g., INIT ACK, COOKIE ACK, HEARTBEAT ACK, etc.) in response to control chunks to the same destination transport address from which it received the control chunk to which it is replying.

The selection of the destination transport address for packets containing SACK chunks is implementation dependent. However, an endpoint SHOULD NOT vary the destination transport address of a SACK when it receives DATA chunks coming from the same source address.

When acknowledging multiple DATA chunks received in packets from different source addresses in a single SACK, the SACK chunk MAY be transmitted to one of the destination transport addresses from which the DATA or control chunks being acknowledged were received.

This text is in final form, and is not further updated in this document.
3.29.3. Solution Description

The SACK transmission policy is left implementation dependent but it is specified to not vary the destination address of a packet containing a SACK chunk unless there are reasons for it as it may negatively impact RTT measurement.

A confusing statement that prescribes to follow the rule for transmittal of reply chunks when the endpoint is bundling DATA chunks together with the reply chunk is removed.

3.30. Outstanding Data, Flightsize and Data In Flight Key Terms

3.30.1. Description of the Problem

[RFC4960] uses outstanding data, flightsize and data in flight key terms in formulas and statements but their definitions are not provided in Section 1.3. Furthermore, outstanding data does not include DATA chunks which are classified as lost but which have not been retransmitted yet and there is a paragraph in Section 6.1 of [RFC4960] where this statement is broken.

3.30.2. Text Changes to the Document
Old text: (Section 1.3)

- Congestion window (cwnd): An SCTP variable that limits the data, in number of bytes, a sender can send to a particular destination transport address before receiving an acknowledgement.

- Outstanding TSN (at an SCTP endpoint): A TSN (and the associated DATA chunk) that has been sent by the endpoint but for which it has not yet received an acknowledgement.

New text: (Section 1.3)

- 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.

- Outstanding data (or Data outstanding or Data in flight): The total amount of the DATA chunks associated with outstanding TSNs. A retransmitted DATA chunk is counted once in outstanding data. A DATA chunk which is classified as lost but which has not been retransmitted yet is not in outstanding data.

- Flightsize: The amount of bytes of outstanding data to a particular destination transport address at any given time.

- Congestion window (cwnd): An SCTP variable that limits outstanding data, in number of bytes, a sender can send to a particular destination transport address before receiving an acknowledgement.

This text is in final form, and is not further updated in this document.
C) When the time comes for the sender to transmit, before sending new DATA chunks, the sender MUST first transmit any outstanding DATA chunks that are marked for retransmission (limited by the current cwnd).

This text is in final form, and is not further updated in this document.

3.30.3. Solution Description

Now Section 1.3, Key Terms, includes explanations of outstanding data, data in flight and flightsize key terms. Section 6.1 is corrected to properly use the outstanding data term.

3.31. CWND Degradation due to Max.Burst

3.31.1. Description of the Problem

Some implementations were experiencing a degradation of cwnd because of the Max.Burst limit. This was due to misinterpretation of the suggestion in [RFC4960], Section 6.1, on how to use the Max.Burst parameter when calculating the number of packets to transmit.

3.31.2. Text Changes to the Document
D) When the time comes for the sender to transmit new DATA chunks, the protocol parameter Max.Burst SHOULD be used to limit the number of packets sent. The limit MAY be applied by adjusting cwnd as follows:

\[
\text{if}\left((\text{flightsize} + \text{Max.Burst} \times \text{MTU}) < \text{cwnd}\right) \text{cwnd} = \text{flightsize} + \text{Max.Burst} \times \text{MTU}
\]

Or it MAY be applied by strictly limiting the number of packets emitted by the output routine.

This text is in final form, and is not further updated in this document.

3.31.3. Solution Description

The new text clarifies that cwnd should not be changed when applying the Max.Burst limit. This mitigates packet bursts related to the reception of SACK chunks, but not bursts related to an application sending a burst of user messages.

3.32. Reduction of RTO.Initial
3.32.1. Description of the Problem

[RFC4960] uses 3 seconds as the default value for RTO.Initial in accordance with Section 4.3.2.1 of [RFC1122]. [RFC6298] updates [RFC1122] and lowers the initial value of the retransmission timer from 3 seconds to 1 second.

3.32.2. Text Changes to the Document

--------
Old text: (Section 15)
--------

The following protocol parameters are RECOMMENDED:

- RTO.Initial - 3 seconds
- RTO.Min - 1 second
- RTO.Max - 60 seconds
- Max.Burst - 4
- RTO.Alpha - 1/8
- RTO.Beta - 1/4
- Valid.Cookie.Life - 60 seconds
- Association.Max.Retrans - 10 attempts
- Path.Max.Retrans - 5 attempts (per destination address)
- Max.Init.Retransmits - 8 attempts
- HB.interval - 30 seconds
- HB.Max.Burst - 1

--------
New text: (Section 15)
--------

The following protocol parameters are RECOMMENDED:

- RTO.Initial - 1 second
- RTO.Min - 1 second
- RTO.Max - 60 seconds
- Max.Burst - 4
- RTO.Alpha - 1/8
- RTO.Beta - 1/4
- Valid.Cookie.Life - 60 seconds
- Association.Max.Retrans - 10 attempts
- Path.Max.Retrans - 5 attempts (per destination address)
- Max.Init.Retransmits - 8 attempts
- HB.interval - 30 seconds
- HB.Max.Burst - 1
- SACK.Delay - 200 milliseconds
This text has been modified by multiple errata. It includes modifications from Section 3.24. It is in final form, and is not further updated in this document.

3.32.3. Solution Description

The value RTO.Initial has been lowered to 1 second to be in tune with [RFC6298].

3.33. Ordering of Bundled SACK and ERROR Chunks

3.33.1. Description of the Problem

When an SCTP endpoint receives a DATA chunk with an invalid stream identifier it shall acknowledge it by sending a SACK chunk and indicate that the stream identifier was invalid by sending an ERROR chunk. These two chunks may be bundled. However, [RFC4960] requires in case of bundling that the ERROR chunk follows the SACK chunk. This restriction of the ordering is not necessary and might only limit interoperability.

3.33.2. Text Changes to the Document

---------
Old text: (Section 6.5)
---------
Every DATA chunk MUST carry a valid stream identifier. If an endpoint receives a DATA chunk with an invalid stream identifier, it shall acknowledge the reception of the DATA chunk following the normal procedure, immediately send an ERROR chunk with cause set to "Invalid Stream Identifier" (see Section 3.3.10), and discard the DATA chunk. The endpoint may bundle the ERROR chunk in the same packet as the SACK as long as the ERROR follows the SACK.

---------
New text: (Section 6.5)
---------
Every DATA chunk MUST carry a valid stream identifier. If an endpoint receives a DATA chunk with an invalid stream identifier, it SHOULD acknowledge the reception of the DATA chunk following the normal procedure, immediately send an ERROR chunk with cause set to "Invalid Stream Identifier" (see Section 3.3.10), and discard the DATA chunk. The endpoint MAY bundle the ERROR chunk and the SACK Chunk in the same packet.
This text is in final form, and is not further updated in this document.

3.33.3. Solution Description

The unnecessary restriction regarding the ordering of the SACK and ERROR chunk has been removed.

3.34. Undefined Parameter Returned by RECEIVE Primitive

3.34.1. Description of the Problem

[RFC4960] provides a description of an abstract API. In the definition of the RECEIVE primitive an optional parameter with name "delivery number" is mentioned. However, no definition of this parameter is given in [RFC4960] and the parameter is unnecessary.

3.34.2. Text Changes to the Document

---------
Old text: (Section 10.1 G))
---------

G) Receive
Format: RECEIVE(association id, buffer address, buffer size [,stream id])

---------
New text: (Section 10.1 G))
---------

G) Receive
Format: RECEIVE(association id, buffer address, buffer size [,stream id])

This text is in final form, and is not further updated in this document.
3.34.3. Solution Description

The undefined parameter has been removed.

3.35. DSCP Changes

3.35.1. Description of the Problem

The upper layer can change the Differentiated Services Code Point (DSCP) used for packets being sent. A change of the DSCP can result in packets hitting different queues on the path and, therefore, the congestion control should be initialized when the DSCP is changed by the upper layer. This is not described in [RFC4960].

3.35.2. Text Changes to the Document

New text: (Section 7.2.5)

7.2.5. Change of Differentiated Services Code Points

SCTP implementations MAY allow an application to configure the Differentiated Services Code Point (DSCP) used for sending packets. If a DSCP change might result in outgoing packets being queued in different queues, the congestion control parameters for all affected destination addresses MUST be reset to their initial values.

This text is in final form, and is not further updated in this document.
Old text: (Section 10.1 M))

Mandatory attributes:

- 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) that the SCTP user wishes to customize.

New text: (Section 10.1 M))

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.

This text is in final form, and is not further updated in this document.

3.35.3. Solution Description

Text describing the required action on DSCP changes has been added.

3.36. Inconsistent Handling of ICMPv4 and ICMPv6 Messages

3.36.1. Description of the Problem

Appendix C of [RFC4960] describes the handling of ICMPv4 and ICMPv6 messages. The handling of ICMP messages indicating that the port number is unreachable described in the enumeration is not consistent with the description given in [RFC4960] after the enumeration. Furthermore, the text explicitly describes the handling of ICMPv6 packets indicating reachability problems, but does not do the same for the corresponding ICMPv4 packets.
3.36.2. Text Changes to the Document

_________
Old text: (Appendix C)
_________

ICMP3) An implementation MAY ignore any ICMPv4 messages where the code does not indicate "Protocol Unreachable" or "Fragmentation Needed".

_________
New text: (Appendix C)
_________

ICMP3) An implementation SHOULD ignore any ICMP messages where the code indicates "Port Unreachable".

This text is in final form, and is not further updated in this document.

_________
Old text: (Appendix C)
_________

ICMP9) If the ICMPv6 code is "Destination Unreachable", the implementation MAY mark the destination into the unreachable state or alternatively increment the path error counter.

_________
New text: (Appendix C)
_________

ICMP9) If the ICMP type is "Destination Unreachable", the implementation MAY mark the destination into the unreachable state or alternatively increment the path error counter.

This text has been modified by multiple errata. It is further updated in Section 3.37.

3.36.3. Solution Description

The text has been changed to describe the intended handling of ICMP messages indicating that the port number is unreachable by replacing the third rule. Furthermore, remove the limitation to ICMPv6 in the ninth rule.
3.37. Handling of Soft Errors

3.37.1. Description of the Problem

[RFC1122] defines the handling of soft errors and hard errors for TCP. Appendix C of [RFC4960] only deals with hard errors.

3.37.2. Text Changes to the Document

---------
Old text: (Appendix C)
---------

ICMP9) If the ICMPv6 code is "Destination Unreachable", the implementation MAY mark the destination into the unreachable state or alternatively increment the path error counter.

---------
New text: (Appendix C)
---------

ICMP9) If the ICMP type is "Destination Unreachable", the implementation MAY mark the destination into the unreachable state or alternatively increment the path error counter. SCTP MAY provide information to the upper layer indicating the reception of ICMP messages when reporting a network status change.

This text has been modified by multiple errata. It includes modifications from Section 3.36. It is in final form, and is not further updated in this document.

3.37.3. Solution Description

Text has been added allowing SCTP to notify the application in case of soft errors.

3.38. Honoring CWND

3.38.1. Description of the Problem

When using the slow start algorithm, SCTP increases the congestion window only when it is being fully utilized. Since SCTP uses DATA chunks and does not use the congestion window to fragment user messages, this requires that some overbooking of the congestion window is allowed.
3.38.2. Text Changes to the Document

---------
Old text: (Section 6.1)
---------

B) At any given time, the sender MUST NOT transmit new data to a given transport address if it has cwnd or more bytes of data outstanding to that transport address.

---------
New text: (Section 6.1)
---------

B) At any given time, the sender MUST NOT transmit new data to a given transport address if it has cwnd + (PMTU - 1) or more bytes of data outstanding to that transport address. If data is available the sender SHOULD exceed cwnd by up to (PMTU-1) bytes on a new data transmission if the flightsize does not currently reach cwnd. The breach of cwnd MUST constitute one packet only.

This text is in final form, and is not further updated in this document.

---------
Old text: (Section 7.2.1)
---------

o Whenever cwnd is greater than zero, the endpoint is allowed to have cwnd bytes of data outstanding on that transport address.

---------
New text: (Section 7.2.1)
---------

o Whenever cwnd is greater than zero, the endpoint is allowed to have cwnd bytes of data outstanding on that transport address. A limited overbooking as described in B) of Section 6.1 SHOULD be supported.

This text is in final form, and is not further updated in this document.

3.38.3. Solution Description

Text was added to clarify how the cwnd limit should be handled.
3.39.  Zero Window Probing

3.39.1.  Description of the Problem

The text describing zero window probing was not clearly handling the case where the window was not zero, but too small for the next DATA chunk to be transmitted. Even in this case, zero window probing has to be performed to avoid deadlocks.

3.39.2.  Text Changes to the Document
A) At any given time, the data sender MUST NOT transmit new data to any destination transport address if its peer’s rwnd indicates that the peer has no buffer space (i.e., rwnd is 0; see Section 6.2.1). However, regardless of the value of rwnd (including if it is 0), the data sender can always have one DATA chunk in flight to the receiver if allowed by cwnd (see rule B, below). This rule allows the sender to probe for a change in rwnd that the sender missed due to the SACK’s having been lost in transit from the data receiver to the data sender.

When the receiver’s advertised window is zero, this probe is called a zero window probe. Note that a zero window probe SHOULD only be sent when all outstanding DATA chunks have been cumulatively acknowledged and no DATA chunks are in flight. Zero window probing MUST be supported.

This text is in final form, and is not further updated in this document.
3.39.3. Solution Description

The terminology is used in a cleaner way.

3.40. Updating References Regarding ECN

3.40.1. Description of the Problem

[RFC4960] refers for ECN only to [RFC3168], which will be updated by [RFC8311]. This needs to be reflected when referring to ECN.

3.40.2. Text Changes to the Document

---------
Old text: (Appendix A)
---------

ECN [RFC3168] describes a proposed extension to IP that details a method to become aware of congestion outside of datagram loss.

---------
New text: (Appendix A)
---------

ECN as specified in [RFC3168] updated by [RFC8311] describes an extension to IP that details a method to become aware of congestion outside of datagram loss.

This text is in final form, and is not further updated in this document.

---------
Old text: (Appendix A)
---------

In general, [RFC3168] should be followed with the following exceptions.

---------
New text: (Appendix A)
---------

In general, [RFC3168] updated by [RFC8311] SHOULD be followed with the following exceptions.

This text is in final form, and is not further updated in this document.
[RFC3168] details negotiation of ECN during the SYN and SYN-ACK stages of a TCP connection.


This text is in final form, and is not further updated in this document.

[RFC3168] details a specific bit for a receiver to send back in its TCP acknowledgements to notify the sender of the Congestion Experienced (CE) bit having arrived from the network.

[RFC3168] updated by [RFC8311] details a specific bit for a receiver to send back in its TCP acknowledgements to notify the sender of the Congestion Experienced (CE) bit having arrived from the network.

This text is in final form, and is not further updated in this document.
Internet-Draft         RFC 4960 Errata and Issues           October 2018

--------
Old text: (Appendix A)
--------

[RFC3168] details a specific bit for a sender to send in the header of its next outbound TCP segment to indicate to its peer that it has reduced its congestion window.

--------
New text: (Appendix A)
--------

[RFC3168] updated by [RFC8311] details a specific bit for a sender to send in the header of its next outbound TCP segment to indicate to its peer that it has reduced its congestion window.

This text is in final form, and is not further updated in this document.

3.40.3. Solution Description

References to [RFC8311] have been added. While there, some wordsmithing has been performed.

3.41. Host Name Address Parameter Deprecated

3.41.1. Description of the Problem

[RFC4960] defines three types of address parameters to be used with INIT and INIT ACK chunks:

1. IPv4 Address parameters.
2. IPv6 Address parameters.
3. Host Name Address parameters.

The first two are supported by the SCTP kernel implementations of FreeBSD, Linux and Solaris, but the third one is not. In addition, the first two were successfully tested in all nine interoperability tests for SCTP, but the third one has never been successfully tested. Therefore, the Host Name Address parameter should be deprecated.

3.41.2. Text Changes to the Document
Note 3: An INIT chunk MUST NOT contain more than one Host Name Address parameter. Moreover, the sender of the INIT MUST NOT combine any other address types with the Host Name Address in the INIT. The receiver of INIT MUST ignore any other address types if the Host Name Address parameter is present in the received INIT chunk.

Note 3: An INIT chunk MUST NOT contain the Host Name Address parameter. The receiver of an INIT chunk containing a Host Name Address parameter MUST send an ABORT and MAY include an Error Cause indicating an Unresolvable Address.

This text is in final form, and is not further updated in this document.

The sender of INIT uses this parameter to pass its Host Name (in place of its IP addresses) to its peer. The peer is responsible for resolving the name. Using this parameter might make it more likely for the association to work across a NAT box.

The sender of an INIT chunk MUST NOT include this parameter. The usage of the Host Name Address parameter is deprecated.

This text is in final form, and is not further updated in this document.
Old text: (Section 3.3.2.1)

Address Type: 16 bits (unsigned integer)

This is filled with the type value of the corresponding address TLV (e.g., IPv4 = 5, IPv6 = 6, Host name = 11).

New text: (Section 3.3.2.1)

Address Type: 16 bits (unsigned integer)

This is filled with the type value of the corresponding address TLV (e.g., IPv4 = 5, IPv6 = 6). The value indicating the Host Name Address parameter (Host name = 11) MUST NOT be used.

This text is in final form, and is not further updated in this document.

Old text: (Section 3.3.3)

Note 3: The INIT ACK chunks MUST NOT contain more than one Host Name Address parameter. Moreover, the sender of the INIT ACK MUST NOT combine any other address types with the Host Name Address in the INIT ACK. The receiver of the INIT ACK MUST ignore any other address types if the Host Name Address parameter is present.

New text: (Section 3.3.3)

Note 3: An INIT ACK chunk MUST NOT contain the Host Name Address parameter. The receiver of INIT ACK chunks containing a Host Name Address parameter MUST send an ABORT and MAY include an Error Cause indicating an Unresolvable Address.

This text is in final form, and is not further updated in this document.

Old text: (Section 5.1.2)

B) If there is a Host Name parameter present in the received INIT or
INIT ACK chunk, the endpoint shall resolve that host name to a list of IP address(es) and derive the transport address(es) of this peer by combining the resolved IP address(es) with the SCTP source port.

The endpoint MUST ignore any other IP Address parameters if they are also present in the received INIT or INIT ACK chunk.

The time at which the receiver of an INIT resolves the host name has potential security implications to SCTP. If the receiver of an INIT resolves the host name upon the reception of the chunk, and the mechanism the receiver uses to resolve the host name involves potential long delay (e.g., DNS query), the receiver may open itself up to resource attacks for the period of time while it is waiting for the name resolution results before it can build the State Cookie and release local resources.

Therefore, in cases where the name translation involves potential long delay, the receiver of the INIT MUST postpone the name resolution till the reception of the COOKIE ECHO chunk from the peer. In such a case, the receiver of the INIT SHOULD build the State Cookie using the received Host Name (instead of destination transport addresses) and send the INIT ACK to the source IP address from which the INIT was received.

The receiver of an INIT ACK shall always immediately attempt to resolve the name upon the reception of the chunk.

The receiver of the INIT or INIT ACK MUST NOT send user data (piggy-backed or stand-alone) to its peer until the host name is successfully resolved.

If the name resolution is not successful, the endpoint MUST immediately send an ABORT with "Unresolvable Address" error cause to its peer. The ABORT shall be sent to the source IP address from which the last peer packet was received.

---------
New text: (Section 5.1.2)
---------

B) If there is a Host Name parameter present in the received INIT or INIT ACK chunk, the endpoint MUST immediately send an ABORT and MAY include an Error Cause indicating an Unresolvable Address to its peer. The ABORT SHALL be sent to the source IP address from which the last peer packet was received.
The use of the host name feature in the INIT chunk could be used to flood a target DNS server. A large backlog of DNS queries, resolving the host name received in the INIT chunk to IP addresses, could be accomplished by sending INITs to multiple hosts in a given domain. In addition, an attacker could use the host name feature in an indirect attack on a third party by sending large numbers of INITs to random hosts containing the host name of the target. In addition to the strain on DNS resources, this could also result in large numbers of INIT ACKs being sent to the target. One method to protect against this type of attack is to verify that the IP addresses received from DNS include the source IP address of the original INIT. If the list of IP addresses received from DNS does not include the source IP address of the INIT, the endpoint MAY silently discard the INIT. This last option will not protect against the attack against the DNS.

The support of the Host Name Address parameter has been removed from the protocol. Endpoints receiving INIT or INIT ACK chunks containing the Host Name Address parameter MUST send an ABORT chunk in response and MAY include an Error Cause indicating an Unresolvable Address.

The usage of the Host Name Address parameter has been deprecated.

When receiving an SCTP packet containing an INIT chunk sent from an address for which the corresponding address type is not listed in the Supported Address Types, there is conflicting text in Section 5.1.2 of [RFC4960]. It is stated that the association MUST be aborted and also that the association SHOULD be established and there SHOULD NOT be any error indication.
3.42.2. Text Changes to the Document

-------
Old text: (Section 5.1.2)
-------

The sender of INIT may include a 'Supported Address Types' parameter in the INIT to indicate what types of addresses are acceptable. When this parameter is present, the receiver of INIT (initiate) MUST either use one of the address types indicated in the Supported Address Types parameter when responding to the INIT, or abort the association with an "Unresolvable Address" error cause if it is unwilling or incapable of using any of the address types indicated by its peer.

-------
New text: (Section 5.1.2)
-------

The sender of INIT chunks MAY include a 'Supported Address Types' parameter in the INIT to indicate what types of addresses are acceptable.

This text is in final form, and is not further updated in this document.

3.42.3. Solution Description

The conflicting text has been removed.

3.43. Integration of RFC 6096

3.43.1. Description of the Problem

[RFC6096] updates [RFC4960] by adding a Chunk Flags Registry. This should be integrated into the base specification.

3.43.2. Text Changes to the Document

-------
Old text: (Section 14.1)
-------

14.1. IETF-Defined Chunk Extension

The assignment of new chunk parameter type codes is done through an IETF Consensus action, as defined in [RFC2434]. Documentation of the chunk parameter MUST contain the following information:
a) A long and short name for the new chunk type.

b) A detailed description of the structure of the chunk, which MUST conform to the basic structure defined in Section 3.2.

c) A detailed definition and description of the intended use of each field within the chunk, including the chunk flags if any.

d) A detailed procedural description of the use of the new chunk type within the operation of the protocol.

The last chunk type (255) is reserved for future extension if necessary.

New text: (Section 14.1)

14.1. IETF-Defined Chunk Extension

The assignment of new chunk type codes is done through an IETF Review action, as defined in [RFC8126]. Documentation of a new chunk MUST contain the following information:

a) A long and short name for the new chunk type;

b) A detailed description of the structure of the chunk, which MUST conform to the basic structure defined in Section 3.2 of [RFC4960];

c) A detailed definition and description of the intended use of each field within the chunk, including the chunk flags if any. Defined chunk flags will be used as initial entries in the chunk flags table for the new chunk type;

d) A detailed procedural description of the use of the new chunk type within the operation of the protocol.

The last chunk type (255) is reserved for future extension if necessary.

For each new chunk type, IANA creates a registration table for the chunk flags of that type. The procedure for registering particular chunk flags is described in the following Section 14.2.

This text has been modified by multiple errata. It includes modifications from Section 3.3. It is in final form, and is not further updated in this document.
New text: (Section 14.2)

14.2. New IETF Chunk Flags Registration

The assignment of new chunk flags is done through an RFC required action, as defined in [RFC8126]. Documentation of the chunk flags MUST contain the following information:

a) A name for the new chunk flag;

b) A detailed procedural description of the use of the new chunk flag within the operation of the protocol. It MUST be considered that implementations not supporting the flag will send ‘0’ on transmit and just ignore it on receipt.

IANA selects a chunk flags value. This MUST be one of 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, or 0x80, which MUST be unique within the chunk flag values for the specific chunk type.

This text is in final form, and is not further updated in this document.

Please note that Sections 14.2, 14.3, 14.4, and 14.5 need to be renumbered.

3.43.3. Solution Description

[RFC6096] was integrated and the reference updated to [RFC8126].

3.44. Integration of RFC 6335

3.44.1. Description of the Problem

[RFC6335] updates [RFC4960] by updating Procedures for the Port Numbers Registry. This should be integrated into the base specification. While there, update the reference to the RFC giving guidelines for writing IANA sections to [RFC8126].

3.44.2. Text Changes to the Document

Old text: (Section 14.5)

SCFP services may use contact port numbers to provide service to unknown callers, as in TCP and UDP. IANA is therefore requested to
open the existing Port Numbers registry for SCTP using the following rules, which we intend to mesh well with existing Port Numbers registration procedures. An IESG-appointed Expert Reviewer supports IANA in evaluating SCTP port allocation requests, according to the procedure defined in [RFC2434].

Port numbers are divided into three ranges. The Well Known Ports are those from 0 through 1023, the Registered Ports are those from 1024 through 49151, and the Dynamic and/or Private Ports are those from 49152 through 65535. Well Known and Registered Ports are intended for use by server applications that desire a default contact point on a system. On most systems, Well Known Ports can only be used by system (or root) processes or by programs executed by privileged users, while Registered Ports can be used by ordinary user processes or programs executed by ordinary users. Dynamic and/or Private Ports are intended for temporary use, including client-side ports, out-of-band negotiated ports, and application testing prior to registration of a dedicated port; they MUST NOT be registered.

The Port Numbers registry should accept registrations for SCTP ports in the Well Known Ports and Registered Ports ranges. Well Known and Registered Ports SHOULD NOT be used without registration. Although in some cases -- such as porting an application from TCP to SCTP -- it may seem natural to use an SCTP port before registration completes, we emphasize that IANA will not guarantee registration of particular Well Known and Registered Ports. Registrations should be requested as early as possible.

Each port registration SHALL include the following information:

- A short port name, consisting entirely of letters (A-Z and a-z), digits (0-9), and punctuation characters from "-_+/\*" (not including the quotes).
- The port number that is requested for registration.
- A short English phrase describing the port’s purpose.
- Name and contact information for the person or entity performing the registration, and possibly a reference to a document defining the port’s use. Registrations coming from IETF working groups need only name the working group, but indicating a contact person is recommended.

Registrants are encouraged to follow these guidelines when submitting a registration.

- A port name SHOULD NOT be registered for more than one SCTP port
number.

- A port name registered for TCP MAY be registered for SCTP as well. Any such registration SHOULD use the same port number as the existing TCP registration.

- Concrete intent to use a port SHOULD precede port registration. For example, existing TCP ports SHOULD NOT be registered in advance of any intent to use those ports for SCTP.

-------------
New text: (Section 14.5)
-------------

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 Port Numbers registry for SCTP using the following rules, which we intend to mesh well with existing Port Numbers registration procedures. An IESG-appointed Expert Reviewer supports IANA in evaluating SCTP port allocation requests, according to the procedure defined in [RFC8126]. The details of this process are defined in [RFC6335].

This text is in final form, and is not further updated in this document.

3.44.3. Solution Description

[RFC6335] was integrated and the reference was updated to [RFC8126].

3.45. Integration of RFC 7053

3.45.1. Description of the Problem

[RFC7053] updates [RFC4960] by adding the I bit to the DATA chunk. This should be integrated into the base specification.

3.45.2. Text Changes to the Document

-------------
Old text: (Section 3.3.1)
-------------
-------------

The following format MUST be used for the DATA chunk:

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Stewart, et al. Expires April 25, 2019
Reserved: 5 bits

Should be set to all ‘0’s and ignored by the receiver.

--------
New text: (Section 3.3.1)
--------

Res: 4 bits

SHOULD be set to all ‘0’s and ignored by the receiver.

I bit: 1 bit

The (I)mmediate Bit MAY be set by the sender, whenever the sender of a DATA chunk can benefit from the corresponding SACK chunk being sent back without delay. See [RFC7053] for a discussion about
Whenever the sender of a DATA chunk can benefit from the corresponding SACK chunk being sent back without delay, the sender MAY set the I bit in the DATA chunk header. Please note that why the sender has set the I bit is irrelevant to the receiver.

Reasons for setting the I bit include, but are not limited to (see Section 4 of [RFC7053] for the benefits):

- The application requests to set the I bit of the last DATA chunk of a user message when providing the user message to the SCTP implementation (see Section 7).
- The sender is in the SHUTDOWN-PENDING state.
- The sending of a DATA chunk fills the congestion or receiver window.

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.
Please note that this change is only about adding a paragraph.
This text is in final form, and is not further updated in this document.

--------
Old text: (Section 10.1 E))
--------

E) Send
-> result

--------
New text: (Section 10.1 E))
--------

E) Send
-> result

This text is in final form, and is not further updated in this document.

--------
New text: (Append optional parameter in Subsection E of Section 10.1)
--------

o sack immediately - set the I bit on the last DATA chunk used for sending buffer.

This text is in final form, and is not further updated in this document.

3.45.3. Solution Description

[RFC7053] was integrated.
3.46. CRC32c Code Improvements

3.46.1. Description of the Problem

The code given for the CRC32c computations uses types like long which may have different length on different operating systems or processors. Therefore, the code is changed to use specific types like uint32_t.

While there, fix also some syntax errors and a comment.

3.46.2. Text Changes to the Document

--------
Old text: (Appendix C)
--------
/***********************************************************/
/* Note Definition for Ross Williams table generator would */
/* be: TB_WIDTH=4, TB_POLLY=0x1EDC6F41, TB_REVER=TRUE    */
/* For Mr. Williams direct calculation code use the settings */
/* cm_width=32, cm_poly=0x1EDC6F41, cm_init=0xFFFFFFFF,   */
/* cm_refin=TRUE, cm_refot=TRUE, cm_xorort=0x00000000       */
/***********************************************************/
/* Example of the crc table file */
 ifndef __crc32cr_table_h__
 #define __crc32cr_table_h__
 #define CRC32C_POLY 0x1EDC6F41
 #define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])

 unsigned long  crc_c[256] =
 {
  0x00000000L, 0xF26B8303L, 0xE13B70F7L, 0x1350F3F4L,
  0xC79A971FL, 0x35F1141CL, 0x26A1E7E8L, 0xD4CA64EBL,
  0x8AD958CFL, 0x78B2DBCCL, 0x6BE22838L, 0x9989AB3BL,
  0x4D43CFD0L, 0xBF284CD3L, 0xAC78BF27L, 0x5E133C24L,
  0x105EC76FL, 0xE235446CL, 0xF165B798L, 0x030E349BL,
  0xD7C45070L, 0x25AFD373L, 0x36FF2087L, 0xC494A384L,
  0x9A879FA0L, 0x668E1CA3L, 0x7BBCE57L, 0x89D76C54L,
  0x5D1D08BFL, 0xAF768BCCL, 0xBC267848L, 0x4E4DFB4BL,
  0x20BD8EDEL, 0xD2D60DDDL, 0xC186FE29L, 0x33ED7D2AL,
  0xE72719C1L, 0x154C9AC2L, 0x061C6936L, 0xF477EA35L,
  0xAA64D611L, 0x580F5512L, 0x4B5FA666L, 0xB93425E5L,
  0x6DPE410EL, 0x9F95C20DL, 0x8CC531F9L, 0x7EAEB2FAL,
  0x30E349B1L, 0xC288CAB2L, 0xD1D83946L, 0x23B3BA45L,
#ifndef __crc32cr_h__
#define __crc32cr_h__
#define CRC32C_POLY 0x1EDC6F41UL
#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])

uint32_t crc_c[256] = {
    0x00000000UL, 0xF26B8303UL, 0xE13B70F7UL, 0x1350F3F4UL,
    0xC79A971FUL, 0x35F1141CUL, 0x26A1E7E8UL, 0xD4CA64EBUL,
    0x8AD958CFUL, 0x78B2DBCCUL, 0x6BE22838UL, 0x9989AB3BUL,
    0x4D43CFD0UL, 0xBF284CD3UL, 0xAC78BF27UL, 0x5E133C24UL,
    0x105EC76FUL, 0xE235446CUL, 0xF165B798UL, 0x030E349BUL,
    0xD7C45070UL, 0x25AFD373UL, 0x36FF2087UL, 0xC494A384UL,
    0x09A879FA0UL, 0x68EC1CA3UL, 0x7BBCEF57UL, 0x89D76C54UL,
    0x5D1D8EDEUL, 0xAF766BCCUL, 0xBC267848UL, 0x4E4DFB4BUL,
    0x20B8DEDEUL, 0x0D26D0DDUL, 0xC186F29UL, 0x33ED7D2AUL,
    0x827219C1UL, 0x154C9AC2UL, 0x061C6936UL, 0xF477EA35UL,
    0xAA64D611UL, 0x580F5512UL, 0x4B5FA6E6UL, 0xB93425E5UL,
    0x6DFF41E0UL, 0x9F95C20DUL, 0x8CC531F9UL, 0x7EAE2F45UL,
    0x30E349B1UL, 0x2C88CAB2UL, 0xD1D83946UL, 0x23B3BA45UL,
    0xF779DEAEUL, 0x05125DADUL, 0x1642AE59UL, 0xE4292D5AUL,
    0x8B3A117EUL, 0x4801927DUL, 0x5B016189UL, 0xA96EA284UL,
    0x7DA86661UL, 0x8F8CB0562UL, 0x9C9BF696UL, 0xEF07595UL,
    0x417BD8CUL, 0xB3109EBFUL, 0xA0406D48UL, 0x522BE48UL,
    0x86E18AA3UL, 0x748A09A0UL, 0x67DAPA54UL, 0x95B17957UL,
    0xCBA24573UL, 0x39C9C670UL, 0x2A993584UL, 0xD8F2B687UL,
};
0x0C38D26CUL, 0xFE53516FUL, 0xED03A29BUL, 0x1F682198UL,
0x5125DAD3UL, 0xA34E59D0UL, 0xB01EAA24UL, 0x42752927UL,
0x96BF4DCCUL, 0x64D4CECFUL, 0x77843D3BUL, 0x85EFBE38UL,
0xDBFC821CUL, 0x2997011FUL, 0x3AC7F2EBUL, 0xC8AC71E8UL,
0x1C661503UL, 0xEE0D9600UL, 0xFD5D65F4UL, 0x0F36E6F7UL,
0x61C69362UL, 0x93AD1061UL, 0x80FDE395UL, 0x72966096UL,
0x197448AEUL, 0x0A24BB5AUL, 0xF84F3859UL,
0x2C855CB2UL, 0xCEEDFB1UL, 0xCDE2C45UL, 0x3FD5AF46UL,
0x719540DUL, 0x83F3D70EUL, 0x90A324FAUL, 0x62C8A7F9UL,
0x46D122B9UL, 0x97BA1A4EUL, 0x84EA524EUL, 0x7681D14DUL,
0xCFBB037UL, 0x0F94AD42UL, 0x0B21572CU,
0x24BB5A6UL, 0x3EDD4300UL, 0xCBCBC033UL,
0xF56BD19UL, 0x0DD3D31AUL, 0x1E6DCDEEU, 0xECOD64EDUL,
0xC38D26C4UL, 0x30E6A5C7UL, 0x22B65633UL, 0x0DDDD350UL,
0x4187B1DUL, 0xF67C32D8UL, 0xE52CC12CUL, 0x1747422FUL,
0x4957E08UL, 0xBB3F08EUL, 0xA86F0EFUL, 0x5A048DFFUL,
0x8ECEE914UL, 0x7CA56A17UL, 0x6FF599E3UL, 0x9D9E1AE0UL,
0x3D3821AUL, 0x21B8E915UL, 0xC083125FUL,
0x144976B8UL, 0xE6E2F5B7UL, 0xF5720634UL, 0x719B540UL,
0x590AB964UL, 0x40B1A67UL, 0xB931C993UL, 0x41A4A90UL,
0x9E902E7BUL, 0x6CBAD78UL, 0x7FAB58CEUL, 0x8D0DDDFUL,
0x330A81AUL, 0x115BB291UL, 0x020BD8EDUL, 0xF060BEEEUL,
0x24AA3F05UL, 0x6D1BC06UL, 0xC5914FF2UL, 0x37ACFCE1UL,
0x69E9F08UL, 0x9B273D6UL, 0x8BD2022UL, 0x7AB9032UL,
0x73673AUL, 0x5C18E49UL, 0x40F8173DUL, 0xBD23943EUL,
0xF36EF75UL, 0x105EC76UL, 0x12551F82UL, 0xE03EC81UL,
0x3F4F86AUL, 0xC069F7E9UL, 0x65CF889DUL, 0x2A40B9EUL,
0x79B737BAUL, 0x8BC9B49UL, 0x980C474DUL, 0x6A57C44EUL,
0xBE2DA0A5UL, 0x4C4623A6UL, 0x5F16D052UL, 0xAD7D5351UL,

};

#endif
This text has been modified by multiple errata. It includes modifications from Section 3.10. It is in final form, and is not further updated in this document.

---------
Old text: (Appendix C)
---------

/* Example of table build routine */

#include <stdio.h>
#include <stdlib.h>

#define OUTPUT_FILE   "crc32cr.h"
#define CRC32C_POLY   0x1EDC6F41L
FILE *tf;
unsigned long
reflect_32 (unsigned long b)
{
  int i;
  unsigned long rw = 0L;

  for (i = 0; i < 32; i++){
    if (b & 1)
      rw |= 1L << (31 - i);
    b >>= 1;
  }
  return (rw);
}

unsigned long
build_crc_table (int index)
{
  int i;
  unsigned long rb;

  rb = reflect_32 (index);

  for (i = 0; i < 8; i++){
    if (rb & 0x80000000L)
      rb = (rb << 1) ^ CRC32C_POLY;
    else
      rb <<= 1;
  }
  return (reflect_32 (rb));
}
main ()
{
    int i;

    printf ("\nGenerating CRC-32c table file <%s>\n", OUTPUT_FILE);
    if ((tf = fopen (OUTPUT_FILE, "w")) == NULL){
        printf ("Unable to open %s\n", OUTPUT_FILE);
        exit (1);
    }
    fprintf (tf, "#ifndef __crc32cr_table_h__\n");
    fprintf (tf, "#define __crc32cr_table_h__\n\n");
    fprintf (tf, "#define CRC32C_POLY 0x%08lX\n", CRC32C_POLY);
    fprintf (tf, "#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF]\n");
    fprintf (tf, "\nunsigned long  crc_c[256] =\n{
    for (i = 0; i < 256; i++){
        fprintf (tf, "0x%08lXL, \n", build_crc_table (i));
        if ((i & 3) == 3)
            fprintf (tf, "\n");
    }
    fprintf (tf, "};\n#endif\n");

    if (fclose (tf) != 0)
        printf ("Unable to close <%s>\n", OUTPUT_FILE);
    else
        printf ("The CRC-32c table has been written to <%s>\n", OUTPUT_FILE);
}

/* 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 t)
{

Stewart, et al.  Expires April 25, 2019
int i;
uint32_t rw = 0UL;

for (i = 0; i < 32; i++) {
    if (b & 1)
        rw |= 1 << (31 - i);
    b >>= 1;
}
return (rw);
}

static uint32_t
build_crc_table(int index)
{
    int i;
    uint32_t rb;

    rb = reflect_32(index);
    for (i = 0; i < 8; i++) {
        if (rb & 0x80000000UL)
            rb = (rb << 1) ^ (uint32_t)CRC32C_POLY;
        else
            rb <<= 1;
    }
    return (reflect_32(rb));
}

int
main (void)
{
    int i;

    printf("nGenerating CRC-32c table file <%s>n", OUTPUT_FILE);
    if (tf = fopen(OUTPUT_FILE, "w")) == NULL) {
        printf("Unable to open %s", OUTPUT_FILE);
        exit (1);  
    }
    fprintf(tf, 
            
#define CRC32C_POLY 0x%08XUL

#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])\n
uint32_t crc_c[256] =
{
for (i = 0; i < 256; i++) {
    fprintf(tf, "0x%08XUL," , build_crc_table (i));
if ((i & 3) == 3)
    fprintf(tf, "\n");
else
    fprintf(tf, " ");
fprintf(tf, "\n\\n#endif\n\n\n\nif (fclose (tf) != 0)
    printf("Unable to close <\%s\.", OUTPUT_FILE);
else
    printf("\nThe CRC-32c table has been written to <\%s\.\n", OUTPUT_FILE);
\}

This text has been modified by multiple errata. It includes modifications from Section 3.10. It is in final form, and is not further updated in this document.

---------
Old text: (Appendix C)
---------

/* Example of crc insertion */

#include "crc32cr.h"

unsigned long
generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    unsigned long crc32 = ~0L;
    unsigned long result;
    unsigned char byte0,byte1,byte2,byte3;

    for (i = 0; i < length; i++){
        CRC32C(crc32, buffer[i]);
    }

    result = ~crc32;

    /* result now holds the negated polynomial remainder;
     * since the table and algorithm is "reflected" [williams95].
     * That is, result has the same value as if we mapped the message
to a polynomial, computed the host-bit-order polynomial
* remainder, performed final negation, then did an end-for-end
* bit-reversal.

Note that a 32-bit bit-reversal is identical to four in-place
8-bit reversals followed by an end-for-end byteswap.
In other words, the bytes of each bit are in the right order,
but the bytes have been byteswapped. So we now do an explicit
byteswap. On a little-endian machine, this byteswap and
the final ntohl cancel out and could be elided.

```c
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 );
}
```

```c
int
insert_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    unsigned long crc32;
    message = (SCTP_message *) buffer;
    message->common_header.checksum = 0L;
    crc32 = generate_crc32c(buffer,length);
    /* and insert it into the message */
    message->common_header.checksum = htonl(crc32);
    return 1;
}
```

```c
int
validate_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    unsigned int i;
    unsigned long original_crc32;
    unsigned long crc32 = ~0L;

    /* save and zero checksum */
    message = (SCTP_message *) buffer;
    original_crc32 = ntohl(message->common_header.checksum);
    message->common_header.checksum = 0L;
    crc32 = generate_crc32c(buffer,length);
    return ((original_crc32 == crc32)? 1 : -1);
}
```
/* Example of crc insertion */

#include "crc32cr.h"

uint32_t generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    uint32_t crc32 = 0xffffffffUL;
    uint32_t result;
    uint8_t byte0, byte1, byte2, byte3;

    for (i = 0; i < length; i++) {
        CRC32C(crc32, buffer[i]);
    }

    result = ~crc32;

    /* result now holds the negated polynomial remainder;
     * since the table and algorithm is "reflected" [williams95].
     * That is, result has the same value as if we mapped the message
     * to a polynomial, computed the host-bit-order polynomial
     * remainder, performed final negation, then did an end-for-end
     * bit-reversal.
     * Note that a 32-bit bit-reversal is identical to four inplace
     * 8-bit reversals followed by an end-for-end byteswap.
     * In other words, the bits of each byte are in the right order,
     * but the bytes have been byteswapped. So we now do an explicit
     * byteswap. On a little-endian machine, this byteswap and
     * the final ntohl cancel out and could be elided.
     */

    byte0 = result & 0xff;
    byte1 = (result>>8) & 0xff;
    byte2 = (result>>16) & 0xff;
    byte3 = (result>>24) & 0xff;
    crc32 = (byte0 << 24) | (byte1 << 16) | (byte2 << 8) | byte3;
    return (crc32);
}

int


```
insert_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    uint32_t crc32;
    message = (SCTP_message *) buffer;
    message->common_header.checksum = 0UL;
    crc32 = generate_crc32c(buffer, length);
    /* and insert it into the message */
    message->common_header.checksum = htonl(crc32);
    return 1;
}

int validate_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    unsigned int i;
    uint32_t original_crc32;
    uint32_t crc32;

    /* save and zero checksum */
    message = (SCTP_message *)buffer;
    original_crc32 = ntohl(message->common_header.checksum);
    message->common_header.checksum = 0L;
    crc32 = generate_crc32c(buffer, length);
    return ((original_crc32 == crc32)? 1 : -1);
}
<CODE ENDS>

This text has been modified by multiple errata. It includes modifications from Section 3.5 and Section 3.10. It is in final form, and is not further updated in this document.

3.46.3. Solution Description

The code was changed to use platform independent types.

3.47. Clarification of Gap Ack Blocks in SACK Chunks

3.47.1. Description of the Problem

The Gap Ack Blocks in the SACK chunk are intended to be isolated. However, this is not mentioned with normative text.

This issue was reported as part of an Errata for [RFC4960] with Errata ID 5202.

3.47.2. Text Changes to the Document

---------
Old text: (Section 3.3.4)
---------

The SACK also contains zero or more Gap Ack Blocks. Each Gap Ack Block acknowledges a subsequence of TSNs received following a break in the sequence of received TSNs. By definition, all TSNs acknowledged by Gap Ack Blocks are greater than the value of the Cumulative TSN Ack.

---------
New text: (Section 3.3.4)
---------

The SACK also contains zero or more Gap Ack Blocks. Each Gap Ack Block acknowledges a subsequence of TSNs received following a break in the sequence of received TSNs. The Gap Ack Blocks SHOULD be isolated. This means that the TSN just before each Gap Ack Block and the TSN just after each Gap Ack Block has not been received. By definition, all TSNs acknowledged by Gap Ack Blocks are greater than the value of the Cumulative TSN Ack.

This text is in final form, and is not further updated in this document.
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:

These fields contain the Gap Ack Blocks. They are repeated for each Gap Ack Block up to the number of Gap Ack Blocks defined in the Number of Gap Ack Blocks field. All DATA chunks with TSNs greater than or equal to (Cumulative TSN Ack + Gap Ack Block Start) and less than or equal to (Cumulative TSN Ack + Gap Ack Block End) of each Gap Ack Block are assumed to have been received correctly. Gap Ack Blocks SHOULD be isolated. That means that the DATA chunks with TSN equal to (Cumulative TSN Ack + Gap Ack Block Start - 1) and (Cumulative TSN Ack + Gap Ack Block End + 1) have not been received.

This text is in final form, and is not further updated in this document.

3.47.3. Solution Description

Normative text describing the intended usage of Gap Ack Blocks has been added.

3.48. Handling of SSN Wrap Arousnd

3.48.1. Description of the Problem

The Stream Sequence Number (SSN) is used for preserving the ordering of user messages within each SCTP stream. The SSN is limited to 16 bits. Therefore, multiple wrap arounds of the SSN might happen within the current send window. To allow the receiver to deliver
ordered user messages in the correct sequence, the sender should limit the number of user messages per stream.

3.48.2. Text Changes to the Document

---------
Old text: (Section 6.1)
---------

Note: The data sender SHOULD NOT use a TSN that is more than 2**31 - 1 above the beginning TSN of the current send window.

---------
New text: (Section 6.1)
---------

Note: The data sender SHOULD NOT use a TSN that is more than 2**31 - 1 above the beginning TSN of the current send window. Note: For each stream, the data sender SHOULD NOT have more than 2**16-1 ordered user messages in the current send window.

This text is in final form, and is not further updated in this document.

3.48.3. Solution Description

The data sender is required to limit the number of ordered user messages within the current send window.

3.49. Update RFC 2119 Boilerplate

3.49.1. Description of the Problem

The text to be used to refer to the [RFC2119] terms has been updated by [RFC8174].

3.49.2. Text Changes to the Document
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].

This text is in final form, and is not further updated in this document.

3.49.3.  Solution Description

The text has been updated to the one specified in [RFC8174].

3.50.  Missed Text Removal

3.50.1.  Description of the Problem

When integrating the changes to Section 7.2.4 of [RFC2960] as described in Section 2.8.2 of [RFC4460] some text was not removed and is therefore still in [RFC4960].

3.50.2.  Text Changes to the Document
Old text: (Section 7.2.4)

A straightforward implementation of the above keeps a counter for each TSN hole reported by a SACK. The counter increments for each consecutive SACK reporting the TSN hole. After reaching 3 and starting the Fast-Retransmit procedure, the counter resets to 0. Because cwnd in SCTP indirectly bounds the number of outstanding TSN’s, the effect of TCP Fast Recovery is achieved automatically with no adjustment to the congestion control window size.

New text: (Section 7.2.4)

This text is in final form, and is not further updated in this document.

3.50.3. Solution Description

The text has finally been removed.

4. IANA Considerations

Section 3.44 of this document updates the port number registry for SCTP to be consistent with [RFC6335]. IANA is requested to review Section 3.44.

IANA is only requested to check if it is OK to make the proposed text change in an upcoming standards track document that updates [RFC4960]. IANA is not asked to perform any other action and this document does not request IANA to make a change to any registry.

5. Security Considerations

This document does not add any security considerations to those given in [RFC4960].

6. Acknowledgments

The authors wish to thank Pontus Andersson, Eric W. Biederman, Cedric Bonnet, Spencer Dawkins, Gorry Fairhurst, Benjamin Kaduk, Mirja Kuehlewind, Peter Lei, Gyula Marosi, Lionel Morand, Jeff Morris, Karen E. E. Nielsen, Tom Petch, Kacheong Poon, Julien Pourtet, Irene Ruengeler, Michael Welzl, and Qiaobing Xie for their invaluable comments.

7. References

7.1. Normative References


7.2. Informative References


Authors’ Addresses

Randall R. Stewart
Netflix, Inc.
Chapin, SC 29036
United States

Email: randall@lakerest.net

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.
[RFC2983]. In such cases, 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;
if the CE counter is positive, it CE-marks the next packet to depart and subtracts its size from the CE counter;

- 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;

- (the previous two steps will often leave a negative remainder in the counters, which is deliberate);

- if neither counter is positive, it marks the next packet to depart as ECT(0);

- 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.

- PPTP (Point-to-Point Tunneling Protocol) [RFC2637];

- 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];

- GRE (Generic Routing Encapsulation) [RFC2784] and NVGRE (Network Virtualization using GRE) [RFC7637];
GTP (GPRS Tunnelling Protocol), specifically GTPv1 [GTPv1], GTP v1 User Plane [GTPv1-U], GTP v2 Control Plane [GTPv2-C];

- Teredo [RFC4380];

- CAPWAP (Control And Provisioning of Wireless Access Points) [RFC5415];

- LISP (Locator/Identifier Separation Protocol) [RFC6830];

- AMT (Automatic Multicast Tunneling) [RFC7450];

- VXLAN (Virtual eXtensible Local Area Network) [RFC7348] and VXLAN-GPE [I-D.ietf-nvo3-vxlan-gpe];

- The Network Service Header (NSH [RFC8300]) for Service Function Chaining (SFC);

- Geneve [I-D.ietf-nvo3-geneve];

- GUE (Generic UDP Encapsulation) [I-D.ietf-intarea-gue];

- Direct tunnelling of an IP packet within a UDP/IP datagram (see Section 3.1.11 of [RFC8085]);

- TCP Encapsulation of IKE and IPsec Packets (see Section 12.5 of [RFC8229]).

Some of the listed protocols enable encapsulation of a variety of network layer protocols as inner and/or outer. This specification applies in the cases where there is an inner and outer IP header as described in Section 3. Otherwise [I-D.ietf-tsvwg-ecn-encap-guidelines] gives guidance on how to design propagation of ECN into other protocols that might encapsulate IP.

Where protocols in the above list need to be updated to specify ECN propagation and they are under IETF change control, update text is given in the following subsections. For those not under IETF control, it is RECOMMENDED that implementations of encapsulation and decapsulation comply with RFC 6040. It is also RECOMMENDED that their specifications are updated to add a requirement to comply with RFC 6040 (as updated by the present document).

PPTP is not under the change control of the IETF, but it has been documented in an informational RFC [RFC2637]. However, there is no need for the present specification to update PPTP because L2TP has been developed as a standardized replacement.
NVGRE is not under the change control of the IETF, but it has been documented in an informational RFC [RFC7637]. NVGRE is a specific use-case of GRE (it re-purposes the key field from the initial specification of GRE [RFC1701] as a Virtual Subnet ID). Therefore the text that updates GRE in Section 6.1.2 below is also intended to update NVGRE.

Although the definition of the various GTP shim headers is under the control of the 3GPP, it is hard to determine whether the 3GPP or the IETF controls standardization of the _process_ of adding both a GTP and an IP header to an inner IP header. Nonetheless, the present specification is provided so that the 3GPP can refer to it from any of its own specifications of GTP and IP header processing.

The specification of CAPWAP already specifies RFC 3168 ECN propagation and ECN capability negotiation. Without modification the CAPWAP specification already interworks with the backward compatible updates to RFC 3168 in RFC 6040.

LISP made the ECN propagation procedures in RFC 3168 mandatory from the start. RFC 3168 has since been updated by RFC 6040, but the changes are backwards compatible so there is still no need for LISP tunnel endpoints to negotiate their ECN capabilities.

VXLAN is not under the change control of the IETF but it has been documented in an informational RFC. In contrast, VXLAN-GPE (Generic Protocol Extension) is being documented under IETF change control. It is RECOMMENDED that VXLAN and VXLAN-GPE implementations comply with RFC 6040 when the VXLAN header is inserted between (or removed from between) IP headers. The authors of any future update to these specifications are encouraged to add a requirement to comply with RFC 6040 as updated by the present specification.

The Network Service Header (NSH [RFC8300]) has been defined as a shim-based encapsulation to identify the Service Function Path (SFP) in the Service Function Chaining (SFC) architecture [RFC7665]. A proposal has been made for the processing of ECN when handling transport encapsulation [I-D.ietf-sfc-nsh-ecn-support].

The specifications of Geneve and GUE already refer to RFC 6040 for ECN encapsulation.

Section 3.1.11 of RFC 8085 already explains that a tunnel that encapsulates an IP header within a UDP/IP datagram needs to follow RFC 6040 when propagating the ECN field between inner and outer IP headers. The requirements in Section 4 update RFC 6040, and hence implicitly update the UDP usage guidelines in RFC 8085 to add the important but previously unstated requirement that, if the UDP tunnel
egress does not, or might not, support ECN propagation, a UDP tunnel ingress has to clear the outer IP ECN field to 0b00, e.g. by configuration.

Section 12.5 of TCP Encapsulation of IKE and IPsec Packets [RFC8229] already recommends the compatibility mode of RFC 6040 in this case, because there is not a one-to-one mapping between inner and outer packets.

6.1. Specific Updates to Protocols under IETF Change Control

6.1.1. L2TP (v2 and v3) ECN Extension

The L2TP terminology used here is defined in [RFC2661] and [RFC3931].

L2TPv3 [RFC3931] is used as a shim header between any packet-switched network (PSN) header (e.g. IPv4, IPv6, MPLS) and many types of layer 2 (L2) header. The L2TPv3 shim header encapsulates an L2-specific sub-layer then an L2 header that is likely to contain an inner IP header (v4 or v6). Then this whole stack of headers can be encapsulated optionally within an outer UDP header then an outer PSN header that is typically IP (v4 or v6).

L2TPv2 is used as a shim header between any PSN header and a PPP header, which is in turn likely to encapsulate an IP header.

Even though these shims are rather fat (particularly in the case of L2TPv3), they still fit the definition of a tightly coupled shim header over an encapsulating header (Section 3.1), because all the headers encapsulating the L2 header are added (or removed) together. L2TPv2 and L2TPv3 are therefore within the scope of RFC 6040, as updated by Section 3 above.

L2TP maintainers are RECOMMENDED to implement the ECN extension to L2TPv2 and L2TPv3 defined in Section 6.1.1.2 below, in order to provide the benefits of ECN [RFC8087], whenever a node within an L2TP tunnel becomes the bottleneck for an end-to-end traffic flow.

6.1.1.1. Safe Configuration of a ‘Non-ECN’ Ingress LCCE

The following text is appended to both Section 5.3 of [RFC2661] and Section 4.5 of [RFC3931] as an update to the base L2TPv2 and L2TPv3 specifications:

The operator of an LCCE that does not support the ECN Extension in Section 6.1.1.2 of RFCXXXX MUST follow the configuration requirements in Section 4 of RFCXXXX to ensure it clears the outer IP ECN field to 0b00 when the outer PSN header is IP (v4 or v6).
(RFCXXXX refers to the present document so it will need to be
inserted by the RFC Editor)

In particular, for an LCCE implementation that does not support the
ECN Extension, this means that configuration of how it propagates the
ECN field between inner and outer IP headers MUST be independent of
any configuration of the Diffserv extension of L2TP [RFC3308].

6.1.1.2. ECN Extension for L2TP (v2 or v3)

When the outer PSN header and the payload inside the L2 header are
both IP (v4 or v6), to comply with RFC 6040, an LCCE will follow the
rules for propagation of the ECN field at ingress and egress in
Section 4 of RFC 6040 [RFC6040].

Before encapsulating any data packets, RFC 6040 requires an ingress
LCCE to check that the egress LCCE supports ECN propagation as
defined in RFC 6040 or one of its compatible predecessors ([RFC4301]
or the full functionality mode of [RFC3168]). If the egress supports
ECN propagation, the ingress LCCE can use the normal mode of
encapsulation (copying the ECN field from inner to outer).
Otherwise, the ingress LCCE has to use compatibility mode [RFC6040]
clearing the outer IP ECN field to 0b00).

An LCCE can determine the remote LCCE’s support for ECN either
statically (by configuration) or by dynamic discovery during setup of
each control connection between the LCCEs, using the Capability AVP
defined in Section 6.1.1.2.1 below.

Where the outer PSN header is some protocol other than IP that
supports ECN, the appropriate ECN propagation specification will need
to be followed, e.g. "Explicit Congestion Marking in MPLS" [RFC5129]. Where no specification exists for ECN propagation by a
particular PSN, [I-D.ietf-tsvwg-ecn-encap-guidelines] gives general
guidance on how to design ECN propagation into a protocol that
encapsulates IP.

6.1.1.2.1. LCCE Capability AVP for ECN Capability Negotiation

The LCCE Capability Attribute-Value Pair (AVP) defined here has
Attribute Type ZZ. The Attribute Value field for this AVP is a bit-
mask with the following 16-bit format:
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

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| V=0 | Type=3 | Reserved | E | P | Request Nonce |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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


Author’s Address

Bob Briscoe
Independent
UK

EMail: ietf@bobbriscoe.net
URI:   http://bobbriscoe.net/
Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC) Schemes for FECFRAME

draft-ietf-tsvwg-rlc-fec-scheme-16

Abstract

This document describes two fully-specified Forward Erasure Correction (FEC) Schemes for Sliding Window Random Linear Codes (RLC), one for RLC over the Galois Field (A.K.A. Finite Field) GF(2), a second one for RLC over the Galois Field GF(2^{8}), each time with the possibility of controlling the code density. They can protect arbitrary media streams along the lines defined by FECFRAME extended to sliding window FEC codes. These sliding window FEC codes rely on an encoding window that slides over the source symbols, generating new repair symbols whenever needed. Compared to block FEC codes, these sliding window FEC codes offer key advantages with real-time flows in terms of reduced FEC-related latency while often providing improved packet erasure recovery capabilities.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 20, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.
This Internet-Draft is submitted in connection with the IETF Trust's
License Program. The IETF Trust operates a License Program that
licenses the use of the Internet specifications and related material
found in the internet drafts and RFCs. You can view an up-to-date
list of Internet Drafts at http://datatracker.ietf.org/drafts/current/
and the RFCs at http://www.rfc-editor.org/.

Internet-Draft               RLC FEC Scheme                    June 2019

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

Table of Contents

1.  Introduction . . . . . . . . . . . . . . . . . . . . . . . .   3
  1.1.  Limits of Block Codes with Real-Time Flows . . . . . . .   4
  1.2.  Lower Latency and Better Protection of Real-Time Flows
       with the Sliding Window RLC Codes . . . . . . . . . . .   4
  1.3.  Small Transmission Overheads with the Sliding Window RLC
       FEC Scheme . . . . . . . . . . . . . . . . . . . . . .   5
  1.4.  Document Organization . . . . . . . . . . . . . . . . . .   6
2.  Definitions and Abbreviations . . . . . . . . . . . . . . . .   6
3.  Common Procedures . . . . . . . . . . . . . . . . . . . . . . .  7
  3.1.  Codec Parameters . . . . . . . . . . . . . . . . . . . . .  7
  3.2.  ADU, ADUI and Source Symbols Mappings . . . . . . . . . .  9
  3.3.  Encoding Window Management . . . . . . . . . . . . . . .  10
  3.4.  Source Symbol Identification . . . . . . . . . . . . . .  11
  3.5.  Pseudo-Random Number Generator (PRNG) . . . . . . . . . .  11
  3.6.  Coding Coefficients Generation Function . . . . . . . . .  13
  3.7.  Finite Fields Operations . . . . . . . . . . . . . . . .  15
    3.7.1.  Finite Field Definitions . . . . . . . . . . . . . .  15
    3.7.2.  Linear Combination of Source Symbols Computation . . .  15
4.  Sliding Window RLC FEC Scheme over GF(2^^8) for Arbitrary
    Packet Flows . . . . . . . . . . . . . . . . . . . . . . . .  16
  4.1.  Formats and Codes . . . . . . . . . . . . . . . . . . . .  16
    4.1.1.  FEC Framework Configuration Information . . . . . . .  16
    4.1.2.  Explicit Source FEC Payload ID . . . . . . . . . . .  18
    4.1.3.  Repair FEC Payload ID . . . . . . . . . . . . . . .  18
  4.2.  Procedures . . . . . . . . . . . . . . . . . . . . . . .  20
5.  Sliding Window RLC FEC Scheme over GF(2) for Arbitrary Packet
    Flows . . . . . . . . . . . . . . . . . . . . . . . . . . . .  20
  5.1.  Formats and Codes . . . . . . . . . . . . . . . . . . . .  20
    5.1.1.  FEC Framework Configuration Information . . . . . . .  20
    5.1.2.  Explicit Source FEC Payload ID . . . . . . . . . . .  20
    5.1.3.  Repair FEC Payload ID . . . . . . . . . . . . . . .  20
  5.2.  Procedures . . . . . . . . . . . . . . . . . . . . . . .  21
6.  FEC Code Specification . . . . . . . . . . . . . . . . . . .  21
  6.1.  Encoding Side . . . . . . . . . . . . . . . . . . . . . .  21
  6.2.  Decoding Side . . . . . . . . . . . . . . . . . . . . . .  22
7.  Implementation Status . . . . . . . . . . . . . . . . . . . .  22

Roca & Teibi            Expires December 20, 2019               [Page 2]
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 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:

\[ a^{\cdot b} \quad a \text{ to the power of } b \]

\[ GF(q) \quad \text{denotes a finite field (also known as the Galois Field) with } q \text{ elements. We assume that } q = 2^{\cdot m} \text{ in this document} \]

\[ m \quad \text{defines the length of the elements in the finite field, in bits. In this document, } m \text{ is equal to 1 or 8} \]

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)
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, $\text{max\_lat}$ (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, $\text{max\_lat}$, 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 $\text{max\_lat}$). $\text{max\_lat}$ 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, $ew_{\text{size}}$ (resp. $ew_{\text{max\_size}}$) (in symbols):

- At a FECFRAME sender, during FEC encoding, a repair symbol is computed as a linear combination of the $ew_{\text{size}}$ source symbols present in the encoding window. The $ew_{\text{max\_size}}$ is the maximum size of this window, while $ew_{\text{size}}$ is the current size. For example, in the common case at session start, upon receiving new source ADUs, the $ew_{\text{size}}$ progressively increases until it reaches its maximum value, $ew_{\text{max\_size}}$. We have:

\[0 < ew_{\text{size}} \leq ew_{\text{max\_size}}\]

Decoding window maximum size, $dw_{\text{max\_size}}$ (in symbols): at a FECFRAME receiver, $dw_{\text{max\_size}}$ is the maximum number of received or lost source symbols that are still within their latency budget;

Linear system maximum size, $ls_{\text{max\_size}}$ (in symbols): at a FECFRAME receiver, the linear system maximum size, $ls_{\text{max\_size}}$, is the maximum number of received or lost source symbols in the linear system (i.e., the variables). It SHOULD NOT be smaller than $dw_{\text{max\_size}}$ 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 $dw_{\text{max\_size}}$ (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: $0 < cr \leq 1$. This is an input parameter that enables a FECFRAME sender to derive other internal parameters, as explained below. However, there is no need to communicate the $cr$ parameter per see (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.
symbol length, E
< ------------------ >< ------------------ >< ------------------ >
|[F| L|                     ADU                     |     Pad     |
+-------------------------+-------------------+
Figure 1: ADUI Creation example (here 3 source symbols are created for this ADUI).

Note that neither the initial 3 bytes nor the optional padding are sent over the network. However, they are considered during FEC encoding, and a receiver who lost a certain FEC Source Packet (e.g., the UDP datagram containing this FEC Source Packet when UDP is used as the transport protocol) will be able to recover the ADUI if FEC decoding succeeds. Thanks to the initial 3 bytes, this receiver will get rid of the padding (if any) and identify the corresponding ADU flow.

3.3. Encoding Window Management

Source symbols and the corresponding ADUs are removed from the encoding window:

o when the sliding encoding window has reached its maximum size, ew_max_size. In that case the oldest symbol MUST be removed before adding a new symbol, so that the current encoding window size always remains inferior or equal to the maximum size: ew_size <= ew_max_size;

o when an ADU has reached its maximum validity duration in case of a real-time flow. When this happens, all source symbols corresponding to the ADUI that expired SHOULD be removed from the encoding window;

Source symbols are added to the sliding encoding window each time a new ADU arrives, once the ADU-to-source symbols mapping has been performed (Section 3.2). The current size of the encoding window, ew_size, is updated after adding new source symbols. This process may require to remove old source symbols so that: ew_size <= ew_max_size.

Note that a FEC codec may feature practical limits in the number of source symbols in the encoding window (e.g., for computational complexity reasons). This factor may further limit the ew_max_size value, in addition to the maximum FEC-related latency budget (Section 3.1).
3.4. Source Symbol Identification

Each source symbol is identified by an Encoding Symbol ID (ESI), an unsigned integer. The ESI of source symbols MUST start with value 0 for the first source symbol and MUST be managed sequentially. Wrapping to zero happens after reaching the maximum value made possible by the ESI field size (this maximum value is FEC Scheme dependant, for instance, $2^{32}-1$ with FEC Schemes XXX and YYY).

No such consideration applies to repair symbols.

3.5. Pseudo-Random Number Generator (PRNG)

In order to compute coding coefficients (see Section 3.6), the RLC FEC Schemes rely on the TinyMT32 PRNG defined in [tinymt32] with two additional functions defined in this section.

This PRNG MUST first be initialized with a 32-bit unsigned integer, used as a seed, with:

```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<sup>8</sup>), 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<sup>8</sup>) 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<sup>8</sup>)), otherwise
 *                    a fraction of them will be 0.
 * (in) m             Finite Field GF(2<sup>m</sup>) 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.
*/
</CODE BEGINS>
int generate_coding_coefficients (uint16_t  repair_key,
             uint8_t*  cc_tab,
             uint16_t  cc_nb,
             uint8_t  dt,
             uint8_t  m)
{
    uint32_t      i;
    tinymt32_t    s;    /* PRNG internal state */

    if (dt > 15) {
        return -1; /* error, bad dt parameter */
    }
    switch (m) {
        case 1:
            if (dt == 15) {
                /* all coefficients are 1 */
                memset(cc_tab, 1, cc_nb);
            } else {
                /* here coefficients are either 0 or 1 */
                tinymt32_init(&s, repair_key);
                for (i = 0 ; i < cc_nb ; i++) {
                    cc_tab[i] = (tinymt32_rand16(&s) <= dt) ? 1 : 0;
                }
            }
            break;
        case 8:
            tinymt32_init(&s, repair_key);
            if (dt == 15) {
                /* coefficient 0 is avoided here in order to include
                 * all the source symbols */
                for (i = 0 ; i < cc_nb ; i++) {
                    do {
                        cc_tab[i] = (uint8_t) tinymt32_rand256(&s);
                    } while (cc_tab[i] == 0);
                }
            } else {
                /* here a certain number of coefficients should be 0 */
                for (i = 0 ; i < cc_nb ; i++) {
                    if (tinymt32_rand16(&s) <= dt) {
                        do {
                            cc_tab[i] = (uint8_t) tinymt32_rand256(&s);
                        } while (cc_tab[i] == 0);
                    } else {
                        cc_tab[i] = 0;
                    }
                }
            }
            break;
    }
    return 0;
}
break;

default:
    return -2; /* error, bad parameter m */
}
return 0; /* success */
</CODE ENDS>

Figure 3: Coding Coefficients Generation Function Reference Implementation

3.7. Finite Fields Operations

3.7.1. Finite Field Definitions

The two RLC FEC Schemes specified in this document reuse the Finite Fields defined in [RFC5510], section 8.1. More specifically, the elements of the field GF(2^m) are represented by polynomials with binary coefficients (i.e., over GF(2)) and degree lower or equal to m-1. The addition between two elements is defined as the addition of binary polynomials in GF(2), which is equivalent to a bitwise XOR operation on the binary representation of these elements.

With GF(2^8), multiplication between two elements is the multiplication modulo a given irreducible polynomial of degree 8. The following irreducible polynomial is used for GF(2^8):

\[ x^8 + x^4 + x^3 + x^2 + 1 \]

With GF(2), multiplication corresponds to a logical AND operation.

3.7.2. Linear Combination of Source Symbols Computation

The two RLC FEC Schemes require the computation of a linear combination of source symbols, using the coding coefficients produced by the generate_coding_coefficients() function and stored in the cc_tab[] array.

With the RLC over GF(2^8) FEC Scheme, a linear combination of the ew_size source symbol present in the encoding window, say src_0 to src_ew_size_1, in order to generate a repair symbol, is computed as follows. For each byte of position i in each source and the repair symbol, where i belongs to [0; E-1], compute:

\[ \text{repair}[i] = \text{cc_tab}[0] \times \text{src}_0[i] \text{ XOR cc_tab}[1] \times \text{src}_1[i] \text{ XOR } \ldots \text{ XOR cc_tab}[\text{ew_size} - 1] \times \text{src}_\text{ew_size}_1[i] \]
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.

<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 2 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoding Symbol Length (E)</td>
</tr>
<tr>
<td>+----------------------------+-------</td>
</tr>
</tbody>
</table>

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.

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.

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 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):
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.

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 FFCl 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:

- 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:

- 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;
- 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 focuses 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 focuses 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 * 100 = 6.25% of times.

Since the RLC FEC Schemes use of this PRNG will be limited to 16-bit seed values only. The distribution (not shown) is of course less uniform, with value occurrences ranging between 6.2121% (i.e., 81,423 occurrences out of a total of 65536*20=1,310,720) and 6.2948% (i.e., 82,507 occurrences). However, we do not believe it significantly impacts the RLC FEC Scheme behavior.

Other types of biases may exist that may be visible with smaller tests, for instance to evaluate the convergence speed to a uniform distribution. We therefore perform 200 tests, each of them consisting in producing 200 sequences, keeping only the first value of each sequence. We use non overlapping repair keys for each sequence, starting with value 0 and increasing it after each use.

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.
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:

\[
dw_{\text{max size}} = \frac{\text{max_lat} \times br_{\text{in}}}{8 \times E}
\]

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:

\[
dw_{\text{max\_size}} = (\text{max\_lat} \times br_{out} \times cr) / (8 \times E)
\]

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:

\[
ew_{\text{max\_size}} = dw_{\text{max\_size}} \times WSR / 255
\]

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:

\[
dw_{\text{max\_size}} = \text{max\_NSS\_observed} \times 255 / WSR
\]

A receiver can then chose an appropriate linear system maximum size:

\[
ls_{\text{max\_size}} \geq dw_{\text{max\_size}}
\]

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 \( \text{ew}_{\text{max}} \) 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 \( \text{dw}_{\text{max}} \) size and derive an appropriate \( \text{ls}_{\text{max}} \) size as explained in Appendix C.1.

When the observed NSS fluctuates significantly, a FECFRAME receiver may want to adapt its \( \text{ls}_{\text{max}} \) size accordingly. In particular when the NSS is significantly reduced, a FECFRAME receiver may want to reduce the \( \text{ls}_{\text{max}} \) size too in order to limit computation complexity. A balance must be found between using an \( \text{ls}_{\text{max}} \) 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 \( \text{ls}_{\text{max}} \) parameter. The same considerations also apply to the FECFRAME sender, where the maximum memory amount and computation complexity depend on the \( \text{ew}_{\text{max}} \) 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 \( \text{ew}_{\text{max}} \) 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:

\[
\text{ls}_{\text{max}} > \text{dw}_{\text{max}}
\]
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.

![Figure 13: Relationship between parameters to decode beyond maximum latency.](image)

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
Abstract

This document describes some implications of applying end-to-end encryption at the transport layer. It first identifies in-network uses of transport layer header information. Then, it reviews some implications of developing end-to-end transport protocols that use encryption to provide confidentiality of the transport protocol headers, or that use authentication to protect the integrity of transport header information. Since measurement and analysis of the impact of network characteristics on transport protocols has been important to the design of current transports, it also considers the impact of transport encryption 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 February 24, 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...
Internet-Draft            Transport Encryption               August 2019

this document strongly supports the increased use of encryption in transport protocols. Encryption and authentication 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 headers, and considers the effect of such changes on transport protocol design and network operations. It also considers some 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 document.

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 the results of measurements 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. The widespread use of transport header encryption may change this.

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 enhance performance. This 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 cases, 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).

As an example of ossification, consider the experience of developing Transport Layer Security (TLS) 1.3 [RFC8446]. This required a design that recognised that deployed middleboxes relied on the presence of certain header file exposed in TLS 1.2, and failed if those headers were changed. Other 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 all these 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. These can be in the form of encrypted transport protocols (i.e., transport protocols that use encryption to provide confidentiality of some or all of the transport-layer header information), and/or the encryption of transport payloads (i.e., confidentiality of the payload data). There are many motivations for deploying such transports, and increasing public concerns about interference with Internet traffic [RFC7624] have led to a rapidly expanding deployment of encrypted transport protocols such as QUIC [I-D.ietf-quic-transport]. Using encryption to provide confidentiality of the transport layer therefore brings some well-known privacy and security benefits.

Authentication and the introduction of cryptographic integrity checks for header fields can prevent undetected manipulation of transport headers by network devices. This does not prevent inspection of the information by devices 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 the certain header information observable by the network. This supports use of this information by on-path devices, but at the same time 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 or value of exposed header fields [I-D.ietf-tls-grease].

A protocol design that uses header encryption with secure key distribution can provide confidentiality for some, or all, of the protocol header information. This prevents an on-path device from observing the transport headers, and stops mechanisms being built that directly rely on transport header information, or that seek to infer semantics of exposed header fields. Transport header encryption 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. This can create 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 measure and analysis 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. Operatioal 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, such as changes in application behaviour or distributed denial of service attacks. For this application to be effective, it needs to be possible for an operator to uniquely disambiguate unwanted traffic. Concealing transport header information would prevent disambiguation based on transport information. This could result in less-efficient identification of unwanted traffic, the use of heuristics to identify anomalous flows, or the introduction of rate limits for uncharacterised traffic.
Network Troubleshooting and Diagnostics: Observable transport headers can be utilised by operators for network troubleshooting and diagnostics. Flows experiencing packet loss or jitter are hard to distinguish from unaffected flows when only observing network layer headers. Effective troubleshooting often requires visibility into the transport layer behaviour.

Concealing transport header information reduces 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 make it harder to localise the network segment introducing 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, but in other applications it is important to relate observations to specific equipment/configurations or particular 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 and need visibility to explore issues relating to Quality of Service (QoS), the ability to perform fast re-routing of critical traffic, or to mitigate the characteristics of specific radio links, and so on.

Open and Verifiable Network Data: Observable transport headers can provide open and verifiable measurement data. The ability of other stakeholders 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 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.

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.

Different parties will view the relative importance of these issues differently. For some, the benefits of encrypting some, or all, of the transport headers may outweigh the impact of doing so; others might make a different trade-off. The purpose of highlighting the trade-offs is to make such analysis possible.

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 affects how protocol information is used [RFC8404], requiring consideration of the trade-offs discussed in Section 2.3. To understand the implications, it is necessary to understand how transport layer headers are currently observed and/or modified by middleboxes within the network.

We review some current uses in the following section. This does not consider the intentional modification of transport headers by middleboxes (such as in Network Address Translation, NAT, or Firewalls). 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 [RFC8546]. 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 Using Transport Layer Headers

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 flows and their connection state, together with the protocol options being used. 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. They also have the possibility to negotiate additional headers at connection setup, identified by an option number in the transport header.

In some uses, a low-numbered (well-known) transport port number can be used to identify the 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.

UDP-based protocols often 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, so 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, including heuristics based on the analysis of traffic patterns. For example, an operator that no longer has access to Session Description Protocol (SDP) session descriptions to classify a flow carry as audio traffic might instead use heuristics to infer that short UDP packets with regular spacing carry audio traffic. Operational practices aimed at inferring 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 measurement and analysis of protocol performance, network anomalies, and failure pathologies at any point along the Internet path. Some operators 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, and make inferences from transport headers to derive performance metrics.

A variety of open source and commercial tools have been deployed that utilise transport header information in this way. The following metrics can be derived:

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 pre-emption. Understanding flow loss rates requires either observing sequence numbers in transport headers, or maintaining per-flow packet counters (but note that flow identification often requires transport header information). Per-hop 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 per-flow 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 where 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: Throughput is the amount of data sent by a flow per time interval. Goodput [RFC7928] is a measure of useful data exchanged (the ratio of useful data to total volume of traffic sent by a flow). The throughput achieved by a flow can be determined even when transport header information is concealed, providing the individual flow can be identified. Goodput requires ability to differentiate loss and retransmission of packets, for example 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 and user-perceived response times. 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 [bufferbloat]. Once the cause of unwanted latency has been identified, this can often be eliminated.

To measure latency across a part of a path, an observation point [RFC7799] can measure the experienced round trip time (RTT) using packet sequence numbers, and acknowledgements, or by observing header timestamp information. Such information allows an observation point 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 configuration changes and to tune deployed services. Latency metrics are key to evaluating and deploying AQM [RFC7567], DiffServ [RFC2474], and Explicit Congestion Notification (ECN) [RFC3168] [RFC8087]. Measurements could identify excessively large buffers, indicating where to deploy or configure AQM. An AQM method is often deployed in combination with other techniques, such as scheduling [RFC7567] [RFC8290] and although parameter-less methods are desired [RFC7567], current methods [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. For example, 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 for observable transport headers 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, 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 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.

The user of a transport that multiplexes multiple sub-flows might want 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 flow label can provide information that can help inform network-layer queuing and forwarding [RFC6438], for example with Equal Cost Multi-Path routing and Link Aggregation [RFC6294]. Considerations when using IPsec are further described in [RFC6438].

**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 field when a network path has support for differentiated service treatment.

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 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 inference and other heuristics. 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) are 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 visualize 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 (e.g., using UDP) 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 RTP and RTCP header information for real-time flows (see Section 3.1.2). The Secure RTP extensions [RFC3711] were
explicitly designed to expose some header information to enable such observation, while protecting the payload data.

### 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. Section 3.1.2 and Section 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 capacity 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 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 can impact the efficiency of a link/path.
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 [RFC8546].

There are several motivations for encryption:

- 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 could 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 use by middleboxes 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), 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]) to supplement the already widespread payload encryption.

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. This 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, allows 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: 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 [RFC8548] 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 the 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 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

An on-path device can make measurements by appending 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 802.1lag 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 measurement results suggest that it can be undesirable to rely on methods requiring end to end support of network options or extension headers across the Internet. IPv4 network options are often not supported (or are carried on a slower processing path) and some IPv6 networks have been observed to drop packets that set an IPv6 header extension (e.g., results from 2016 in [RFC7872]). Another possibility is that protocols that separately expose header information do not necessarily have an incentive to expose the actual information that is utilised by the protocol itself and could therefore manipulate the exposed header information to gain an advantage from the network. The incentive to reflect actual transport information needs to be considered when proposing a method that selectively exposes header information.

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 if the transport community is to maintain an accurate understanding of the network. Encrypting transport header encryption changes the ability 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 for loss detection and recovery, congestion detection and congestion control. Others 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, many 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 became 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, since the protocol will not work otherwise. This increases 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 or 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 behaviour of a specific mechanism with the configuration and traffic using operational equipment (e.g., combining transport and network measurements to explore congestion control dynamics, the implications of designs for active queue management or circuit breakers).

For some usage a standardised endpoint-based logging format (e.g., based on Quic-Trace [Quic-Trace]) could offer an alternative for some in-network measurement. Such information will have a diversity of uses, including developers wishing to debug/understand the transport/application protocols with which they work, researchers seeking to spot trends and anomalies, and to characterise variants of protocols. Measurements based on logging will need to establish the validity and provenance of the logged information 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 path, 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 combinations 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, DCTCP, and methods proposed for L4S). The growth and diversity of applications and protocols using the Internet also continues to expand. For each new method or application it is desirable to build a body of data reflecting its behaviour under a wide range of deployment scenarios, traffic load, and interactions with other deployed/candidate methods.

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.
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]).

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 levels (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 stakeholders, 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.

There are trade-offs and implications of increased use of encryption when designing a protocol. 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 header information. This lack of visibility of transport header information can be expected to 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 could 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 analysed [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 throughout this document.
Authentication, confidentiality protection, and integrity protection are identified as Transport Features by [RFC8095]. As currently deployed in the Internet, these features are generally provided by a protocol or layer on top of the transport protocol [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 seeks 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 against 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.
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-tls-grease]
[I-D.ietf-tsvwg-rtcweb-qos]

[I-D.trammell-plus-abstract-mech]


[Quic-Trace]
"https://QUIC trace utilities //github.com/google/quic-trace".


Internet-Draft            Transport Encryption               August 2019

[ RFC 6347 ] Rescorla, E. and N. Modadugu, "Datagram Transport Layer
                    Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347,

[ RFC 6437 ] Amante, S., Carpenter, B., Jiang, S., and J. Rajahalme,
                    "IPv6 Flow Label Specification", RFC 6437,
                    DOI 10.17487/RFC6437, November 2011,

                    for Equal Cost Multipath Routing and Link Aggregation in
                    Tunnels", RFC 6438, DOI 10.17487/RFC6438, November 2011,

                    Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May

[ RFC 7525 ] Sheffer, Y., Holz, R., and P. Saint-Andre,
                    "Recommendations for Secure Use of Transport Layer
                    Security (TLS) and Datagram Transport Layer Security
                    (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May

                    Recommendations Regarding Active Queue Management",
                    BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015,

[ RFC 7594 ] Eardley, P., Morton, A., Bagnulo, M., Burbridge, T.,
                    Aitken, P., and A. Akhter, "A Framework for Large-Scale
                    Measurement of Broadband Performance (LMAp)", RFC 7594,
                    DOI 10.17487/RFC7594, September 2015,

[ RFC 7624 ] Barnes, R., Schneier, B., Jennings, C., Hardie, T.,
                    Trammell, B., Huiitema, C., and D. Borkmann,
                    "Confidentiality in the Face of Pervasive Surveillance: A
                    Threat Model and Problem Statement", RFC 7624,
                    DOI 10.17487/RFC7624, August 2015,

                    Hybrid Types In-Between)", RFC 7799, DOI 10.17487/RFC7799,


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
 Added some text on TLS story (additional input sought on relevant considerations).

Section 2, paragraph 8 - changed to be clearer, in particular, added "Encryption with secure key distribution prevents"

Flow label description rewritten based on PS/BCP RFCs.

Clarify requirements from RFCs concerning the IPv6 flow label and highlight ways it can be used with encryption. (section 3.1.3)

Add text on the explicit spin-bit work in the QUIC DT. Added greasing of spin-bit. (Section 6.1)

Updated section 6 and added more explanation of impact on operators.

Other comments addressed.

-05 Editorial pass and minor corrections noted on TSVWG list.

-06 Updated conclusions and minor corrections. Responded to request to add OAM discussion to Section 6.1.

-07 Addressed feedback from Ruediger and Thomas.

Section 2 deserved some work to make it easier to read and avoid repetition. This edit finally gets to this, and eliminates some duplication. This also moves some of the material from section 2 to reform a clearer conclusion. The scope remains focussed on the usage of transport headers and the implications of encryption - not on proposals for new techniques/specifications to be developed.

-08 Addressed feedback and completed editorial work, including updating the text referring to RFC7872, in preparation for a WGLC.

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/
Transport Options for UDP
draft-ietf-tsvwg-udp-options-08.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 March 12, 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 (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
Abstract

Transport protocols are extended through the use of transport header options. This document extends UDP by indicating the location, syntax, and semantics for UDP transport layer options.

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)................................9
   5.2. No Operation (NOP).......................................10
   5.3. Option Checksum (OCS)..................................10
   5.4. Alternate Checksum (ACS).................................11
   5.5. Lite (LITE)..............................................12
   5.6. Maximum Segment Size (MSS)...............................14
   5.7. Fragmentation (FRAG)....................................15
   5.8. Coupling FRAG with LITE................................17
   5.9. Timestamps (TIME)........................................18
   5.10. Authentication and Encryption (AE)......................19
6. Echo request (REQ) and echo response (RES)....................20
   6.1. Experimental (EXP).......................................20
7. Rules for designing new options...............................21
8. Option inclusion and processing...............................22
9. UDP API Extensions............................................24
10. Whose options are these?....................................24
11. UDP options LITE option vs. UDP-Lite........................25
12. Interactions with Legacy Devices.............................26
13. Options in a Stateless, Unreliable Transport Protocol......26
14. UDP Option State Caching.....................................27
15. Updates to RFC 768...........................................27
16. Multicast Considerations....................................27
17. Security Considerations......................................28
18. IANA Considerations..........................................28
19. References...................................................29
   19.1. Normative References................................29
   19.2. Informative References...............................29
20. Acknowledgments..............................................31

Touch                   Expires March 12, 2020                 [Page 2]
1. Introduction

Transport protocols use options as a way to extend their capabilities. TCP [RFC793], SCTP [RFC4960], and DCCP [RFC4340] include space for these options but UDP [RFC768] currently does not. This document defines an extension to UDP that provides space for transport options including their generic syntax and semantics for their use in UDP's stateless, unreliable message protocol.

2. Conventions used in this document

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

In this document, the characters ">>" preceding an indented line(s) indicates a statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the portions of this RFC covered by these key words.

3. 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].

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...
Internet-Draft        Transport Options for UDP         September 2019

protection. This document 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 * 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 - 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

The Kind field is always one byte. The Length field is one byte for all lengths below 255 (including the Kind and Length bytes). A Length of 255 indicates use of the UDP option extended format shown in Figure 5. The Extended Length field is a 16-bit field in network standard byte order.
UDP options MAY begin 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.

Option Lengths (or Extended Lengths, where applicable) smaller than the minimum for the corresponding Kind and default format MUST be treated as an error.

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>3</td>
<td>Option checksum (OCS)</td>
</tr>
<tr>
<td>3*</td>
<td>6</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 "*" above: 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).

>> Options with fixed lengths MUST use the default option format.

>> Options with variable lengths MUST use the default option format where their total length is 254 bytes or less.

>> Options using the extended option format MUST indicate extended lengths of 255 or higher; smaller extended length values MUST be treated as an error.

>> 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 6 UDP EOL option format

>> When the UDP options do not consume the entire option area, the last non-NOP option MUST be EOL.

>> All bytes in the surplus area after EOL MUST be zero.

Requiring the post-option surplus area to be zero prevents side-channel uses of this area, requiring instead that all use of the surplus area be UDP options supported by both endpoints. It is
useful to allow for such padding to increase the packet length without affecting the payload length, e.g., for UDP PLPMTUD [Fa19].

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.

```
+--------+
<table>
<thead>
<tr>
<th>Kind=1</th>
</tr>
</thead>
</table>
```

Figure 7 UDP NOP option format

>> 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) option is conventional Internet checksum [RFC791] that covers all of the surplus area. The primary purpose of OCS is to detect non-standard (i.e., non-option) uses of that area.

OCS is calculated by computing the Internet checksum over the surplus area. OCS protects the option area from errors in a similar way that the UDP checksum protects the UDP user data (when not zero).

```
+-----------------------------+
<table>
<thead>
<tr>
<th>Kind=2</th>
<th>checksum</th>
</tr>
</thead>
</table>
+-----------------------------+
```

Figure 8 UDP OCS option format

>> OCS is REQUIRED when the UDP checksum is nonzero and UDP options are present.
>> When present, OCS SHOULD occur as early as possible, preceded by only NOP options for alignment and the LITE option if present.

>> OCS MUST be half-word coordinated with the start of the UDP options area.

This coordination is accomplished by computing the Internet checksum over the surplus 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 the UDP checksum is zero. When not used, OCS is assumed to be "correct" for the purpose of accepting UDP packets at a receiver (see Section 8).

OCS is intended to check for accidental errors, not for attacks.

5.4. Alternate Checksum (ACS)

The Alternate Checksum (ACS) option provides a stronger alternative to the checksum in the UDP header, using a 32-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.

CRC32c has been chosen because of its ubiquity and use in other Internet protocols, including iSCSI and SCTP. The option contains CRC32c in network standard byte order, as described in [RFC3385].

```
+--------+--------+--------+--------+
| Kind=3 | Len=6  |    CRC32c...    |
+--------+--------+--------+--------+
|  CRC32c (cont.) |
+-------------------+
```

Figure 9 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 10   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 11).

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 12). 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  |
+---------+--------------+--------------+------------------+
<------------------------>
    UDP Length
```

Figure 11   LITE option formation - LITE goes first

```
+---------+--------------+--------------+------------------+
| UDP Hdr |  user data   |  LITE data   |LITE| other opts  |
+---------+--------------+--------------+------------------+
    ^^^^^  ^^^^  
     +--------------+
```

Figure 12   Before sending swap LITE option and front of LITE data
The resulting packet has the format shown in Figure 13. 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 13  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 11.

3. Continue processing the remainder of the options, which are now
in the format shown in Figure 12.

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) option 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 14  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 (FRAG) option supports UDP fragmentation and
reassembly, which can be used to transfer UDP messages larger than
limited by the IP receive MTU (EMTU_R [RFC1122]). 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 15  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 16.
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).

<table>
<thead>
<tr>
<th>Kind=6</th>
<th>Len=10</th>
<th>Frag. Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Identification</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Checksum</td>
<td></td>
</tr>
</tbody>
</table>

Figure 16  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 17 and Figure 18.
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 Timestamp (TIME) option 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].

```
+--------+--------+------------------+------------------+
| Kind=7 | Len=10 |      TSval       |      TSecr       |
+--------+--------+------------------+------------------+
    1 byte  1 byte       4 bytes            4 bytes
```

Figure 19  UDP TIME option format

TS Value (TSval) and TS Echo Reply (TSecr) are used in a similar manner to the TCP TS option [RFC7323]. On transmitted 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 (AE) option 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 20   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 21.

```
+--------+--------+------------------+
|  Kind  | Len=6  |      nonce       |
+--------+--------+------------------+
```

Figure 21   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].
Figure 22  UDP EXP option format

8. 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.

>> New options with fixed lengths smaller than 255 or variable
lengths that are always smaller than 255 MUST use only the default
option format.

Note that only certain of the initially defined options violate
these rules:
Internet-Draft        Transport Options for UDP         September 2019

- >> 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_OPT_OCS</td>
<td>Enable UDP OCS option</td>
</tr>
<tr>
<td>UDP_OPT_ACS</td>
<td>Enable UDP ACS option</td>
</tr>
<tr>
<td>UDP_OPT_LITE</td>
<td>Enable UDP LITE option</td>
</tr>
<tr>
<td>UDP_OPT_MSS</td>
<td>Enable UDP MSS option</td>
</tr>
<tr>
<td>UDP_OPT_TIME</td>
<td>Enable UDP TIME option</td>
</tr>
<tr>
<td>UDP_OPT_FRAG</td>
<td>Enable UDP FRAG option</td>
</tr>
<tr>
<td>UDP_OPT_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>
Abstract

HTTP/2 provides multiplexing of HTTP requests over a single underlying transport connection. HTTP/2 Transport defines a transport abstraction enabling delivery of byte stream and datagram data using streams of an HTTP/2 connection.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 11, 2019.

Copyright Notice

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

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

HTTP/2 [RFC7540] provides a framing layer that describes the exchange of HTTP messages. This framing layer includes multiplexing of multiple streams on a single underlying transport connection, flow control, stream dependencies and priorities, and exchange of configuration information between endpoints.

Section 8.3 of [RFC7540] defines the HTTP CONNECT method for HTTP/2, which converts a HTTP/2 stream into a tunnel for arbitrary data. [RFC8441] describes the use of the extended CONNECT method to negotiate the use of the WebSocket Protocol [RFC6455] on an HTTP/2 stream.

This document defines protocol names for use in the extended CONNECT handshake that allow negotiation of HTTP/2 streams that transport arbitrary byte streams or datagrams. It also extends the CONNECT handshake to allow both endpoints of an HTTP/2 connection to establish streams that tunnel data. Being able to transport arbitrary data on individual HTTP/2 streams allows an underlying connection to be shared by multiple protocols and allows all protocols to benefit from the features provided by HTTP/2 framing.

1.1. Notational Conventions

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

As described in Section 5.5 of [RFC7540], SETTINGS parameters allow endpoints to negotiate use of protocol extensions that would otherwise generate protocol errors. Use of the CONNECT method extension defined in [RFC6455] requires the SETTINGS_ENABLE_CONNECT_PROTOCOL parameter to be received by a client prior to its use.

This document introduces another SETTINGS parameter, SETTINGS_ENABLE_BIDIRECTIONAL_CONNECT, which MUST have a value of 0 or 1.

Once a SETTINGS_ENABLE_BIDIRECTIONAL_CONNECT parameter has been sent with a value of 1, an endpoint MUST NOT send the parameter with a value of 0.

Upon receipt of SETTINGS_ENABLE_BIDIRECTIONAL_CONNECT with a value of 1, an endpoint MAY use the extended CONNECT defined in [RFC6455] with the protocol values defined in this document. An endpoint that supports receiving the extended CONNECT method SHOULD send this setting with a value of 1.

Note that [RFC6455] restricts SETTINGS_ENABLE_CONNECT_PROTOCOL to have no effect if received by a server. This document modifies that restriction and allows both SETTINGS_ENABLE_CONNECT_PROTOCOL and SETTINGS_ENABLE_BIDIRECTIONAL_CONNECT to take effect if received by either endpoint of an HTTP/2 connection.

3. Negotiating Byte Stream and Datagram Tunnels

[RFC6455] defines the psuedo-header field :protocol which can indicate the protocol intended to be used on the tunnel established by the CONNECT method. Values for the :protocol psuedo-header field are maintained in an Upgrade Token Registry established by [RFC7230] for protocol-name tokens.

After receiving both SETTINGS_ENABLE_CONNECT_PROTOCOL and SETTINGS_ENABLE_BIDIRECTIONAL_CONNECT, either endpoint of an HTTP/2 connection can send a request in HEADERS frames to establish a byte stream or datagram tunnel via the extended CONNECT method. Similarly, either endpoint may be required to respond to an incoming CONNECT request seeking to establish such a tunnel.
3.1. Initiating the Extended CONNECT Handshake

Endpoints using this mechanism to establish byte stream or datagram tunnels over HTTP/2 streams follow the CONNECT handshake procedure defined in [RFC6455]. However, instead of supplying "websocket" for the :protocol pseudo-header field to indicate a WebSocket connection, they specify "bytestream" or "datagram" to indicate a byte stream or datagram connection, respectively.

The :scheme and :path psuedo-headers are required by [RFC6455]. The scheme of the target URI MUST be set to "https" for both byte stream and datagram tunnels. The path is used in the same manner as for the WebSocket protocol, and MAY be set to "/" (an empty path component) if not desired for use.

Implementations should note that the Origin, Sec-WebSocket-Version, Sec-WebSocket-Protocol, and Sec-WebSocket-Extensions header fields are not included in the CONNECT request and response header fields, since this handshake mechanism is not being used to negotiate a WebSocket connection.

If the response to the extended CONNECT request indicates success of the handshake, then all further data sent or received on the new HTTP/2 stream is considered byte stream or datagram data.

3.2. Responding to the Extended CONNECT Handshake

A recipient of the extended CONNECT method follows the same procedure outlined by [RFC8441].

If the recipient encounters a :protocol psuedo-header with an unknown value or a value corresponding to a protocol they do not support, or if the recipient encounters violations of the extended CONNECT handshake protocol, they MUST return an HTTP response with an appropriate error code, such as 400 Bad Request. Otherwise, unknown header fields are ignored.

Once the handshake has been validated and is considered successful, the responder sends a HTTP response with status 200. After that response, all further data sent or received on the new HTTP/2 stream is considered byte stream or datagram data.

4. Using Tunnels Established via the Extended CONNECT Handshake

DATA frames are used as usual on the stream established by the CONNECT handshake to transmit data.
If the application negotiated the "bytestream" protocol, then individual DATA frames represent segments of an in-order bytestream and are delivered to the application as a stream of bytes. Implementations can deliver data to the application as soon as it becomes available, since there are no message boundaries to preserve.

If the application negotiated the "datagram" protocol, individual DATA frames are considered complete messages on the stream. Implementations SHOULD preserve these message boundaries when delivering data to the application. This prevents applications from needing to insert another level of framing to delineate message boundaries while transmitting datagram messages over HTTP/2 streams. Additionally, if an application is forwarding messages received over a "datagram" stream, the contents of each DATA frame should be sent in individual datagrams where possible.

The same considerations around intermediaries as defined in Section 7 of [RFC6455] apply to the extended CONNECT method, a client that connects via HTTP/2 to an HTTP proxy should use a traditional CONNECT request to tunnel through that proxy to the destination server. It should then

Streams created via the extended CONNECT method participate in flow control, stream prioritization, and other HTTP/2 features in the same manner as request and response streams defined in [RFC7540]. Stream closure continues to be interpreted as defined in Section 5 of [RFC8441].

Note that the frame type restrictions defined in Section 8.3 of [RFC7540] remain in effect: only DATA, RST_STREAM, WINDOW_UPDATE, and PRIORITY frames are allowed on the connected streams and any other frame types MUST be treated as a stream error (Section 5.4.2 of [RFC7540]) if received.

4.1. Example

An example of negotiating a "bytestream" stream on an HTTP/2 connection follows. This example is intended to closely follow the example in Section 5.1 of [RFC8441] to help illustrate the minor differences defined in this document.
5. IANA Considerations

This specification registers two entries in the "HTTP Upgrade Tokens" registry that was established by [RFC7230].

A new token, "bytestream", for byte stream data.

- Value: bytestream

- Description: Arbitrary bidirectional byte stream data

- Expected Version Tokens:

- References: [[RFC Editor: Please fill in this value with the RFC number for this document.]]

A new token, "datagram" for datagram data.

- Value: datagram
6. Security Considerations

The tunnels established by the CONNECT handshake are expected to be protected with a TLS connection. They inherit the security properties of this cryptographic context.

The security considerations of [RFC8441] Section 8 and [RFC7540] Section 10, and Section 10.5.2 especially, still apply to this use of the CONNECT method.

7. Acknowledgments

Thanks to Anthony Chivetta, Joshua Otto, and Valentin Pistol for their contributions in the design and implementation of this work.

8. Normative References


[RFC8441] McManus, P., "Bootstrapping WebSockets with HTTP/2",
RFC 8441, DOI 10.17487/RFC8441, September 2018,

Authors’ Addresses

Eric Kinnear
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: ekinnear@apple.com

Tommy Pauly
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: tpauly@apple.com
LOOPS (Localized Optimizations on Path Segments) Problem Statement and Opportunities for Network-Assisted Performance Enhancement
draft-li-tsvwg-loops-problem-opportunities-03

Abstract

In various network deployments, end to end forwarding paths are partitioned into multiple segments. For example, in some cloud-based WAN communications, stitching multiple overlay tunnels are used for traffic policy enforcement matters such as to optimize traffic distribution or to select paths exposing a lower latency. Likewise, in satellite communications, the communication path is decomposed into two terrestrial segments and a satellite segment. Such long-haul paths are naturally composed of multiple network segments with various encapsulation schemes. Packet loss may show different characteristics on different segments.

Traditional transport protocols (e.g., TCP) respond to packet loss slowly especially in long-haul networks: they either wait for some signal from the receiver to indicate a loss and then retransmit from the sender or rely on sender’s timeout which is often quite long. Non-congestive loss may make the TCP sender over-reduce the sending rate unnecessarily. With the increase of end-to-end transport encryption (e.g., QUIC), traditional PEP (performance enhancing proxy) techniques such as TCP splitting are no longer applicable.

LOOPS (Local Optimizations on Path Segments) is a network-assisted performance enhancement over path segment and it aims to provide local in-network recovery to achieve better data delivery by making packet loss recovery faster and by avoiding the senders over-reducing their sending rate. In an overlay network scenario, LOOPS can be performed over a variety of the existing, or purposely created, tunnel-based path segments.
Status of This Memo

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

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

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

This Internet-Draft will expire on January 8, 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. The Problem ........................................ 3
   1.2. Sketching a Work Direction: Rationale & Goals ...... 4
2. Terminology ............................................. 6
3. Cloud-Internet Overlay Network .......................... 7
   3.1. Tail Loss or Loss in Short Flows .................... 9
   3.2. Packet Loss in Real Time Media Streams ............. 9
   3.3. Packet Loss and Congestion Control in Bulk Data Transfer 10
   3.4. Multipathing ....................................... 10
4. Satellite Communication .................................. 11
5. Branch Office WAN Connection ............................ 13
6. Features and Impacts to be Considered for LOOPS ....... 14
   6.1. Local Recovery and End-to-end Retransmission ....... 15
       6.1.1. OE to OE Measurement, Recovery, and Multipathing .. 17
Tunnels are widely deployed within many networks to achieve various engineering goals, including long-haul WAN interconnection or enterprise wireless access networks. A connection between two endpoints can be decomposed into many connection legs. As such, the corresponding forwarding path can be partitioned into multiple path segments that some of them are using network overlays by means of tunnels. This design serves a number of purposes such as steering the traffic, optimize egress/ingress link utilization, optimize traffic performance metrics (such as delay, delay variation, or loss), optimize resource utilization by invoking resource bonding, provide high-availability, etc.

A reliable transport layer normally employs some end-to-end retransmission mechanisms which also address congestion control [RFC0793] [RFC5681]. The sender either waits for the receiver to send some signals on a packet loss or sets some form of timeout for retransmission. For unreliable transport protocols such as RTP [RFC3550], optional and limited usage of end-to-end retransmission is employed to recover from packet loss [RFC4585] [RFC4588].

End-to-end retransmission to recover lost packets is slow especially when the network is long-haul. When a path is partitioned into multiple path segments that are realized typically as overlay tunnels, LOOPS (Local Optimizations on Path Segments) aims to provide local segment based in-network recovery to achieve better data delivery by making packet loss recovery faster and by avoiding the senders over-reducing their sending rate. In an overlay network scenario, LOOPS can be performed over the existing, or purposely created, overlay tunnel based path segments. Figure 1 show a basic usage scenario of LOOPS.

Some link types (satellite, microwave, drone-based networking, etc.) may exhibit unusually high loss rate in special conditions (e.g., fades due to heavy rain). The traditional TCP sender interprets loss as congestion and over-reduces the sending rate, degrading the
throughput. LOOPS is also applicable to such scenarios to improve
the throughput.

Also, multiple paths may be available in the network that may be used
for better performance. These paths are not visible to endpoints. Means
to make use of these paths while ensuring the overall
performance is enhanced would contribute to customer satisfaction.
Blindly implementing link aggregation may lead to undesired effects
(e.g., underperform compared to single path).

1.2. Sketching a Work Direction: Rationale & Goals

This document sketches a proposal that is meant to experimentally
investigate to what extent a network-assisted approach can contribute
to increase the overall perceived quality of experience in specific
situations (e.g., Sections 3.5 and 3.6 of [RFC8517]) without
requiring access to internal transport primitives. The rationale
beneath this approach is that some information (loss detection,
better visibility on available paths and their characteristics, etc.)
can be used to trigger local actions while avoiding as much as
possible undesired side effects (e.g., expose a behavior that would
be interpreted by an endpoint as an anomaly (corrupt data) and which
would lead to exacerbate end-to-end recovery. Such local actions
would have a faster effect (e.g., faster recovery, used multiple
paths simultaneously).

To that aim, the work is structured into two (2) phased stages:

- Stage 1: Network-assisted optimization. This one assumes that
  optimizations (e.g., support latency-sensitive applications) can
  be implemented at the network without requiring defining new
  interaction with the endpoint. Existing tools such as ECN will be
  used. Some of these optimizations may be valuable in deployments
  where communications are established over paths that are not
  exposing the same performance characteristics.

- Stage 2: Collaborative networking optimization. This one requires
  more interaction between the network and an endpoint to implement
  coordinated and more surgical network-assisted optimizations based
  on information/instructions shared by an endpoint or sharing
  locally-visible information with endpoint for better and faster
  recovery.

The document focuses on the first stage. Effort related to the
second stage is out of scope of the initial planned work.
Nevertheless, future work will be planned once progress is
(hopefully) made on the first stage.
The proposed mechanism is not meant to be applied to all traffic, but only to a subset which is eligible to the network-assisted optimization service.

Which traffic is eligible is deployment-specific and policy-based. For example, techniques for dynamic information of optimization function (e.g., SFC) may be leveraged to unambiguously identify the aggregate of traffic that is eligible to the service. Such identification may be triggered by subscription actions made by customers or be provided by a network provider (e.g., specific applications, during specific events such as during severe DDoS attack or flash crowds events).

Likewise, whether the optimization function is permanently instantiated or on-demand is deployment-specific.

This document does not intend to provide a comprehensive list of target deployment cases. Sample scenarios are described to illustrate some LOOPS potentials. Similar issues and optimizations may be helpful in other deployments such as enhancing the reliability of data transfer when a fleet of drones are used for specific missions (e.g., site inspection, live streaming, and emergency service). Captured data should be reliably transmitted via paths involving radio connections.

It is not required that all segments are LOOPS-aware to benefit from LOOPS advantages.

Section 3 presents some of the issues and opportunities found in Cloud-Internet overlay networks that require higher performance and more reliable packet transmission over best effort networks. Section 4 discusses applications of LOOPS in satellite communication. Section 6 describes the corresponding solution features and their impact on existing network technologies.
2. Terminology

This document makes use of the following terms:

LOOPS: Local Optimizations on Path Segments. LOOPS includes the local in-network (i.e., non end-to-end) recovery functions and other supporting features such as local measurement, loss detection, and congestion feedback.

LOOPS Node: A node supporting LOOPS functions.

Overlay Node (ON): A node having overlay functions (e.g., overlay protocol encapsulation/decapsulation, header modification, TLV inspection) and LOOPS functions in LOOPS overlay network usage scenario.

Overlay Tunnel: A tunnel with designated ingress and egress nodes using some network overlay protocol as encapsulation, optionally with a specific traffic type.

Overlay Edge (OE): Edge node of an overlay tunnel. It can behave as ingress or egress as a function of the traffic direction.

Path segment: A LOOPS enabled tunnel-based network subpath. It is used interchangeably with overlay segment in this document when the context wants to emphasize on its overlay encapsulated nature. It is also called segment for simplicity in this document.
Overlay segment: Refers to path segment.

Underlay Node (UN): A node not participating in the overlay network.

3. Cloud-Internet Overlay Network

CSPs (Cloud Service Providers) are connecting their data centers using the Internet or via self-constructed networks/links. This expands the traditional Internet’s infrastructure and, together with the original ISP’s infrastructure, forms the Internet underlay.

Automation techniques and NFV (Network Function Virtualization) further ambitions to make it easier to dynamically provision a new virtual node/function as a workload in a cloud for CPU/storage intensive functions. With the aid of various mechanisms such as kernel bypassing and Virtual IO, forwarding based on virtual nodes is becoming more and more effective. The interconnection among the purposely positioned virtual nodes and/or the existing nodes with virtualization functions potentially form an overlay infrastructure. It is called the Cloud-Internet Overlay Network (CION) in this document for short.

This architecture scenario makes use of overlay technologies to direct the traffic going through the specific overlay path regardless of the underlying physical topology, in order to achieve better service delivery. It purposely creates or selects overlay nodes (ON) from providers. By continuously measuring the delay of path segments and use them as metrics for path selection, when the number of overlay nodes is sufficiently large, there is a high chance that a better path could be found [DOI_10.1109_ICDCS.2016.49] [DOI_10.1145_3038912.3052560]. [DOI_10.1145_3038912.3052560] further shows all cloud providers experience random loss episodes and random loss accounts for more than 35% of total loss.

Some of the considerations that are discussed below may also apply for interconnecting DCs owned by a network provider.

Figure 2 shows an example of an overlay path over large geographic distances. Three path segments, i.e., ON1-ON2, ON2-ON3, ON3-ON4 are shown. ON is usually a virtual node, though it does not have to be. Each segment transmits packets using some form of network overlay protocol encapsulation. ON has the computing and memory resources that can be used for some functions like packet loss detection, network measurement and feedback, and packet recovery. ONs are managed by a single administrator though they can be workloads created from different CSPs.
Figure 2: Cloud-Internet Overlay Network (CION)

We tested based on 37 overlay nodes from multiple cloud providers globally. Each pair of the overlay nodes are used as sender and receiver. When the traffic is not intentionally directed to go through any intermediate virtual nodes, we call the path followed by the traffic in the test as the default path. When any of the virtual nodes is intentionally used as an intermediate node to forward the traffic, the path that the traffic takes is called an overlay path. The preliminary experiments showed that the delay of an overlay path is shorter than the one of the default path in 69% of cases at 99% percentile and improvement is 17.5% at 99% percentile when we probe Ping packets every second for a week. More experimental information can be found in [OCN].

Lower delay does not necessarily mean higher throughput. Different path segments may have different packet loss rates. Loss rate is another major factor impacting the overall TCP throughput. From some customer requirements, the target loss rate is set in the test to be less than 1% at 99% percentile and 99.9% percentile, respectively. The loss was measured between any two overlay nodes, i.e., any potential path segment. Two thousand Ping packets were sent every 20
Internet-Draft        LOOPS Problem & opportunities            July 2019

seconds between two overlay nodes for 55 hours. This preliminary
experiment showed that the packet loss rate satisfaction are 44.27%
and 29.51% at the 99% and 99.9% percentiles, respectively.

Hence packet loss in an overlay segment is a key issue to be solved
in such architecture. In long-haul networks, the end-to-end
retransmission of lost packet can result in an extra round trip time
(RTT). Such extra time is not acceptable in some latency-sensitive
applications. As CION naturally consists of multiple overlay
segments, LOOPS leverages this to perform local optimizations on a
single hop between two overlay nodes. (“Local” here is a concept
relative to end-to-end, it does not mean such optimization is limited
to LAN networks.)

The following subsections present different scenarios using multiple
segment-based overlay paths with a common need of local in-network
loss recovery in best effort networks.

3.1. Tail Loss or Loss in Short Flows

When the lost segments are at the end of a transaction, TCP’s fast
retransmit algorithm does not work as there are no ACKs to trigger
it. When a sender does not receive an ACK for a given segment within
a certain amount of time called retransmission timeout (RTO), it re-
sends the segment [RFC6298]. RTO can be as long as several seconds.
Hence the recovery of lost segments triggered by RTO is lengthy.
[I-D.dukkipati-tcpm-tcp-loss-probe] indicates that large RTOs make a
significant contribution to the long tail on the latency statistics
of short flows such as loading web pages.

The short flow often completes in one or two RTTs. Even when the
loss is not a tail loss, it can possibly add another RTT because of
end-to-end retransmission (not enough packets are in flight to
trigger fast retransmit). In long-haul networks, it can result in
extra time of tens or even hundreds of milliseconds.

An overlay segment transmits the aggregated flows from ON to ON. As
short-lived flows are aggregated, the probability of tail loss over
this specific overlay segment decreases compared to an individual
flow. The overlay segment is much shorter than the end-to-end path
in a Cloud-Internet overlay network, hence loss recovery over an
overlay segment is faster.

3.2. Packet Loss in Real Time Media Streams

The Real-time transport protocol (RTP) is widely used in interactive
audio and video. Packet loss degrades the quality of the received
media. When the latency tolerance of the application is sufficiently
large, the RTP sender may use RTCP NACK feedback from the receiver
[RFC4585] to trigger the retransmission of the lost packets before
the playout time is reached at the receiver.

In a Cloud-Internet overlay network, the end-to-end path can be
hundreds of milliseconds. End-to-end feedback based retransmission
may be not be very useful when applications can not tolerate one more
RTT of this length. Loss recovery over an overlay segment can then
be used for the scenarios where RTCP NACK triggered retransmission is
not appropriate.

3.3. Packet Loss and Congestion Control in Bulk Data Transfer

TCP congestion control algorithms such as Reno and CUBIC basically
interpret packet loss as congestion experienced somewhere in the
path. When a loss is detected, the congestion window will be
decreased at the sender to make the sending slower. It has been
observed that packet loss is not an accurate way to detect congestion
in the current Internet [I-D.cardwell-iccrg-bbr-congestion-control].
In long-haul links, when the loss is caused by non-persistent burst
which is extremely short and pretty random, the sender’s reaction of
reducing sending rate is not able to respond in time to the
instantaneous path situation or to mitigate such bursts. On the
contrary, reducing window size at the sender unnecessarily or too
aggressively harms the throughput for application’s long lasting
traffic like bulk data transfer.

The overlay nodes are distributed over the path with computing
capability, they are in a better position than the end hosts to
quickly deduce the underlying links’ instantaneous situation from
measuring the delay, loss or other metrics over the segment. Shorter
round trip time over a path segment will benefit more accurate and
immediate measurements for the maximum recent bandwidth available,
the minimum recent latency, or trend of change. ONs can further
decide if the sending rate reduction at the sender is necessary when
a loss happened. Section 6.2 talks more details on this.

3.4. Multipathing

As an overlay path may suffer from an impairment of the underlying
network, two or more overlay paths between the same set of ingress
and egress overlay nodes can be combined for reliability purpose.
During a transient time when a network impairment is detected,
sending replicating traffic over two paths can improve reliability.

When two or more disjoint overlay paths are available as shown in
Figure 3 from ON1 to ON2, different sets of traffic may use different
overlay paths. For instance, one path is for low latency and the
other is for higher bandwidth, or they can be simply used as load balancing for better bandwidth utilization.

Two disjoint paths can be, for example, found by measurement to figure out the segments with very low "mathematical correlation" in latency change. When the number of overlay nodes is large, it is easy to find disjoint or partially disjoint segments. This information may be available if the ONs are managed by the network provider managing the underlying forwarding paths.

Different overlay paths may have varying characteristics, obviously. The overlay tunnel should allow the overlay path to handle the packet loss depending on its own path measurements.

```
+----------o------------------+
|                             |
|                             |
A -----o ON1                      ON2o----- B
|                             |
+-----------------------o-----+

ON-B
```

Figure 3: Example of Multiple Overlay Paths

In reference to Figure 3, both A and B are not aware of the existence of these multiple paths. A network-assistance would be valuable for the sake of better resilience and performance. Note that in a collaborative context (a.k.a., stage 2 mentioned in Section 1.2) LOOPS may target means to advertise the available path characteristics to an endpoint A/B, to allow an endpoint A/B to control the traffic distribution policy to be enforced by ON1/ON2, or to let endpoint A/B notify ON1/ON2 with their multipathing preference.

4. Satellite Communication

Traditionally, satellite communications deploy PEP (performance enhancing proxy [RFC3135]) nodes around the satellite link to enhance end-to-end performance. TCP splitting is a common approach employed by such PEPs, where the TCP connection is split into three: the segment before the satellite hop, the satellite section (uplink, downlink), and the segment behind the satellite hop. This requires heavy interactions with the end-to-end transport protocols, usually without the explicit consent of the end hosts. Unfortunately, this is indistinguishable from a man-in-the-middle attack on TCP. With end-to-end encryption moving under the transport (QUIC), this approach is no longer useful.
Geosynchronous Earth Orbit (GEO) satellites have a one-way delay (up to the satellite and back) on the order of 250 milliseconds. This does not include queueing, coding and other delays in the satellite ground equipment. The Round Trip Time for a TCP or QUIC connection going over a satellite hop in both directions, in the best case, will be on the order of 600 milliseconds. And, it may be considerably longer. RTTs on this order of magnitude have significant performance implications.

Packet loss recovery is an area where splitting the TCP connection into different parts helps. Packets lost on the terrestrial links can be recovered at terrestrial latencies. Packet loss on the satellite link can be recovered more quickly by an optimized satellite protocol between the PEPs and/or link layer FEC than they could be end to end. Again, encryption makes TCP splitting no longer applicable. Enhanced error recovery at the satellite link layer helps for the loss on the satellite link but doesn’t help for the terrestrial links. Even when the terrestrial segments are short, any loss must be recovered across the satellite link delay. And, there are cases when a satellite ground station connects to the general Internet with a potentially larger terrestrial segment (e.g., to a correspondent host in another country). Faster recovery over such long terrestrial segments is desirable.

Another aspect of recovery is that terrestrial loss is highly likely to be congestion related but satellite loss is more likely to be transmission errors due to link conditions. A transport endpoint slowing down because of mis-interpreting these errors as congestion losses unnecessarily reduces performance. But, at the end points, the difference between the two is not easily distinguished. To elaborate more on the loss recovery for satellite communications, while the error rate on the satellite paths is generally very low most of the time, it might get higher during special link conditions (e.g., fades due to heavy rain). The satellite hop itself does know which losses are due to link conditions as opposed to congestion, but it has no mechanism to signal this difference to the end hosts.

We will need the protocol under QUIC to try to minimize non-congestion packet drop. Specific link layers may have techniques such as satellite FEC to recover. Where the capabilities of that may be exceeded (e.g., rain fade), we can look at LOOPS-like approaches.

There are two high level classes of solutions for making encrypted transport traffic like QUIC work well over satellite:

- Hooks in the transport protocol which can adapt to large BDPs where both the bandwidth and the latency are large. This would require end to end enhancement.
o Capabilities (such as LOOPS) under the transport protocol to improve performance over specific segments of the path. In particular, separating the terrestrial from the satellite losses. Fixing the terrestrial loss quickly and keeping throughput high over satellite segment by not causing the end-hosts to over-reduce their sending window in case of non-congestion loss.

This document focuses on the latter.

5. Branch Office WAN Connection

Enterprises usually require network connections between the branch offices or between branch offices and cloud data center over geographic distances. With the increasing deployment of vCPE (virtual CPE), some services usually hosted on the CPE are moved to the provider network from the customer site. Such vCPE approach enables some value added service to be provided such as WAN optimization and traffic steering.

Figure 4 shows an example of two branch offices WAN connection via Internet. Figure 5 shows a branch office access to public cloud via a selected PoP (point of presence). vCPE connects to that PoP which can be hundreds of kilometers away via Internet. In both cases, the path segments over Internet is subject to loss. Similar problems presented in subsections of Section 3 should be solved. The GW1 may be reachable via multiple paths.

Requirements to steer traffic through different sub-paths for latency optimization, resource optimization, balancing, or other purposes are increasing. For example, directing the traffic from vCPE to a lightly loaded PoP rather than to the closest one. Mere best effort transport is not sufficient. New technologies like SFC (Service Function Chaining), SRv6 (segment routing over IPv6), and NFV/SDN used together with vCPE to enable the potentials to embed more complicated loss recovery functions at intermediate nodes in end-to-end path.

```
+------+-+|GW1|+-+|vCPE1|-----------------|vCPE2|+-+|GW2|
+------+-+|-----|++-|------|++-|-----|++-|
 Site A               Site B
```

Figure 4: Branch Office WAN Connection via Internet
6. Features and Impacts to be Considered for LOOPS

This section provides an overview of the proposed LOOPS solution. This section is not meant to document a detailed specification, but it is meant to highlight some design choices that may be followed during the solution design phase.

LOOPS aims to improve the transport performance "locally" in addition to native end-to-end mechanism supported by a given transport protocol. This is possible because LOOPS nodes will be instantiated to partition the path into multiple segments. With the advent of automation and technologies like NFV and virtual IO, it is possible to dynamically instantiate functions to nodes. Some overlay protocols such as VXLAN [RFC7348], GENEVE [I-D.ietf-nvo3-geneve], LISP [RFC6830] or CAPWAP [RFC5415] may be used in the network. In overlay network usage scenario, LOOPS can extend a specific overlay
protocol header to perform local measurement and local recovery functions, like the example shown in Figure 6.

```
+------------+------------+-----------------+---------+---------+
|Outer IP hdr|Overlay hdr |LOOPS information|Inner hdr|payload  |
+------------+------------+-----------------+---------+---------+
```

**Figure 6: LOOPS Extension Header Example**

LOOPS should be designed to minimize its overhead while increasing the benefit (e.g., reduces the completion time of a video application, reduces the loss). Also, LOOPS should be designed to auto-tune itself in case its overhead is exceeding a threshold.

For example, LOOPS uses packet number space independent from that of the transport layer. Acknowledgment should be generated from ON receiver to ON sender for packet loss detection and local measurement. To reduce overhead, negative ACK over each path segment is a good choice here. A Timestamp echo mechanism, analogous to TCP’s Timestamp option, should be employed in-band in LOOPS extension to measure the local RTT and variation for an overlay segment. Local in-network recovery is performed. The measurement over segment is expected to give a hint on whether the lost packet of locally recovered one was caused by congestion. Such a hint could be further feedback, using like by ECN Congestion Experienced (CE) markings, to the end host sender. It directs the end host sender if congestion window adjustment is necessary. LOOPS normally works on the overlay segment which aggregates the same type of traffic, for instance TCP traffic or finer granularity like TCP throughput sensitive traffic. LOOPS does not look into the inner packet (when an encapsulation scheme is used). Elements to be considered in LOOPS are discussed briefly here.

6.1. Local Recovery and End-to-end Retransmission

There are basically two ways to perform local recovery, retransmission and FEC (Forward Error Correction). They are possibly used together in some cases. Such approaches between two overlay nodes recover the lost packet in relatively shorter distance and thus shorter latency. Therefore the local recovery is always faster compared to end-to-end.

At the same time, most transport layer protocols have their own end-to-end retransmission to recover the lost packet. It would be ideal if end-to-end retransmission at the sender was not triggered when the local recovery is successful.
End-to-end retransmission is normally triggered by a NACK as in RTCP or multiple duplicate ACKs as in TCP.

When FEC is used for local recovery, it may come with a buffer to make sure the recovered packets delivered are in order subsequently. Therefore the receiver side is unlikely to see the out-of-order packets and then send a NACK or multiple duplicate ACKs. The side effect to unnecessarily trigger end-to-end retransmit is minimum. When FEC is used, if redundancy and block size are determined, extra latency required to recover lost packets is also bounded. Then RTT variation caused by it is predictable. In some extreme case like a large number of packet loss caused by persistent burst, FEC may not be able to recover it. Then end-to-end retransmit will work as a last resort. In summary, when FEC is used as local recovery, the impact on end-to-end retransmission is limited.

When local retransmission is used, more care is required.

For packet loss in RTP streaming, local retransmission can recover those packets which would not be retransmitted end-to-end otherwise due to long RTT. It would be ideal if the retransmitted packet reaches the receiver before it sends back information that the sender would interpret as a NACK for the lost packet. Therefore when the segment(s) being retransmitted is a small portion of the whole end to end path, the retransmission will have a significant effect of improving the quality at receiver. When the sender also re-transmits the packet based on a NACK received, the receiver will receive the duplicated retransmitted packets and should ignore the duplication.

For packet loss in TCP flows, TCP RENO and CUBIC use duplicate ACKs as a loss signal to trigger the fast retransmit. There are different ways to avoid the sender’s end-to-end retransmission being triggered prematurely:

- The egress overlay node can buffer the out-of-order packets for a while, giving a limited time for a packet being retransmitted somewhere in the overlay path to reach it. The retransmitted packet and the buffered packets caused by it may increase the RTT variation at the sender. When the retransmitted latency is a small portion of RTT or the loss is rare, such RTT variation will be smoothed without much impact. Another possible way is to make the sender exclude such packets from the RTT measurement. The locally recovered packets can be specially marked and this marking is spin back to end host sender. Then RTT measurement should not use that packet.

The buffer management is nontrivial in this case. It has to be determined how many out-of-order packets can be buffered at the
egress overlay node before it gives up waiting for a successful local retransmission. In some extreme case the lost packet is not recovered successfully locally, the sender may invoke end-to-end fast retransmit slower than it would be in classic TCP.

- If LOOPS network does not buffer the out-of-order packets caused by packet loss, TCP sender can use a time based loss detection like RACK [I-D.ietf-tcpm-rack] to prevent the TCP sender from invoking fast retransmit too early. RACK uses the notion of time to replace the conventional DUPACK threshold approach to detect losses. RACK is required to be tuned to fit the local retransmission better. If there are n similar segments over the path, segment retransmission will at least add RTT/n to the reordering window by average when the packet is lost only once over the whole overlay path. This approach is more preferred than one described in previous bullet. On the other hand, if time based loss detection is not supported at the sender, end to end retransmission will be invoked as usual. It wastes some bandwidth.

6.1.1. OE to OE Measurement, Recovery, and Multipathing

When multiple segments are stitched, another type of local recovery can be performed between OE (Overlay Edge) to OE. When the segments of an overlay path have similar characteristics and/or only OE has the expected processing capability, OE to OE based local recovery can be used instead of per-segment based recovery.

If there is more than one overlay path between two OEs, multipathing can split and recombine the traffic. Measurements such as RTT and loss rate between OEs have to be specific to each path. The ingress OE can use the feedback measurement to determine the FEC parameter settings for different path. FEC can also be configured to work over the combined path. FEC should not increase redundancy over the path where a congestion is found. The egress OE should be able to remove the duplicated packets when multipathing is available.

OE to OE measurement can help each segment determine its proportion in edge to edge delay. It is useful for ON to decide if it is necessary to turn on the per segment recovery or how to fine tune the parameter settings. When the segment delay ratio is small, the segment retransmission is more effective. Such approach requires nested LOOPS function. This draft does not focus on the nest LOOPS now. More details will be discussed later if comments showing interests in it are received.
6.2. Congestion Control Interaction

When a TCP-like transport layer protocol is used, local recovery in LOOPS has to interact with the upper layer transport congestion control. Classic TCP adjusts the congestion window when a loss is detected and fast retransmit is invoked.

The local recovery mechanism breaks the assumption of the necessary and sufficient conditional relationship between detected packet loss and congestion control trigger at the sender in classic TCP. The loss that is locally recovered can be caused by a non-persistent congestion such as a random loss or a microburst, both of which ideally would not let the sender invoke the congestion control mechanism. But then, loss can also possibly caused by a real persistent congestion which should let the sender aware of it and reduces its sending rate.

When a local recovery takes effect, we consider the following two cases. Firstly, the classic TCP sender does not see enough number of duplicate ACKs to trigger fast retransmit. This may be due to the local recovery procedures, which hides the out-of-order packet from receiver using mechanisms like reordering buffer at egress node. Classic TCP sender in this case will not reduce congestion window as no loss is detected. Secondly, if a time based loss detection such as RACK is used, as long as the locally recovered packet’s ACK reaches the sender before the reordering window expires, the congestion window will not be reduced.

Such behavior brings the desirable throughput improvement when the recovered packet is lost due to non-persistent congestion. It solves the throughput problem mentioned in Section 3.3 and Section 4. However, it also brings the risk that the sender is not able to detect a real persistent congestion in time, and then overshooting may occur. Eventually a severe congestion that is not recoverable by a local recovery mechanism will be detected by sender. In addition, it may be unfriendly to other flows (possibly pushing them out) if those flows are running over the same underlying bottleneck links.

There is a spectrum of approaches. On one end, each locally recovered packet can be treated exactly as a loss in order to invoke the congestion control at the sender to guarantee the fair sharing as classic TCP by setting its CE (Congestion Experienced) bit. Explicit Congestion Notification (ECN) can be used here as ECN marking was required to be equivalent to a packet drop [RFC3168]. Congestion control at the sender works as usual and no throughput improvement could be achieved (although the benefit of faster recovery is still there). On the other hand, ON can perform its congestion measurement over the segment, for instance local RTT and its variation trend.
Such measurement can help to determine if a lost packet by congestion. It will further decide if it is necessary to set CE marking or even what ratio is set to make the sender adjust the sending rate.

There are possible cases that the sender detects the loss even with local recovery in function. For example, when the re-ordering window in RACK is not optimally adapted, the sender may trigger the congestion control at the same time of end-to-end retransmission. If spurious retransmission detection based on DSACK [RFC3708] is used, such end-to-end retransmission will be found out unnecessary when locally recovered packets reaches the receiver successfully. Then congestion control changes will be undone at the sender. This results in similar pros and cons as described earlier. Pros are preventing the unnecessary window reduction and improving the throughput when the loss is caused by non-congestive loss. Cons are some mechanisms like ECN or its variants should be used wisely to make sure the congestion control is invoked in case of persistent congestion.

An approach where the losses on a path segment are not immediately made known to the end-to-end congestion control can be combined with a "circuit breaker" style congestion control on the path segment. When the usage of path segment by the overlay flow starts to become unfair, the path segment sends congestion signals up to the end-to-end congestion control. This must be carefully tuned to avoid unwanted oscillation.

In summary, local recovery can improve Flow Completion Time (FCT) by eliminating tail loss in small flows. As it may change loss event to out-of-order event in most cases to TCP sender, if TCP sender uses loss based congestion control, there is no much throughput improvement. We suggest ECN and spurious retransmission to be enabled when local recovery is in use, it would give the desirable throughput performance, i.e. when loss is caused by congestion, reduce congestion window; otherwise keep sender’s sending rate. We do not suggest to use spurious retransmission alone together with local recovery as it may cause the TCP sender falsely undo window reduction when congestion occurs. If only ECN is enabled or neither ECN nor spurious retransmission is enabled, the throughput with local recovery in use is no much difference from that of the tradition TCP.

6.3. Overlay Protocol Extensions

The overlay usually has no control over how packets are routed in the underlying network between two overlay nodes, but it can control, for example, the sequence of overlay nodes a message traverses before reaching its destination. LOOPS assumes the overlay protocol can
deliver the packets in such designated sequence. Most forms of overlay networking use some sort of "encapsulation". The whole path taken can be performed by stitching multiple overlay paths, like VXLAN [RFC7348], GENEVE [I-D.ietf-nvo3-geneve], or it can be a single overlay path with a sequence of intermediate overlay nodes specified, as in SRv6 [I-D.ietf-6man-segment-routing-header]. In either way, LOOPS information is required to be embedded in some form to support the data plane measurement and feedback. Retransmission or FEC based loss recovery can be either per ON-hop or OE to OE based.

LOOPS alone has no setup requirement on control plane. Some overlay protocols, e.g., CAPWAP [RFC5415], has session setup phase, it can be used to exchange the information such as dynamic FEC parameters.

6.4. Summary

LOOPS is expected to extend the existing overlay protocols in data plane. Path selection is assumed a feature provided by the overlay protocols via SDN techniques [RFC7149] or other approaches and is not a part of LOOPS. LOOPS is a set of functions to be implemented on Overlay Nodes, that will be involved in forwarding packets in a long haul overlay network. LOOPS targets the following features.

1. Local recovery: Retransmission, FEC, or combination thereof can be used as local recovery method. Such recovery mechanism is in-network. It is performed by two network nodes with computing and memory resources.

2. Local congestion measurement: Ingress/Egress overlay nodes measure the local segment RTT, loss and/or throughput to immediately get the overlay segment status.

3. Signal to end-to-end congestion control: Strategy to set ECN CE marking or simply not to recover the packet to signal the end host sender about if and/or how to adjust the sending rate is required.

7. Security Considerations

LOOPS does not require access to the traffic payload in clear, so encrypted payload does not affect functionality of LOOPS.

The use of LOOPS introduces some issues which impact security. ON with LOOPS function represents a point in the network where the traffic can be potentially manipulated and intercepted by malicious nodes. Means to ensure that only legitimate nodes are involved should be considered.
Denial of service attack can be launched from an ON. A rogue ON might be able to spoof packets as if it come from a legitimate ON. It may also modify the ECN CE marking in packets to influence the sender's rate. In order to protected from such attacks, the overlay protocol itself should have some build-in security protection which inherently be used by LOOPS. The operator should use some authentication mechanism to make sure ONs are valid and non-compromised.

8. IANA Considerations

No IANA action is required.

9. Acknowledgements

Thanks to etosat mailing list about the discussion about the SatCom and LOOPS use case.

10. Informative References


[I-D.dukkipati-tcpm-tcp-loss-probe]

[I-D.ietf-nvo3-geneve]

[I-D.ietf-tcpm-rack]

[I-D.ietf-6man-segment-routing-header]

[I-D.cardwell-iccrg-bbr-congestion-control]

[DOI_10.1109_ICDCS.2016.49]

[DOI_10.1145_3038912.3052560]

[OCN]

Authors’ Addresses

Yizhou Li
Huawei Technologies
101 Software Avenue,
Nanjing 210012
China

Phone: +86-25-56624584
Email: liyizhou@huawei.com

Xingwang Zhou
Huawei Technologies
101 Software Avenue,
Nanjing 210012
China

Email: zhouxingwang@huawei.com

Mohamed Boucadair
Orange

Email: mohamed.boucadair@orange.com

Jianglong Wang
China Telecom

Email: wangjll.bri@chinatelecom.cn
The Some Congestion Experienced ECN Codepoint
draft-morton-taht-tsvwg-sce-00

Abstract

This memo reclassifies ECT(1) to be an early notification of congestion on ECT(0) marked packets, which can be used by AQM algorithms and transports as an earlier signal of congestion than CE. It is a simple, transparent, and backward compatible upgrade to existing IETF-approved AQMs, RFC3168, and nearly all congestion control algorithms.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 11, 2019.

Copyright Notice

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

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

Table of Contents

1. Terminology ................................. 2
2. Introduction ................................. 2
3. Background .................................. 2
4. Some Congestion Experienced ................. 3
5. Examples of use .............................. 5
   5.1. Cubic .................................. 5
   5.2. TCP receiver side handling ............... 5
   5.3. Other .................................. 5
6. Related Work ................................ 5
7. IANA Considerations ........................... 5
8. Security Considerations ....................... 6
9. Acknowledgements ............................. 6
10. References .................................. 6
  10.1. Normative References ...................... 6
  10.2. Informative References .................... 6
Authors’ Addresses .............................. 7

1. Terminology

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in [RFC2119].

2. Introduction

This memo reclassifies ECT(1) to be an early notification of congestion on ECT(0) marked packets, which can be used by AQM algorithms and transports as an earlier signal of congestion than CE ("Congestion Experienced").

This memo limits its scope to the redefinition of the ECT(1) codepoint as SCE, "Some Congestion Experienced", with a few brief illustrations of how it may be used.

3. Background

[RFC3168] defines the lower two bits of the (former) TOS byte in the IPv4/6 header as the ECN field. This may take four values: Not-ECT, ECT(0), ECT(1) or CE.

Binary Keyword References

------------------------------------------------------------
Research has shown that the ECT(1) codepoint goes essentially unused, with the "Nonce Sum" extension to ECN having not been implemented in practice and thus subsequently obsoleted by [RFC8311] (section 3). Additionally, known [RFC3168] compliant senders do not emit ECT(1), and compliant middleboxes do not alter the field to ECT(1), while compliant receivers all interpret ECT(1) identically to ECT(0). These are useful properties which represent an opportunity for improvement.

Experience gained with 7 years of [RFC8290] deployment in the field suggests that it remains difficult to maintain the desired 100% link utilisation, whilst simultaneously strictly minimising induced delay due to excess queue depth - irrespective of whether ECN is in use. This leads to a reluctance amongst hardware vendors to implement the most effective AQM schemes because their headline benchmarks are throughput-based.

The underlying cause is the very sharp "multiplicative decrease" reaction required of transport protocols to congestion signalling (whether that be packet loss or CE marks), which tends to leave the congestion window significantly smaller than the ideal BDP when triggered at only slightly above the ideal value. The availability of this sharp response is required to assure network stability (AIMD principle), but there is presently no standardised and backwards-compatible means of providing a less drastic signal.

4. Some Congestion Experienced

As consensus has arisen that some form of ECN signaling should be an earlier signal than drop, this Internet Draft changes the meaning of ECT(1) to be SCE, meaning "Some Congestion Experienced". The above ECN-field codepoint table then becomes:

Binary Keyword References

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Not-ECT (Not ECN-Capable Transport) [RFC 3168]</td>
</tr>
<tr>
<td>01</td>
<td>SCE (Some Congestion Experienced) [This Internet-draft]</td>
</tr>
<tr>
<td>10</td>
<td>ECT (ECN-Capable Transport) [RFC 3168]</td>
</tr>
<tr>
<td>11</td>
<td>CE (Congestion Experienced) [RFC 3168]</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, and both existing and SCE-aware middleboxes MAY convert SCE to CE in the same circumstances as for ECT, thus ensuring backwards compatibility with [RFC3168] ECN endpoints.

Permitted ECN codepoint packet transitions by middleboxes are:

Not-ECT -> Not-ECT or DROP  
ECT     -> ECT or SCE or CE or DROP  
SCE     -> SCE or CE or DROP  
CE      -> CE or DROP

In other words, for ECN-aware flows, the ECN marking of an individual packet MAY be increased by a middlebox to signal congestion, but MUST NOT be decreased, and packets SHALL NOT be altered to appear to be ECN-aware if they were not originally, nor vice versa. Note however that SCE is numerically less than ECT, but semantically greater, and the latter definition applies for this rule.

New SCE-aware receivers and transport protocols SHALL continue to apply the [RFC3168] interpretation of the CE codepoint, that is, to signal the sender to back off send rate to the same extent as if a packet loss were detected. This maintains compatibility with existing middleboxes, senders and receivers.

New SCE-aware receivers and transport protocols SHOULD interpret the SCE codepoint as an indication of mild congestion, and respond accordingly by applying send rates intermediate between those resulting from a continuous sequence of ECT codepoints, and those resulting from a CE codepoint. The ratio of ECT and SCE codepoints received indicates the relative severity of such congestion, such that 100% SCE is very close to the threshold of CE marking, 100% ECT indicates that the bottleneck link may not be fully utilised, and a 1:1 balance of ECT and SCE codepoints indicates that the present send rate is a good match to the bottleneck link.

Details of how to implement SCE awareness at the transport layer will be left to additional Internet Drafts yet to be submitted.

To maximise the benefit of SCE, middleboxes SHOULD produce SCE markings sooner than they produce CE markings, when the level of congestion increases.
5. Examples of use

5.1. Cubic

Consider a TCP transport implementing CUBIC congestion control. This presently exhibits exponential cwnd growth during slow-start, polynomial cwnd growth in steady-state, and multiplicative decrease upon detecting a single CE marking or packet loss in one RTT cycle.

With SCE awareness, it might exit slow-start upon detecting a single SCE marking, switch from polynomial to Reno-linear cwnd growth when the SCE:ECT ratio exceeds 1:2, halt cwnd growth entirely when it exceeds 1:1, and implement a Reno-linear decline when it exceeds 2:1, in addition to retaining the sharp 40% decrease on detecting CE.

In ideal circumstances, the above behaviour would result in the send rate stabilising at a level which produces between 50% and 66% SCE marking at some bottleneck on the path. The middlebox performing this marking can thus control the send rate smoothly to an ideal value, maximising throughput with minimum average queue length.

5.2. TCP receiver side handling

SCE can potentially be handled entirely by the receiver and be entirely independent of any of the dozens of [RFC3168] compliant congestion control algorithms, for example by manipulating the TCP receive window in a similar manner to the sender’s congestion window.

Alternatively, some mechanism may be defined to feed back SCE signals to the sender explicitly. Details of this are left to future I-Ds.

5.3. Other

New transports under development such as QUIC SHOULD implement a multi-bit, sub-RTT, and finer grained signal back to the sender based on SCE.

6. Related Work

[RFC8087] [RFC7567] [RFC7928] [RFC8290] [RFC8289] [RFC8033] [RFC8034]

7. IANA Considerations

There are no IANA considerations.
8. Security Considerations

There are no security considerations.

9. Acknowledgements

Many thanks to John Gilmore, the members of the ecn-sane project and the cake@lists.bufferbloat.net mailing list, and the former IETF AQM working group.

10. References

10.1. Normative References


10.2. Informative References


Morton & Taeht Expires September 11, 2019 [Page 6]


Authors’ Addresses

Jonathan Morton
Bufferbloat.net
Koekkoekranta 21
PITKAeJAEaRVI 31520
FINLAND
Phone: +358 44 927 2377
Email: chromatix99@gmail.com

David M. Taeht
TekLibre
20600 Aldercroft Heights Rd
Los Gatos, Ca  95033
USA
Phone: +18312059740
Email: dave@taht.net
SOCKS Protocol Version 6
draft-olteanu-intarea-socks-6-07

Abstract

The SOCKS protocol is used primarily to proxy TCP connections to arbitrary destinations via the use of a proxy server. Under the latest version of the protocol (version 5), it takes 2 RTTs (or 3, if authentication is used) before data can flow between the client and the server.

This memo proposes SOCKS version 6, which reduces the number of RTTs used, takes full advantage of TCP Fast Open, and adds support for 0-RTT authentication.

Status of This Memo

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

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

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

This Internet-Draft will expire on 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
Internet-Draft          SOCKS 6                       July 2019

to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1.  Introduction  .....................................  3
   1.1.  Revision log  .................................  4
2.  Requirements language  ................................  9
3.  Mode of operation  ..................................  9
4.  Requests  ......................................... 11
5.  Version Mismatch Replies  ............................ 13
6.  Authentication Replies  ............................... 13
7.  Operation Replies  .................................. 14
   7.1.  Handling CONNECT  .................................. 16
   7.2.  Handling BIND  .................................... 16
   7.3.  Handling UDP ASSOCIATE  ......................... 16
      7.3.1.  Proxying UDP servers  ......................... 18
      7.3.2.  Proxying multicast traffic .................... 18
8.  SOCKS Options  .................................... 18
   8.1.  Stack options  .................................. 19
      8.1.1.  IP TOS options  .............................. 20
      8.1.2.  TFO options  .................................. 21
      8.1.3.  Multipath options  ............................ 22
      8.1.4.  Listen Backlog options  ....................... 22
   8.2.  Authentication Method options  .................... 23
   8.3.  Authentication Data options  ...................... 25
   8.4.  Session options  ................................ 25
      8.4.1.  Session initiation  ........................... 26
      8.4.2.  Further SOCKS Requests  ....................... 27
      8.4.3.  Tearing down the session  ..................... 28
   8.5.  Idempotence options  ............................. 28
      8.5.1.  Requesting a token window  .................... 29
      8.5.2.  Spending a token  ............................. 30
      8.5.3.  Shifting windows  ............................. 31
      8.5.4.  Out-of-order Window Advertisements  .......... 31
   8.6.  Address Resolution Options  ...................... 31
      8.6.1.  Forward resolution  .......................... 31
      8.6.2.  Reverse resolution  ........................... 32
9.  Username/Password Authentication  ....................... 33
10. TCP Fast Open on the Client-Proxy Leg  .................. 34
11. False Starts  ...................................... 34
12. Security Considerations  ................................ 35
   12.1.  Large requests  ................................ 35
   12.2.  Replay attacks  ................................ 35
   12.3.  Resource exhaustion  ............................ 35
13. IANA Considerations  ................................ 36
1. Introduction

Versions 4 and 5 [RFC1928] of the SOCKS protocol were developed two decades ago and are in widespread use for circuit level gateways or as circumvention tools, and enjoy wide support and usage from various software, such as web browsers, SSH clients, and proxifiers. However, their design needs an update in order to take advantage of the new features of transport protocols, such as TCP Fast Open [RFC7413], or to better assist newer transport protocols, such as MPTCP [RFC6824].

One of the main issues faced by SOCKS version 5 is that, when taking into account the TCP handshake, method negotiation, authentication, connection request and grant, it may take up to 5 RTTs for a data exchange to take place at the application layer. This is especially costly in networks with a large delay at the access layer, such as 3G, 4G, or satelite.

The desire to reduce the number of RTTs manifests itself in the design of newer security protocols. TLS version 1.3 [RFC8446] defines a zero round trip (0-RTT) handshake mode for connections if the client and server had previously communicated.

TCP Fast Open [RFC7413] is a TCP option that allows TCP to send data in the SYN and receive a response in the first ACK, and aims at obtaining a data response in one RTT. The SOCKS protocol needs to concern itself with at least two TFO deployment scenarios: First, when TFO is available end-to-end (at the client, at the proxy, and at the server); second, when TFO is active between the client and the proxy, but not at the server.

This document describes the SOCKS protocol version 6. The key improvements over SOCKS version 5 are:

- The client sends as much information upfront as possible, and does not wait for the authentication process to conclude before requesting the creation of a socket.

- The connection request also mimics the semantics of TCP Fast Open [RFC7413]. As part of the connection request, the client can supply the potential payload for the initial SYN that is sent out to the server.
The protocol can be extended via options without breaking backward-compatibility.

The protocol can leverage the aforementioned options to support 0-RTT authentication schemes.

1.1. Revision log

Typos and minor clarifications are not listed.

draft-07

- All fields are now aligned.
- Eliminated version minors
- Lots of changes to options
  - 2-byte option kinds
  - Flattened option kinds/types/reply codes; also renamed some options
  - Socket options
    - Proxies MUST always answer them (Clients can probe for support)
    - MPTCP Options: expanded functionality ("please do/don’t do MPTCP on my behalf")
    - MPTCP Scheduler options removed
    - Listen Backlog options: code changed to 0x03
  - Revamped Idempotence options
  - Auth data options limited to one per method
- Authentication Reply: all authentication-related information is now in the options
  - Authentication replies no longer have a field indicating the chosen auth. method
  - Method that must proceed (or whereby authentication succeeded) indicated in options
* Username/password authentication: proxy now sends reply in option

  o Removed requirements w.r.t. caching authentication methods by multihomed clients

  o UDP: 8-byte association IDs

  o Sessions

    * The proxy is now free to terminate ongoing connections along with the session.

    * The session-terminating request is not part of the session that it terminated.

  o Address Resolution options

draft-06

  o Session options

  o Options now have a 2-byte length field.

  o Stack options

    * Stack options can no longer contain duplicate information.

    * TFO: Better payload size semantics

    * TOS: Added missing code field.

    * MPTCP Scheduler options:

      + Removed support for round-robin

      + "Default" renamed to "Lowest latency first"

    * Listen Backlog options: now tied to sessions, instead of an authenticated user

  o Idempotence options

    * Now used in the context of a session (no longer tied to an authenticated user)

    * Idempotence options have a different codepoint: 0x05. (Was 0x04.)
* Clarified that implementations that support Idempotence Options must support all Idempotence Option Types.

* Shifted Idempotence Option Types by 1. (Makes implementation easier.)

  o Shrunk vendor-specific option range to 32 (down from 64).

  o Removed reference to dropping initial data. (It could no longer be done as of -05.)

  o Initial data size capped at 16KB.

  o Application data is never encrypted by SOCKS 6. (It can still be encrypted by the TLS layer under SOCKS.)

  o Messages now carry the total length of the options, rather than the number of options. Limited options length to 16KB.

  o Security Considerations

    * Updated the section to reflect the smaller maximum message size.

    * Added a subsection on resource exhaustion.

draft-05

  o Limited the "slow" authentication negotiations to one (and Authentication Replies to 2)

  o Revamped the handling of the first bytes in the application data stream

    * False starts are now recommended. (Added the "False Start" section.)

    * Initial data is only available to clients willing to do "slow" authentication. Moved the "Initial data size" field from Requests to Authentication Method options.

    * Initial data size capped at 2^13. Initial data can no longer be dropped by the proxy.

    * The TFO option can hint at the desired SYN payload size.

  o Request: clarified the meaning of the Address and Port fields.
- Better reverse TCP proxy support: optional listen backlog for TCP BIND
- TFO options can no longer be placed inside Operation Replies.
- IP TOS stack option
- Suggested a range for vendor-specific options.
- Revamped UDP functionality
  - Now using fixed UDP ports
  - DTLS support
- Stack options: renamed Proxy-Server leg to Proxy-Remote leg
draft-04
- Moved Token Expenditure Replies to the Authentication Reply.
draft-03
- Shifted some fields in the Operation Reply to make it easier to parse.
- Added connection attempt timeout response code to Operation Replies.
- Proxies send an additional Authentication Reply after the authentication phase. (Useful for token window advertisements.)
- Renamed the section "Connection Requests" to "Requests"
- Clarified the fact that proxies don’t need to support any command in particular.
- Added the section "TCP Fast Open on the Client-Proxy Leg"
- Options:
  - Added constants for option kinds
  - Salt options removed, along with the relevant section from Security Considerations. (TLS 1.3 Makes AEAD mandatory.)
* Limited Authentication Data options to one per method.

* Relaxed proxy requirements with regard to handling multiple Authentication Data options. (When the client violates the above bullet point.)

* Removed interdependence between Authentication Method and Authentication Data options.

* Clients SHOULD omit advertising the "No authentication required" option. (Was MAY.)

* Idempotence options:
  + Token Window Advertisements are now part of successful Authentication Replies (so that the proxy-server RTT has no impact on their timeliness).
  + Proxies can’t advertise token windows of size 0.
  + Tweaked token expenditure response codes.
  + Support no longer mandatory on the proxy side.

* Revamped Socket options
  + Renamed Socket options to Stack options.
  + Banned contradictory socket options.
  + Added socket level for generic IP. Removed the "socket" socket level.
  + Stack options no longer use option codes from setsockopt().
  + Changed MPTCP Scheduler constants.

draft-02
  o Made support for Idempotence options mandatory for proxies.
  o Clarified what happens when proxies can not or will not issue tokens.
  o Limited token windows to 2^31 - 1.
  o Fixed definition of "less than" for tokens.
NOOP commands now trigger Operation Replies.

Renamed Authentication options to Authentication Data options.

Authentication Data options are no longer mandatory.

Authentication methods are now advertised via options.

Shifted some Request fields.

Option range for vendor-specific options.

Socket options.

Password authentication.

Salt options.

Added this section.

Support for idempotent commands.

Removed version numbers from operation replies.

Request port number for SOCKS over TLS. Deprecate encryption/encapsulation within SOCKS.

Added Version Mismatch Replies.

Renamed the AUTH command to NOOP.

Shifted some fields to make requests and operation replies easier to parse.

2. Requirements language

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

3. Mode of operation
Figure 1: The SOCKS version 6 protocol message exchange

When a TCP-based client wishes to establish a connection to a server, it must open a TCP connection to the appropriate SOCKS port on the SOCKS proxy. The client then enters a negotiation phase, by sending the request in Figure 1, that contains, in addition to fields present in SOCKS 5 [RFC1928], fields that facilitate low RTT usage and faster authentication negotiation.

Next, the server sends an authentication reply. If the request did not contain the necessary authentication information, the proxy indicates an authentication method that must proceed. This may trigger a longer authentication sequence that could include tokens.
for ulterior faster authentications. The part labeled "Authentication protocol" is specific to the authentication method employed and is not expected to be employed for every connection between a client and its proxy server. The authentication protocol typically takes up 1 RTT or more.

If the authentication is successful, an operation reply is generated by the proxy. It indicates whether the proxy was successful in creating the requested socket or not.

In the fast case, when authentication is properly set up, the proxy attempts to create the socket immediately after the receipt of the request, thus achieving an operational connection in one RTT (provided TFO functionality is available at the client, proxy, and server).

4. Requests

The client starts by sending a request to the proxy.

```
+---------------+---------------+---------------+---------------+
|  Version = 6  | Command Code  |        Options Length         |
+---------------+---------------+---------------+---------------+
|             Port              |  Padding = 0  | Address Type  |
+-------------------------------+---------------+---------------+
|                                                             ...
|                                                             ...
|                                                             ...
|                                                             ...
|                                                             ...
|                                                             ...
|                                                             ...
|                                                             ...
+---------------------------------------------------------------+

Figure 2: SOCKS 6 Request
```

- **Version**: 6
- **Command Code**:
  - *0x00 NOOP*: does nothing.
  - *0x01 CONNECT*: requests the establishment of a TCP connection. TFO MUST NOT be used unless explicitly requested.
  - *0x02 BIND*: requests the establishment of a TCP port binding.
* 0x03 UDP ASSOCIATE: requests a UDP port association.

- Address Type:
  - 0x01: IPv4
  - 0x03: Domain Name
  - 0x04: IPv6

- Address: this field’s format depends on the address type:
  - IPv4: a 4-byte IPv4 address
  - Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated, but padded by NUL characters, if needed.
  - IPv6: a 16-byte IPv6 address

- Port: the port in network byte order.

- Padding: set to 0

- Options Length: the total size of the SOCKS options that appear in the Options field. MUST NOT exceed 16KB.

- Options: see Section 8.

The Address and Port fields have different meanings based on the Command Code:

- NOOP: The fields have no meaning. The Address Type field MUST be either 0x01 (IPv4) or 0x04 (IPv6). The Address and Port fields MUST be 0.

- CONNECT: The fields signify the address and port to which the client wishes to connect.

- BIND, UDP ASSOCIATE: The fields indicate the desired bind address and port. If the client does not require a certain address, it can set the Address Type field to 0x01 (IPv4) or 0x04 (IPv6), and the Address field to 0. Likewise, if the client does not require a certain port, it can set the Port field to 0.

Clients can advertise their supported authentication methods by including an Authentication Method Advertisement option (see Section 8.2).
5. Version Mismatch Replies

Upon receipt of a request starting with a version number other than 6, the proxy sends the following response:

```
0 1 2 3 4 5 6 7
+---------------+
|  Version = 6  |
+---------------+
```

![Figure 3: SOCKS 6 Version Mismatch Reply](image)

- Version: 6
- A client MUST close the connection after receiving such a reply.

6. Authentication Replies

Upon receipt of a valid request, the proxy sends an Authentication Reply:

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+-------------------------------+
|  Version = 6  |     Type      |        Options Length         |
+---------------+---------------+-------------------------------+
|                                                             ...
...                 Options (variable length)                 ...
...                                                             |
+---------------------------------------------------------------+
```

![Figure 4: SOCKS 6 Authentication Reply](image)

- Version: 6
- Type:
  * 0x00: authentication successful.
  * 0x01: authentication failed.
- Options Length: the total size of the SOCKS options that appear in the Options field. MUST NOT exceed 16KB.
- Options: see Section 8.
If the server signals that the authentication has failed and does not signal that any authentication negotiation can continue (via an Authentication Method Selection option), the client MUST close the connection.

The client and proxy begin a method-specific negotiation. During such negotiations, the proxy MAY supply information that allows the client to authenticate a future request using an Authentication Data option. Application data is not subject to any encryption negotiated during this phase. Descriptions of such negotiations are beyond the scope of this memo.

When the negotiation is complete (either successfully or unsuccessfully), the proxy sends a second Authentication Reply. The second Authentication Reply MUST NOT allow for further negotiations.

7. Operation Replies

After the authentication negotiations are complete, the proxy sends an Operation Reply:

```
+---------------+---------------+---------------+---------------+
|  Version = 6  |  Reply Code   |        Options Length         |
| +---------------+---------------+---------------+---------------+
|           Bind Port           |  Padding = 0  | Address Type |
+-------------------------------+---------------+---------------+
     |                                                             ...
     |                 Bind Address (variable length)               ...
     |                                                             ...
     +---------------------------------------------------------------+
     |                                                             ...
     |                 Options (variable length)                     ...
     |                                                             ...
     +---------------------------------------------------------------+
```

Figure 5: SOCKS 6 Operation Reply

- Version: 6
- Reply Code:
  * 0x00: Success
  * 0x01: General SOCKS server failure
* 0x02: Connection not allowed by ruleset
* 0x03: Network unreachable
* 0x04: Host unreachable
* 0x05: Connection refused
* 0x06: TTL expired
* 0x07: Command not supported
* 0x08: Address type not supported
* 0x09: Connection attempt timed out

  o Bind Port: the proxy bound port in network byte order.
  o Padding: set to 0
  o Address Type:
    * 0x01: IPv4
    * 0x03: Domain Name
    * 0x04: IPv6
  o Bind Address: the proxy bound address in the following format:
    * IPv4: a 4-byte IPv4 address
    * Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated, but padded by NUL characters, if needed.
    * IPv6: a 16-byte IPv6 address
  o Options Length: the total size of the SOCKS options that appear in the Options field. MUST NOT exceed 16KB.
  o Options: see Section 8.

Proxy implementations MAY support any subset of the client commands listed in Section 4.

If the proxy returns a reply code other than "Success", the client MUST close the connection.
If the client issued an NOOP command, the client MUST close the connection after receiving the Operation Reply.

7.1. Handling CONNECT

In case the client has issued a CONNECT request, data can now pass.

7.2. Handling BIND

In case the client has issued a BIND request, it must wait for a second Operation reply from the proxy, which signifies that a host has connected to the bound port. The Bind Address and Bind Port fields contain the address and port of the connecting host. Afterwards, application data may pass.

7.3. Handling UDP ASSOCIATE

Proxies offering UDP functionality must be configured with a UDP port used for relaying UDP datagrams to and from the client, and/or a port used for relaying datagrams over DTLS.

Following a successful Operation Reply, the proxy sends a UDP Association Initialization message:

```
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
+---------------------------------------------------------------+
|                        Association ID                         |
|                           (8 bytes)                           |
+---------------------------------------------------------------+
```

Figure 6: UDP Association Initialization

- Association ID: the identifier of the UDP association

Proxy implementations SHOULD generate Association IDs randomly or pseudo-randomly.

Clients may start sending UDP datagrams to the proxy either in plaintext, or over an established DTLS session, using the proxy’s configured UDP ports. A client’s datagrams are prefixed by a SOCKS Datagram Header, indicating the remote host’s address and port:
Figure 7: SOCKS 6 Datagram Header

- **Version**: 0x06
- **Association ID**: the identifier of the UDP association
- **Address Type**:
  - 0x01: IPv4
  - 0x03: Domain Name
  - 0x04: IPv6
- **Address**: this field’s format depends on the address type:
  - IPv4: a 4-byte IPv4 address
  - Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated.
  - IPv6: a 16-byte IPv6 address
- **Port**: the port in network byte order.

Following the receipt of the first datagram from the client, the proxy makes a one-way mapping between the Association ID and:

- the 5-tuple of the UDP conversation, if the datagram was received over plain UDP, or
- the DTLS connection, if the datagram was received over DTLS. The DTLS connection is identified either by its 5-tuple, or some other mechanism, like [I-D.ietf-tls-dtls-connection-id].
Further datagrams carrying the same Association ID, but not matching the established mapping, are silently dropped.

The proxy then sends an UDP Association Confirmation message over the TCP connection with the client:

```
0 1 2 3 4 5 6 7
+---------------+
| Status = 0    |
+---------------+
```

Figure 8: UDP Association Confirmation

- Status: MUST be 0x00

Following the confirmation message, UDP packets bound for the proxy’s bind address and port are relayed to the client, also prefixed by a Datagram Header.

The UDP association remains active for as long as the TCP connection between the client and the proxy is kept open.

7.3.1. Proxying UDP servers

Under some circumstances (e.g. when hosting a server), the SOCKS client expects the remote host to send UDP datagrams first. As such, the SOCKS client must trigger a UDP Association Confirmation without having the proxy relay any datagrams on its behalf.

To that end, it sends an empty datagram prefixed by a Datagram Header with an IP address and port consisting of zeroes. The client SHOULD resend the empty datagram if an UDP Association Confirmation is not received after a timeout.

7.3.2. Proxying multicast traffic

The use of multicast addesseses is permitted for UDP traffic only.

8. SOCKS Options

SOCKS options have the following format:
8.1. Stack options

Stack options can be used by clients to alter the behavior of the protocols on top of which SOCKS is running, as well as the protocols used by the proxy to communicate with the remote host (i.e., IP, TCP, UDP). A Stack option can affect either the proxy’s protocol on the client-proxy leg or on the proxy-remote leg. Clients can only place Stack options inside SOCKS Requests.

Proxies MAY choose not to honor any Stack options sent by the client.

Proxies include Stack options in their Operation Replies to signal their behavior, and MUST do so for every supported Stack option sent by the client. Said options MAY also be unsolicited, i.e., the proxy MAY send them to signal behavior that was not explicitly requested by the client.

If a particular Stack option is unsupported, the proxy MUST silently ignore it.

In case of UDP ASSOCIATE, the stack options refer to the UDP traffic relayed by the proxy.
Stack options that are part of the same message MUST NOT contradict one another or contain duplicate information.

Figure 10: Stack Option

- Leg:
  - 1: Client-Proxy Leg
  - 2: Proxy-Remote Leg
  - 3: Both Legs

- Level:
  - 1: IP: options that apply to either IPv4 or IPv6
  - 2: IPv4
  - 3: IPv6
  - 4: TCP
  - 5: UDP

- Code: Option code

- Option Data: Option-specific data

8.1.1. IP TOS options
The client can use IP TOS options to request that the proxy use a certain value for the IP TOS field. Likewise, the proxy can use IP TOS options to advertise the TOS values being used.

8.1.2. TFO options

- **Leg**: 0x02 (Proxy-Remote leg).
- **Payload Size**: The desired payload size of the TFO SYN. Ignored in case of a BIND command.

If a SOCKS Request contains a TFO option, the proxy SHOULD attempt to use TFO in case of a CONNECT command, or accept TFO in case of a BIND command. Otherwise, the proxy MUST NOT attempt to use TFO in case of a CONNECT command, or accept TFO in case of a BIND command.

In case of a CONNECT command, the client can indicate the desired payload size of the SYN. If the field is 0, the proxy can use an arbitrary payload size. If the field is non-zero, the proxy MUST NOT use a payload size larger than the one indicated. The proxy MAY use a smaller payload size than the one indicated.
8.1.3. Multipath options

In case of a CONNECT or BIND command, the client can inform the proxy whether MPTCP is desired on the proxy-remote leg by sending a Multipath option.

Conversely, the proxy can use a Multipath option to convey the following information: * whether or not the connection uses MPTCP or not, when replying to a CONNECT command, or in the second Operation reply to a BIND command, or * whether an MPTCP connection will be accepted, when first replying to a BIND command.

```
+-------------------------------+-------------------------------+
|           Kind = 1            |          Length = 8           |
+---+-----------+---------------+---------------+---------------+
|Leg| Level = 4 |   Code = 2    | Availability  |  Padding = 0  |
+---+-----------+---------------+---------------+---------------+
```

Figure 13: Multipath Option

- Leg: 0x02 (Proxy-Remote leg)
- Availability:
  - 0x01: MPTCP is not desired or available
  - 0x02: MPTCP is desired or available

In the absence of such an option, the proxy SHOULD NOT enable MPTCP.

8.1.4. Listen Backlog options

```
+-------------------------------+-------------------------------+
|           Kind = 1            |          Length = 8           |
+---+-----------+---------------+-------------------------------+
|Leg| Level = 4 |   Code = 3    |            Backlog            |
+---+-----------+---------------+-------------------------------+
```

Figure 14: Listen Backlog Option

- Leg: 0x02 (Proxy-Remote leg)
o Backlog: The length of the listen backlog.

The default behavior of the BIND does not allow a client to simultaneously handle multiple connections to the same bind address. A client can alter BIND’s behavior by adding a TCP Listen Backlog Option to a BIND Request, provided that the Request is part of a Session.

In response, the proxy sends a TCP Listen Backlog Option as part of the Operation Reply, with the Backlog field signalling the actual backlog used. The proxy SHOULD NOT use a backlog longer than requested.

Following the successful negotiation of a backlog, the proxy listens for incoming connections for as long as the initial connection stays open. The initial connection is not used to relay data between the client and a remote host.

To accept connections, the client issues further BIND Requests using the bind address and port supplied by the proxy in the initial Operation Reply. Said BIND requests must belong to the same Session as the original Request.

If no backlog is issued, the proxy signals a backlog length of 0, and BIND’s behavior remains unaffected.

8.2. Authentication Method options

A client that is willing to go through the authentication phase MUST include an Authentication Method Advertisement option in its Request. In case of a CONNECT Request, the option is also used to specify the amount of initial data supplied before any method-specific authentication negotiations take place.
Figure 15: Authentication Method Advertisement Option

- Initial Data Size: A two-byte number in network byte order. In case of CONNECT, this is the number of bytes of initial data that are supplied by the client immediately following the Request. This number MUST NOT be larger than \(2^{14}\).

- Methods: One byte per advertised method. Method numbers are assigned by IANA.

- Padding: A minimally-sized sequence of zeroes, such that the option length is a multiple of 4. Note that 0 coincides with the value for "No Authentication Required".

Clients MUST support the "No authentication required" method. Clients SHOULD omit advertising the "No authentication required" option.

The proxy indicates which authentication method must proceed by sending an Authentication Method Selection option in the corresponding Authentication Reply:

Figure 16: Authentication Method Selection Option

Olteanu & Niculescu Expires January 9, 2020 [Page 24]
8.3. Authentication Data options

Authentication Data options carry method-specific authentication data. They can be part of SOCKS Requests and Authentication Replies.

Authentication Data options have the following format:

```
+-------------------------------+-------------------------------+
|           Kind = 4            |            Length             |
+---------------+---------------+-------------------------------+
|    Method     |                                             ...
+---------------+                                             ...
...           Authentication Data (variable length)           ...
...                                                             |
+---------------------------------------------------------------+
```

Figure 17: Authentication Data Option

- **Method**: The number of the authentication method. These numbers are assigned by IANA.

- **Authentication Data**: The contents are specific to each method.

Clients MUST only place one Authentication Data option per authentication method.

8.4. Session options

Clients and proxies can establish SOCKS sessions, which span one or more Requests. All session-related negotiations are done via Session Options, which are placed in Requests and Authentication Replies by the client and, respectively, by the proxy.

Client and proxy implementations MUST either support all Session Option Types, or none.
8.4.1. Session initiation

A client can initiate a session by sending a Session Request Option:

```
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
+-------------------------------+-------------------------------+
|           Kind = 5            |          Length = 4           |
+-------------------------------+-------------------------------+
```

Figure 18: Session Request Option

The proxy then replies with a Session ID Option in the successful Operation Reply:

```
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
+-------------------------------+-------------------------------+
|           Kind = 6            |            Length             |
|                                                             ...
| ...               Session ID (variable length)                ...
| ...                                                             |
+---------------------------------------------------------------+
```

Figure 19: Session ID Option

- Session ID: An opaque sequence of bytes specific to the session. The size MUST be a multiple of 4. MUST NOT be empty.

The Session ID serves to identify the session and is opaque to the client.

The credentials, or lack thereof, used to initiate the session are tied to the session. If authentication is to be performed for further Requests, the session is deemed "untrusted", and the proxy also places a Session Untrusted option in the Authentication Reply:

```
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
+-------------------------------+-------------------------------+
|           Kind = 7            |          Length = 4           |
+-------------------------------+-------------------------------+
```

Figure 20: Session Untrusted Option
The SOCKS Request that initiated the session is considered part of the session. A client MUST NOT attempt to initiate a session from within a different session.

If the proxy can not or will not honor the Session Request, it does so silently.

8.4.2. Further SOCKS Requests

Any further SOCKS Requests that are part of the session MUST include a Session ID Option (as seen in Figure 19).

The authentication procedure is altered based on the Session ID’s validity and whether or not the Session is untrusted.

For valid Session IDs:

- If the session is untrusted, the proxy MUST reject clients that do not authenticate using the same method and credentials that were used to initiate the session.
- Otherwise, the proxy MUST silently ignore any authentication attempt in the Request, and MUST NOT require any authentication.

The proxy then replies by placing a Session OK option in the successful Authentication Reply:

```
+-------------------------------+-------------------------------+
|           Kind = 8            |          Length = 4           |
+-------------------------------+-------------------------------+
```

Figure 21: Session OK Option

If the ticket is invalid, the first Authentication Reply MUST signal that authentication failed and can not continue (by setting the Type field to 0x01). Further, it SHALL contain a Session Invalid option:

```
+-------------------------------+-------------------------------+
|           Kind = 9            |          Length = 4           |
+-------------------------------+-------------------------------+
```

Figure 22: Session Invalid Option
8.4.3. Tearing down the session

Proxies can, at their discretion, tear down a session and free all associated state. Proxy implementations SHOULD feature a timeout mechanism that destroys sessions after a period of inactivity. When a session is terminated, the proxy MAY close all connections associated with said session.

Clients can signal that a session is no longer needed, and can be torn down, by sending a Session Teardown option in addition to the Session ID option:

```
+-------------------------------+-------------------------------+
|           Kind = 10           |          Length = 4           |
+-------------------------------+-------------------------------+
```

Figure 23: Session Teardown Option

After sending such an option, the client MUST assume that the session is no longer valid. The proxy MUST treat the connection-terminating request as if it were not part of any session.

8.5. Idempotence options

To protect against duplicate SOCKS Requests, clients can request, and then spend, idempotence tokens. A token can only be spent on a single SOCKS request.

Tokens are 4-byte unsigned integers in a modular 4-byte space. Therefore, if \( x \) and \( y \) are tokens, \( x \) is less than \( y \) if \( 0 < (y - x) < 2^{31} \) in unsigned 32-bit arithmetic.

Proxies grant contiguous ranges of tokens called token windows. Token windows are defined by their base (the first token in the range) and size.

All token-related operations are done via Idempotence options.

Idempotence options are only valid in the context of a SOCKS Session. If a SOCKS Request is not part of a Session (either by supplying a valid Session ID or successfully initiating one via a Session Request), the proxy MUST silently ignore any Idempotence options.
Token windows are tracked by the proxy on a per-session basis. There can be at most one token window for every session and its tokens can only be spent from within said session.

Client and proxy implementations MUST either support all Idempotence Option Types, or none.

8.5.1. Requesting a token window

A client can obtain a window of tokens by sending an Idempotence Request option as part of a SOCKS Request:

```
+-------------------------------+-------------------------------+
|           Kind = 11           |          Length = 8           |
+-------------------------------+-------------------------------+
|                          Window Size                          |                  |
+---------------------------------------------------------------+
```

Figure 24: Token Request

- Window Size: The requested window size.

Once a token window is issued, the proxy MUST include an Idempotence Window option in all subsequent successful Authentication Replies:

```
+-------------------------------+-------------------------------+
|           Kind = 12           |          Length = 12          |
+-------------------------------+-------------------------------+
|                          Window Base                          |                  |
|                          Window Size                          |
+---------------------------------------------------------------+
```

Figure 25: Idempotence Window

- Window Base: The first token in the window.

- Window Size: The window size. This value MAY differ from the requested window size. Window sizes MUST be less than $2^{31}$. Window sizes MUST NOT be 0.
If no token window is issued, the proxy MUST silently ignore the Token Request. If there is already a token window associated with the session, the proxy MUST NOT issue a new window.

8.5.2. Spending a token

The client can attempt to spend a token by including an Idempotence Expenditure option in its SOCKS request:

```
+-------------------------------+-------------------------------+
|           Kind = 13           |          Length = 8           |
|                             Token                             |
+---------------------------------------------------------------+
```

Figure 26: Idempotence Expenditure

- **Kind**: 13 (Idempotence Expenditure option)
- **Length**: 8
- **Token**: The token being spent.

Clients SHOULD prioritize spending the smaller tokens.

The proxy responds by sending either an Idempotence Accepted or Rejected option as part of the Authentication Reply:

```
+-------------------------------+-------------------------------+
|           Kind = 14           |          Length = 8           |
+-------------------------------+-------------------------------+
```

Figure 27: Idempotence Accepted

```
+-------------------------------+-------------------------------+
|           Kind = 15           |          Length = 8           |
+-------------------------------+-------------------------------+
```

Figure 28: Idempotence Rejected
If eligible, the token is spent before attempting to honor the Request. If the token is not eligible for spending, the Authentication Reply MUST indicate failure.

8.5.3. Shifting windows

Windows can be shifted (i.e. have their base increased, while retaining their size) unilaterally by the proxy.

Proxy implementations SHOULD shift the window: * as soon as the lowest-order token in the window is spent and * when a sufficiently high-order token is spent.

Proxy implementations SHOULD NOT shift the window’s base beyond the highest unspent token.

8.5.4. Out-of-order Window Advertisements

Even though the proxy increases the window’s base monotonically, there is no mechanism whereby a SOCKS client can receive the Token Window Advertisements in order. As such, clients SHOULD disregard Token Window Advertisements with a Window Base less than the previously known value.

8.6. Address Resolution Options

Clients wishing to resolve an address from the proxy’s vantage point can do so by sending a Resolution Request option:

```
| Kind = 16 | Length = 4 |
```

Figure 29: Resolution Request

The proxy’s reply is included in the Operation Reply.

8.6.1. Forward resolution

If the address supplied by the client is a Domain Name, the proxy sends a Resolution option for each supported address type:
If resolving a given address type is supported, but no addresses were found, the proxy MUST send an empty option, rather than none at all. If the proxy does not support resolving either one of the address types, it MUST omit sending the corresponding option.

8.6.2. Reverse resolution

If the client supplies either an IPv4 or IPv6 address, the proxy performs a reverse lookup and replies with a list of domain names:
Figure 32: Domain Name Resolution

- Domain Names: The resolved domain names. Their format is the same as the one used in Requests and Operation Replies, with each string being individually padded. (See Section 4.)

If no domain names could be found, the proxy MUST send an empty option. If reverse resolution is unsupported, the proxy MUST NOT send any such option.

9. Username/Password Authentication

Username/Password authentication is carried out as in [RFC1929].

Clients can also attempt to authenticate by placing the Username/Password request in an Authentication Data Option.

Figure 33: Password authentication via a SOCKS Option

- Username/Password Request: The Username/Password Request, as described in [RFC1929].
Proxies reply by including a Authentication Data Option in the next Authentication Reply which contains the Username/Password reply:

```
+-------------------------------+-------------------------------+
|           Kind = 4            |          Length = 8           |
+---------------+---------------+---------------+---------------+
|  Method = 2   |    Username/Password Reply   |  Padding = 0  |
+---------------+-------------------------------+---------------+
```

Figure 34: Reply to password authentication via a SOCKS Option

- Username/Password Reply: The Username/Password Reply, as described in [RFC1929].

10. TCP Fast Open on the Client-Proxy Leg

TFO breaks TCP semantics, causing replays of the data in the SYN’s payload under certain rare circumstances [RFC7413]. A replayed SOCKS Request could itself result in a replayed connection on behalf of the client.

As such, client implementations SHOULD NOT use TFO on the client-proxy leg unless:

- The protocol running on top of SOCKS tolerates the risks of TFO, or
- The SYN’s payload does not contain any application data (so that no data is replayed to the server, even though duplicate connections are still possible), or
- The client uses Idempotence Options, making replays very unlikely, or
- SOCKS is running on top of TLS and Early Data is not used.

11. False Starts

In case of CONNECT Requests, the client MAY start sending application data as soon as possible, as long as doing so does not incur the risk of breaking the SOCKS protocol.

Clients must work around the authentication phase by doing any of the following:
If the Request does not contain an Authentication Method Advertisement option, the authentication phase is guaranteed not to happen. In this case, application data MAY be sent immediately after the Request.

Application data MAY be sent immediately after receiving an Authentication Reply indicating success.

When performing a method-specific authentication sequence, application data MAY be sent immediately after the last client message.

12. Security Considerations

12.1. Large requests

Given the format of the request message, a malicious client could craft a request that is in excess of 16 KB and proxies could be prone to DDoS attacks.

To mitigate such attacks, proxy implementations SHOULD be able to incrementally parse the requests. Proxies MAY close the connection to the client if:

- the request is not fully received after a certain timeout, or
- the number of options or their size exceeds an imposed hard cap.

12.2. Replay attacks

In TLS 1.3, early data (which is likely to contain a full SOCKS request) is prone to replay attacks.

While Token Expenditure options can be used to mitigate replay attacks, anything prior to the initial Token Request is still vulnerable. As such, client implementations SHOULD NOT make use of TLS early data unless the Request attempts to spend a token.

12.3. Resource exhaustion

Malicious clients can issue a large number of Session Requests, forcing the proxy to keep large amounts of state.

To mitigate this, the proxy MAY implement policies restricting the number of concurrent sessions on a per-IP or per-user basis, or barring unauthenticated clients from establishing sessions.
13. IANA Considerations

This document requests that IANA allocate 2-byte option kinds for
SOCKS 6 options. Further, this document requests the following
option kinds:

- Unassigned: 0
- Stack: 1
- Authentication Method Advertisement: 2
- Authentication Method Selection: 3
- Authentication Data: 4
- Session Request: 5
- Session ID: 6
- Session Untrusted: 7
- Session OK: 8
- Session Invalid: 9
- Session Teardown: 10
- Idempotence Request: 11
- Idempotence Window: 12
- Idempotence Expenditure: 13
- Idempotence Accepted: 14
- Idempotence Rejected: 15
- Resolution Request: 16
- IPv4 Resolution: 17
- IPv6 Resolution: 18
- Domain Name Resolution: 19
- Vendor-specific: 64512–0xFFFF
This document also requests that IANA allocate a TCP and UDP port for SOCKS over TLS and DTLS, respectively.

14. Acknowledgements

The protocol described in this draft builds upon and is a direct continuation of SOCKS 5 [RFC1928].

15. References

15.1. Normative References


15.2. Informative References


Authors’ Addresses

Vladimir Olteanu
University Politehnica of Bucharest
313 Splaiul Independentei, Sector 6
Bucharest
Romania

Email: vladimir.olteanu@cs.pub.ro

Dragos Niculescu
University Politehnica of Bucharest
313 Splaiul Independentei, Sector 6
Bucharest
Romania

Email: dragos.niculescu@cs.pub.ro
Abstract

This document defines a method which helps an SCTP sender to understand when a received SACK acknowledges the original transmission of a TSN or its retransmission. It is done by specifying a new bit, called Retransmit bit (R-bit), in the header of DATA, I-DATA and SACK chunks. The bit is used when a TSN is retransmitted and returned back in the acknowledgement.

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
the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction ............................................. 2
2. Conventions ............................................. 3
3. Updates in SCTP Chunks Header .................................. 3
   3.1. R-bit in DATA Chunk Header ............................. 3
   3.2. R-bit in I-DATA Chunk Header ............................ 3
   3.3. R-bit in SACK Chunk Header ............................. 4
4. Procedures .............................................. 5
   4.1. Negotiation ........................................... 5
   4.2. Sender Side Considerations ............................. 6
   4.3. Receiver Side Considerations ........................... 7
   4.4. Processing of SACK with and without R-bit .............. 7
5. Interoperability Considerations ................................ 8
6. Socket API Considerations .................................... 8
7. Acknowledgements .......................................... 9
8. IANA Considerations ......................................... 9
9. Security Considerations ...................................... 10
10. References ............................................... 10
    10.1. Normative References ................................ 11
    10.2. Informative References .............................. 11
Author's Address ........................................... 12

1. Introduction

SCTP which is defined in [RFC4960] is a reliable message-oriented protocol. The SCTP sender splits user messages to DATA chunks and sends them to the receiver. The SCTP receiver uses the SACK chunk to acknowledge incoming data. The reliability in SCTP is achieved by the retransmission of DATA chunks which were not acknowledged.

If a DATA chunk has been retransmitted at least once, at SACK reception SCTP cannot understand if the SACK was sent in response to the originally sent DATA or retransmitted one. Thus, due to that ambiguity, [RFC4960] prohibits making RTT measurements. Some other SCTP mechanisms such as loss recovery and congestion control are not accurate in that case either.

This document describes a simple extension of the DATA and SACK chunks by a new bit, so called Retransmit bit (R-bit). The sender sets the R-bit in the DATA chunk header when it retransmits a DATA and the receiver sets it in the SACK chunk header when a DATA with R-bit is acknowledged. The sender can now distinguish when a SACK acknowledges the originally sent DATA or retransmitted one. The extension requires support by the sender and the receiver.
The mechanism described in this document is equally relevant for I-DATA chunk which is introduced in [RFC8260].

2. Conventions

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

3. Updates in SCTP Chunks Header

3.1. R-bit in DATA Chunk Header

Figure 1 describes the extended DATA chunk header.

```
 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.
Figure 3: Extended SACK chunk

The only difference between the SACK chunk in Figure 3 and the SACK chunk defined in [RFC4960] is the addition of the R-bit in the flags field of the SACK chunk header. [RFC4960] specified that bit as Reserved and that it should be set to 0 by the sender and ignored by the receiver.

4. Procedures

4.1. Negotiation

R-bit MUST NOT be used unless both SCTP peers negotiated its support.

The following new optional parameter is added to the INIT and INIT ACK chunks to negotiate R-bit support during association setup:
The parameter format is the following:

```
 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
+-------------------------------------------+-------------------+
|    Parameter Type = 0x8100     |    Parameter Length = 4      |
+-------------------------------------------+-------------------+
```

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].
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
TinyMT32 Pseudo Random Number Generator (PRNG)
draft-roca-tsvwg-tinymt32-01

Abstract

This document describes the TinyMT32 Pseudo Random Number Generator (PRNG) that produces 32-bit pseudo-random unsigned integers and aims at having a simple-to-use and deterministic solution. This PRNG is a small-sized variant of Mersenne Twister (MT) PRNG, also designed by M. Saito and M. Matsumoto. The main advantage of TinyMT32 over MT is the use of a small internal state, compatible with most target platforms including embedded devices, while keeping a reasonably good randomness.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 6, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of
1. Introduction

This document specifies the TinyMT32 PRNG, as a specialization of the reference implementation version 1.1 (2015/04/24) by Mutsuo Saito and Makoto Matsumoto, from Hiroshima University:

- Official web site: <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/TINYMT/>
- Official github site and reference implementation: <https://github.com/MersenneTwister-Lab/TinyMT>

This specialisation aims at having a simple-to-use and deterministic PRNG, as explained below.

TinyMT is a new small-sized variant of Mersenne Twister (MT) introduced by Mutsuo Saito and Makoto Matsumoto in 2011. This document focusses on the TinyMT32 variant (rather than TinyMT64) of the PRNG, which outputs 32-bit unsigned integers.

The purpose of TinyMT is not to replace Mersenne Twister: TinyMT has a far shorter period than MT. The merit of TinyMT is in its small size of the internal state of 127 bits, far smaller than 19937 bits of MT. According to statistical tests (BigCrush in TestU01 <http://simul.iro.umontreal.ca/testu01/tu01.html> and AdaptiveCrush
the quality of the outputs of TinyMT seems pretty good, taking the small size of the internal state into consideration. From this point of view, TinyMT32 represents a major improvement with respect to the Park-Miler Linear Congruential PRNG (e.g., as specified in [RFC5170]).

The TinyMT32 PRNG initialization depends, among other things, on a parameter set -- namely (mat1, mat2, tmat) -- that needs to be well chosen (pre-calculated values are available in the official web site). In order to facilitate the use of this PRNG, and unlike the implementation version 1.1 (2015/04/24) by Mutsuo Saito and Makoto Matsumoto, this specification requires the use of a specific parameter set (see Section 3.1). The implementation version 1.1 (2015/04/24) also proposes two initialisation functions that differ on the approach to seed the PRNG. A second difference is the removal of the tinymt32_init_by_array() function to keep only the simple initialisation through a single seed value (see Section 3.2).

Finally, the determinism of this PRNG, for a given seed, has been carefully checked (see Section 3.3). Indeed, this determinism can be a key requirement as it the case with [RLC-ID] that normatively depends on this specification.

2. Definitions

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

3. TinyMT32 PRNG Specification

3.1. TinyMT32 Source Code

The TinyMT32 PRNG requires to be initialized with a parameter set that needs to be well chosen. In this specification, for the sake of simplicity, the following parameter set MUST be used:

- mat1 = 0x8f7011ee = 2406486510
- mat2 = 0xfc78ff1f = 4235788063
- tmat = 0x3793fdff = 932445695

This parameter set is the first entry of the precalculated parameter sets in file tinymt32dc.0.1048576.txt, by Kenji Rikitake, and available at <https://github.com/jjibdx/tinymtdc-
longbatch/blob/master/tinymt32dc/tinymt32dc.0.1048576.txt>. This is also the parameter set used in [KR12].

The TinyMT32 PRNG reference implementation is reproduced in Figure 1, with the following differences with respect to the original source code:

- the original copyright and licence have been removed, in accordance with BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info);
- 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;
- 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.

```c
/**
 * Tiny Mersenne Twister only 127 bit internal state.
 * Derived from the reference implementation version 1.1 (2015/04/24)
 * by Mutsuo Saito (Hiroshima University) and Makoto Matsumoto
 * (Hiroshima University).
 */
#include <stdint.h>

/**
 * tinymt32 internal state vector and parameters
 */
typedef struct {
    uint32_t status[4];
    uint32_t mat1;
    uint32_t mat2;
    uint32_t tmat;
} tinymt32_t;

static void tinymt32_next_state (tinymt32_t * s);
static uint32_t tinymt32_temper (tinymt32_t * s);

/**
 * Parameter set to use for this IETF specification. Don’t change.
 */
This parameter set is the first entry of the precalculated parameter sets in file tinymt32dc.0.1048576.txt, by Kenji Rikitake, available at:
It is also the parameter set used:

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]
                           ^ (s->status[(i - 1) & 3] >> 30));
    }
    /*
    * NB: the parameter set of this specification warrants
    * that none of the possible 2^32 seeds leads to an all-zero 127-bit internal state. Therefore, the period_certification() function of the original TinyMT32 source code has been safely removed. If another parameter set is used, this function will have to be re-introduced here.
    */
    for (int i = 0; i < PRE_LOOP; i++) {
        tinymt32_next_state(s);
    }
}

**
* This function outputs a 32-bit unsigned integer from
  * the internal state.
  * @param s pointer to tinymt internal state.
  * @return 32-bit unsigned integer r (0 <= r < 2^32).
  */
uint32_t tinymt32_generate_uint32 (tinymt32_t * s)
{
    tinymt32_next_state(s);
    return tinymt32_temper(s);
}

/**
  * Internal tinymt32 constants and functions.
  * Users should not call these functions directly.
  */
const uint32_t  TINYMT32_SH0 = 1;
const uint32_t  TINYMT32_SH1 = 10;
const uint32_t  TINYMT32_SH8 = 8;
const uint32_t  TINYMT32_MASK = UINT32_C(0x7fffffff);

/**
  * This function changes the internal state of tinymt32.
  * @param s pointer to tinymt internal state.
  */
static void tinymt32_next_state (tinymt32_t * s)
{
    uint32_t x;
    uint32_t y;

    y = s->status[3];
    x = (s->status[0] & TINYMT32_MASK)
        ^ s->status[1]
        ^ s->status[2];
    x ^= (x << TINYMT32_SH0);
    y ^= (y >> TINYMT32_SH0) ^ x;
    s->status[0] = s->status[1];
    s->status[1] = s->status[2];
    s->status[2] = x ^ (y << TINYMT32_SH1);
    s->status[3] = y;
    /*
     * The if (y & 1) {...} block below replaces:
     *     s->status[1] ^= -((int32_t)(y & 1)) & s->mat1;
     *     s->status[2] ^= -((int32_t)(y & 1)) & s->mat2;
     * The adopted code is equivalent to the original code
     * but does not depend on the representation of negative
     * integers by 2’s complements. It is therefore more
     * portable, but includes an if-branch which may slow
     * down the generation speed.
     */
}
if (y & 1) {
    s->status[1] ^= s->mat1;
    s->status[2] ^= s->mat2;
}

static uint32_t tinymt32_temper (tinymt32_t * s)
{
    uint32_t t0, t1;
    t0 = s->status[3];
    t1 = s->status[0] + (s->status[2] >> TINYMT32_SH8);
    t0 ^= t1;
    t0 ^= -((int32_t)(t1 & 1)) & s->tmat;
    return t0;
}

3.2. TinyMT32 Usage

This PRNG MUST first be initialized with the following function:

    void tinymt32_init (tinymt32_t * s, uint32_t seed);

It takes as input a 32-bit unsigned integer used as a seed (note that value 0 is authorized by TinyMT32). This function also takes as input a pointer to an instance of a tinymt32_t structure that needs to be allocated by the caller but left uninitialized. This structure will then updated by the various TinyMT32 functions in order to keep the internal state of the PRNG. The use of this structure authorizes several instances of this PRNG to be used in parallel, each of them having its own instance of the structure.

Then, each time a new 32-bit pseudo-random unsigned integer between 0 and $2^{32} - 1$ inclusive is needed, the following function is used:

    uint32_t tinymt32_generate_uint32 (tinymt32_t * s);

Of course, the tinymt32_t structure must be left unchanged by the caller between successive calls to this function.
3.3. Specific Implementation Validation and Deterministic Behavior

PRNG determinism, for a given seed, can be a requirement (e.g., with [RLC-ID]). Consequently, any implementation of the TinyMT32 PRNG in line with this specification MUST comply with the following criteria. Using a seed value of 1, the first 50 values returned by tinymt32_generate_uint32(s) as 32-bit unsigned integers MUST be equal to values provided in Figure 2. Note that these values come from the tinymt/check32.out.txt file provided by the PRNG authors to validate implementations of TinyMT32, as part of the MersenneTwister-Lab/TinyMT Github repository.

2545341989 981918433 3715302833 2387538352 3591001365 3820442102 2114400566 2196103051 2783359912 764534509 643179475 1822416315 881558334 4207026366 3690273640 3240535687 2921447122 3984931427 4092394160 44209675 2188315343 2908663843 1834519336 3774670961 3019990707 4065554902 1239765502 4035716197 3412127188 552822483 161364450 353727785 140085994 149132008 2547770827 4064042525 4078297538 2057335507 622384752 2041665899 2139813187 1080849512 33160901 662956935 642999063 3384709977 1723175122 3866752252 521822317 2292524454

Figure 2: First 50 decimal values returned by tinymt32_generate_uint32(s) as 32-bit unsigned integers, with a seed value of 1.

In particular, the deterministic behavior of the Figure 1 source code has been checked across several platforms: high-end laptops running 64-bits Mac OSX and Linux/Ubuntu; a board featuring a 32-bits ARM Cortex-A15 and running 32-bit Linux/Ubuntu; several embedded cards featuring either an ARM Cortex-M0+, a Cortex-M3 or a Cortex-M4 32-bit microcontroller, all of them running RIOT [Baccelli18]; two low-end embedded cards featuring either a 16-bit microcontroller (TI MSP430) or a 8-bit microcontroller (Arduino ATMEGA2560), both of them running RIOT.

This specification only outputs 32-bit unsigned pseudo-random numbers and does not try to map this output to a smaller integer range (e.g., between 10 and 49 inclusive). If a specific use-case needs such a mapping, it will have to provide its own function. In that case, if PRNG determinism is also required, the use of floating point (single or double precision) to perform this mapping should probably be avoided, these calculations leading potentially to different rounding errors across different target platforms. Great care should also be put on not introducing biases in the randomness of the mapped output (it may be the case with some mapping algorithms) incompatible with...
the use-case requirements. The details of how to perform such a mapping are out-of-scope of this document.

4. Security Considerations

The authors do not believe the present specification generates specific security risks per se.

5. IANA Considerations

This document does not require any IANA action.

6. Acknowledgments

The authors would like to thank Belkacem Teibi with whom we explored TinyMT32 specificities when looking to an alternative to the Park-Miler Linear Congruential PRNG. The authors would like to thank the three TSVWG chairs, Wesley Eddy, our shepherd, David Black and Gorry Fairhurst, as well as Spencer Dawkins and Mirja Kuhlewind. Last but not least, the authors are really grateful to the IESG members, in particular Benjamin Kaduk, Eric Rescorla, and Adam Roach for their highly valuable feedbacks that greatly contributed to improve this specification.

7. References

7.1. Normative References


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
Abstract

NOTE: This document is a reformatted version of [LLD-white-paper].

The evolution of the bandwidth capabilities - from kilobits per second to gigabits - across generations of DOCSIS cable broadband technology has paved the way for the applications that today form our digital lives. Along with increased bandwidth, or "speed", the latency performance of DOCSIS technology has also improved in recent years. Although it often gets less attention, latency performance contributes as much or more to the broadband experience and the feasibility of future applications as does speed.

Low Latency DOCSIS technology (LLD) is a specification developed by CableLabs in collaboration with DOCSIS vendors and cable operators that tackles the two main causes of latency in the network: queuing delay and media acquisition delay. LLD introduces an approach wherein data traffic from applications that aren’t causing latency can take a different logical path through the DOCSIS network without getting hung up behind data from applications that are causing latency, as is the case in today’s Internet architectures. This mechanism doesn’t interfere with the way applications share the total bandwidth of the connection, and it doesn’t reduce one application’s latency at the expense of others. In addition, LLD improves the DOCSIS upstream media acquisition delay with a faster request-grant loop and a new proactive scheduling mechanism. LLD makes the internet experience better for latency sensitive applications without any negative impact on other applications.

The latest generation of DOCSIS equipment that has been deployed in the field - DOCSIS 3.1 - experiences typical latency performance of around 10 milliseconds (ms) on the Access Network link. However, under heavy load, the link can experience delay spikes of 100 ms or more. LLD systems can deliver a consistent 1 ms delay on the DOCSIS network for traffic that isn’t causing latency, imperceptible for nearly all applications. The experience will be more consistent with much smaller delay variation.
LLD can be deployed by field-upgrading DOCSIS 3.1 cable modem and cable modem termination system devices with new software. The technology includes tools that enable automatic provisioning of these new services, and it also introduces new tools to report statistics of latency performance to the operator.

Cable operators, DOCSIS equipment manufacturers, and application providers will all have to act in order to take advantage of LLD. This white paper explains the technology and describes the role that each of these parties plays in making LLD a reality.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 12, 2019.

Copyright Notice

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

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

Let’s begin with bandwidth (or "speed"): the amount of data that can be delivered across a network connection over a period of time. Sometimes bandwidth is very important to the broadband experience, particularly when an application is trying to send or receive large amounts of data, such as watching videos on Netflix, downloading videos/music, syncing file-shares or email clients, uploading a video to YouTube or Instagram, or downloading a new application or system update. Other times, bandwidth (or bandwidth alone) isn’t enough, and latency has a big effect on the user experience.

Latency is the time that it takes for a short message (a packet, in networking terminology) to make it across the network from the sender to the receiver and for a response to come back. Network latency is commonly measured as round-trip-time and is sometimes referred to as "ping time." Applications that are more interactive or real-time,
Like web browsing, online gaming, and video conferencing/chatting, perform the best when latency is kept low, and adding more bandwidth without addressing latency doesn’t make things better.

When multiple applications share the broadband connection of one household (e.g., several users doing different activities at the same time), each of those applications can have an impact on the performance of the others. They all share the total bandwidth of the connection (so more active applications mean less bandwidth for each one), and they can all cause the latency of the connection to increase.

It turns out that applications today that want to send a lot of data all at once do a reasonably good job of sharing the bandwidth in a fair manner, but they actually cause a pretty big latency problem when they do it because they send data too quickly and expect the network to queue it up. We call these applications "queue-building" applications, e.g., video streaming (Netflix). There are also plenty of other applications that don’t send data too quickly, so they don’t cause latency. We call these "non-queue-building" applications, e.g., video chatting (FaceTime).

LLD separates these two types of traffic into two logical queues, which greatly improves the latency experienced by the non-queue-building applications (many of which may be latency-sensitive) without having any downside for the queue-building applications. In addition, two queues allow LLD to support a next-generation application protocol that can scale up to sending data at 10 Gbps and beyond while maintaining ultra-low queuing delay, which means that in the future, there may not be queue-building applications at all.

As of the writing of this document, the Low Latency DOCSIS specifications have just been published ([DOCSIS-MULPIv3.1], [DOCSIS-CCAP-OSSIv3.1], [DOCSIS-CM-OSSIv3.1]), and DOCSIS equipment manufacturers are working on building support for the functionality. In addition, work is underway in the Internet Engineering Task Force to standardize low-latency architectures across the broader Internet ecosystem.

2. Latency in DOCSIS Networks

Low Latency DOCSIS technology is the next step in a progression of latency improvements that have been made to the DOCSIS specifications by CableLabs in recent years. Table 1 provides a snapshot of the milestones in round-trip latency performance with DOCSIS technology from the first DOCSIS 3.0 equipment to DOCSIS 3.1 equipment that supports [RFC8034] Active Queue Management, and finally the new Low Latency DOCSIS, which achieves ~1 ms of round-trip latency. The
table references three metrics that describe the range of latencies added by the DOCSIS network link that would be experienced by a broadband user. The first, "When Idle," refers to a broadband connection that is not being actively used by the customer. The second, "Under Load," represents average latency while the user is actively using the service (e.g., streaming video). Finally, the third, "99th Percentile," gives an indication of the maximum latency that a customer would commonly experience in real usage scenarios. The table uses order-of-magnitude numbers because the actual performance will vary because of a number of factors including DOCSIS channel configuration and actual application usage pattern.

For latency-sensitive applications, the 99th percentile value has the most impact on user experience.

TABLE 1. EVOLUTION OF LATENCY PERFORMANCE IN DOCSIS NETWORKS (ROUND-TRIP TIME IN MILLISECONDS BETWEEN THE CM AND CMTS)

<table>
<thead>
<tr>
<th></th>
<th>When Idle</th>
<th>Under Load</th>
<th>99th Percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOCSIS 3.0 Early Equipment</td>
<td>˜10 ms</td>
<td>˜1000 ms</td>
<td>˜1000 ms</td>
</tr>
<tr>
<td>DOCSIS 3.0 w/ Buffer Control</td>
<td>˜10 ms</td>
<td>˜100 ms</td>
<td>˜100 ms</td>
</tr>
<tr>
<td>DOCSIS 3.1 Active Queue</td>
<td>˜10 ms</td>
<td>˜10 ms</td>
<td>˜100 ms</td>
</tr>
<tr>
<td>Management</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low Latency DOCSIS 3.1</td>
<td>˜1 ms</td>
<td>˜1 ms</td>
<td>˜1 ms</td>
</tr>
</tbody>
</table>

Table 1

The latency described in Table 1 is caused by a series of factors in the DOCSIS cable modem (CM) and cable modem termination system (CMTS). Figure 1 in [LLD-white-paper] illustrates the range of latencies caused by those factors in DOCSIS 3.1 networks.

The lowest two latency sources in Figure 1 in [LLD-white-paper] have minor impacts on overall latency.

The "Switching/Forwarding" delay represents the amount of time it takes for the CM and CMTS to make the decision to forward a packet. This has a very minor impact on overall latency.

The "Propagation" delay (the amount of time it takes for a signal to travel on the HFC plant) is set by the speed of light and the distance from CM to CMTS. Not much can be done to affect latency from this source.
Of the sources in Figure 1 in [LLD-white-paper], the top three significantly drive latency performance.

The range of the "Serialization/Encoding" delay comes from the upstream and downstream channel configuration options available to the operator. Some of these configurations provide significant robustness benefits at the expense of latency, whereas others may be less robust to noise but provide very low latency. The LLD specification does not modify the set of options available to the operator. Rather, operators should be encouraged to use the lowest latency channel configurations that they can, given the plant conditions.

The "Media Acquisition" delay is a result of the shared-medium scheduling currently provided by DOCSIS technology, in which the CMTS arbitrates access to the upstream channel via a request–grant mechanism.

The "Queuing" delay is mainly caused by the current TCP protocol and its variants. Applications today that need to seek out as much bandwidth as possible use a transport protocol like TCP (or the TCP-replacement known as QUIC), which uses a "congestion control" algorithm (such as Reno, Cubic, or BBR) to adjust to the available bandwidth at the bottleneck link through the network. Typically, this will be the last mile link - the DOCSIS link for cable customers - where the bandwidth available for each application often varies rapidly as the activity of all the devices in the household varies.

With today’s congestion control algorithms, the sender ramps up the sending rate until it’s sending data faster than the bottleneck link can support. Packets then start queuing in a buffer at the entrance to the link, i.e. the CM or CMTS. This queue of packets grows quickly until the device decides to discard some newly arriving packets, which triggers the sender to pause for a bit in order to allow the buffer to drain somewhat before resuming sending. This process is an inherent feature of the TCP family of Internet transport protocols, and it repeats over and over again until the file transfer completes. In doing so, it causes latency and packet loss for all of the traffic that shares the broadband link.

LLD tackles the two main causes of latency in the network: queuing delay and media acquisition delay.

- LLD addresses Queuing Delay by allowing non-queue-building applications to avoid waiting behind the delays caused by the current TCP or its variants. At a high level, the low-latency architecture consists of a dual-queue approach that treats both queues as a single pool of bandwidth.
LLD cuts Media Acquisition Delay by using a faster request-grant loop and by adding support for a new proactive scheduler that can provide extremely low latency service.

In addition, LLD introduces detailed statistics on queueing delay via histogram calculations performed by the CM (for upstream) and CMTS (for downstream). Furthermore, CableLabs is working with a broad cross-section of stakeholders in the IETF to standardize an end-to-end service architecture that can leverage LLD to enable even high bandwidth TCP flows to achieve ultra-low queuing delay. This technology will be important for future, interactive high-data-rate applications like holographic light field experiences, as well as for enabling higher performance versions of today’s applications like web and video conferencing.

The sections below describe these features in more detail.

3. New Dual-Queue Approach

Of all the features of LLD, the dual-queue mechanism has by far the greatest impact on round-trip latency and latency variation. The concept of the dual-queue approach is that the majority of the applications that use the internet can be divided into two categories:

- Queue-Building Applications: These application traffic flows frequently send data faster than the path between sender and receiver can support. The most common instance of queue-building flows are flows that use the current TCP or QUIC protocols. As discussed above, these capacity-seeking protocols use a legacy congestion control algorithm that probes for available capacity on the path by sending data faster than the path can support and expecting the network to queue the excess data in internal buffers. The majority of traffic (by volume) today is queue-building. Some examples of queue-building applications are video streaming (e.g., Netflix, YouTube) and application downloads.

- Non-Queue-Building Applications: These application traffic flows very rarely send data faster than the path can support. They come in two subcategories:
  * Today’s self-limited, non-capacity-seeking apps, such as multiplayer online games and IP communication apps (such as Skype or FaceTime). These applications send data at a relatively low data rate and generally space their packets out in a manner that does not cause a queue to form in the network.
Future capacity-seeking TCP/QUIC applications that adopt the new L4S congestion control algorithm (see Appendix A) and so can immediately respond to fast congestion signals sent by the network. These applications are still in development, as networks must first support L4S before applications are able to take advantage, but some prime candidates are web browsing, cloud VR, and interactive light field experiences.

Queue-building (QB) application flows are the source of queueing delay, and today’s non-queue-building (NQB) apps typically suffer from the latency caused by the QB flows.

The purpose of the dual-queue mechanism is to segment queue-building traffic from non-queue-building traffic in a manner that can be readily implemented in DOCSIS 3.1 equipment and that doesn’t alter the overall bandwidth of the broadband service.

By segmenting these two types of applications into separate queues, each can get optimal performance. The QB traffic can build a queue and achieve the necessary and expected throughput performance, and the NQB traffic can take advantage of the available lower latencies by avoiding the delay caused by the QB flows. It is important to note that this segmentation of traffic isn’t for purposes of giving one class of traffic benefits at the expense of the other – it isn’t a high-priority queue and a low-priority queue. Instead, each queue is optimized for the distinct features and requirements of the two classes of traffic, enabling increased functionality and adding value for the broadband user. This is smart network management at work.

3.1. Low-Latency Aggregate Service Flows

DOCSIS 3.1 equipment, like equipment built against earlier versions of the specification, supports a number of upstream and downstream Service Flows (SFs). These Service Flows are logical pipes that are defined by their configured Quality of Service (QoS) parameters (most commonly, the rate shaping parameters [MULPIv3.1] that specify the speed of user connections) and that carry a subset of the traffic to/from a particular CM, as specified by a set of packet classifiers configured by the operator. Traditionally, each Service Flow provides near-complete isolation of its traffic from the traffic transiting other Service Flows (those on the same CM as well as those on other CMs) – each Service Flow has its own buffer and queue and is scheduled independently by the CMTS.

Typically, the operator defines a service offering via the configuration of a single upstream Service Flow and a single downstream Service Flow with rate shaping enabled, and all of the user’s traffic transits these two Service Flows.
The DOCSIS 3.1 specification already includes optional support in the CMTS for a mechanism to group any number of the Service Flows serving a particular CM. LLD leverages and extends this "Aggregate Service Flow" (ASF) feature to establish (and group) a pair of Service Flows in each direction specifically to enable low-latency services. One of the Service Flows in the pair (the "Low Latency Service Flow") will carry NQB traffic, and the other Service Flow (the "Classic Service Flow") will carry QB traffic. The Aggregate Service Flow is configured for the service's rate shaping setting, and the two constituent Service Flows inside the Aggregate have rate shaping disabled. The result is that the operator can configure the total aggregate rate of the service offering in each direction and does not have to configure (or even consider) how much of the user’s traffic is likely to be NQB vs QB.

Figure 2 in [LLD-white-paper] illustrates an example configuration of broadband service as it might look in a current DOCSIS deployment, as well as how it would look with Low Latency DOCSIS. In the traditional configuration, there is a single downstream Service Flow with a rate of 100 Mbps and a single upstream Service Flow with a rate of 20 Mbps. In the LLD configuration, there is a single downstream Aggregate Service Flow with a rate of 100 Mbps, containing two individual Service Flows, one for Low Latency traffic and one for Classic traffic. Similarly, there is single upstream Aggregate Service Flow with a rate of 20 Mbps, containing two individual Service Flows for Low Latency and Classic traffic.

The CMTS will enforce the Aggregate "Max Sustained Traffic Rate" (AMSR), and the end-user’s applications determine how much of the aggregate bandwidth they consume irrespective of which SF they use – just as they do today with a single DOCSIS SF.

As described later, Inter-Service-Flow scheduling is arranged to make the ASF function as a single pool of bandwidth.

3.2. Identifying NQB Packets – Default Classifiers

By default, the traffic within an Aggregate Service Flow is segmented into the two constituent Service Flows by a set of packet classifiers (see Figure 3 in [LLD-white-paper]) that examine the Differentiated Services (DiffServ) Field and the Explicit Congestion Notification (ECN) Field, which are standard elements of the IPv4/IPv6 header [RFC3168]. Specifically, packets with an NQB DiffServ value or an ECN field indicating either ECN Capable Transport 1 (ECT(1)) or Congestion Experienced (CE) will get mapped to the Low Latency Service Flow, and the rest of the traffic will get mapped to the Classic Service Flow.
As of the writing of this draft, it is proposed that the DiffServ value 0x2A be standardized in IETF/IANA to indicate NQB [I-D.white-tsvwg-nqb]. Certain existing DiffServ values may also be classified as NQB by default, such as Expedited Forwarding (EF).

The expectation is that non-queue-building traffic sources (applications) will either mark their packets with an NQB DiffServ value or support ECN.

Although the DiffServ Field is being used to indicate NQB behavior, that does not imply adoption of the Differentiated Services architecture as it is typically understood. In the traditional DiffServ architecture, applications indicate a desire for a particular treatment of their packets – often implemented as a priority level – which in essence conveys a value judgement as to the importance of that traffic relative to the traffic of other applications. Such an architecture can work just fine in a managed environment where all applications conform to a common view of their relative priority levels and so can be trusted to mark their packets appropriately. It fails, however, when applications need to send packets across trust boundaries between networks, where there would be no common view on their relative importance. As a result, the DiffServ architecture is often used within managed networks (corporate networks, campus networks, etc.) but is not used on the Internet.

LLD’s usage of the DiffServ Field to indicate NQB sidesteps this fundamental problem by eliminating the subjective value judgement on the relative importance of applications. Instead, this usage of the DiffServ Field describes objectively verifiable behavior on the part of the application – that it will not build a queue. Therefore, networks can verify that the marking has been applied properly before a packet is allowed into the Low Latency Service Flow queue (see Section 3.4).

The ECN classifiers enable LLD’s support of the IETF’s Low Latency Low Loss Scalable throughput (L4S) service [I-D.ietf-tsvwg-ecn-l4s-id], which is an evolution of the original ECN facility to support applications needing both high bandwidth and low latency (see Appendix A).

3.3. Coupled AQM

To manage queuing delay, both the Low Latency Service Flow queue and the Classic Service Flow queue support Active Queue Management (AQM) (see Figure 4 in [LLD-white-paper]).
In the case of the Classic Service Flow, the queue implements the same state-of-the-art Active Queue Management techniques used in today’s DOCSIS 3.1 networks. For upstream Classic Service Flows, the DOCSIS 3.1 specification mandates that the CM implement the DOCSIS-PIE (Proportional-Integral-Enhanced AQM Algorithm), which introduces packet drops at an appropriate rate to drive the queue delay to the default target value of 10 ms. For downstream Classic Service Flows, the AQM in the CMTS is still vendor specific.

In the case of the Low Latency Service Flow, the queue supports L4S congestion controllers by implementing an Immediate Active Queue Management algorithm that utilizes ECN marking instead of packet drops. By default, the algorithm does not mark the packet if the queuing delay is less than 0.475 milliseconds and always marks the packet if the delay is greater than 1 ms. Between those configurable values, the algorithm marks at a rate that ramps up from 0% to 100% over the range. In addition, per [I-D.ietf-tsvwg-aqm-dualq-coupled], the Immediate AQM in the Low Latency Queue is coupled to the Classic Queue AQM so that congestion in the Classic Queue will induce ECN marking in the Low Latency Queue that will act to balance the per-flow throughput across all of the flows in both queues. L4S congestion control and the role of the dual-queue-coupled-aqm in providing flow balance is described further in Appendix A.

To enable the Low Latency Queue to rapidly dequeue an arrived burst of traffic, the Inter-Service-Flow scheduler gives a higher weight to the Low Latency Queue than it does to the Classic Queue. The coupling to the Low Latency AQM counterbalances the weighted scheduler by making low-latency applications leave space for Classic traffic. This ensures that the weighted scheduler does not give priority over bandwidth, as a traditional weighted scheduler would.

3.4. Queue Protection

Because of the small buffer size of the Low Latency Queue, classic TCP flows or other queue-building flows would see poor performance (due to high packet loss) if they were to end up in the Low Latency Queue. In addition, they would destroy the latency performance for the non-queue-building flows, negating the primary benefits of LLD.

To prevent this situation, the packets that are classified to the Low Latency queue pass through a "Queue Protection" function (see Figure 5 in [LLD-white-paper]), which scores each flow’s contribution to the growth of the queue. If the queue delay exceeds a threshold, the Queue Protection function identifies the flow or flows that have contributed most to the growth of the queue delay, and it redirects future packets from those flows to the Classic Service Flow. This
mechanism is performed objectively and statistically, without examining the identifiers or contents of the data being transmitted.

4. Upstream Scheduling Improvements

The DOCSIS upstream Media Access Control (MAC) Layer uses a request-grant mechanism. When data to be transmitted arrive at the CM, a request message is sent from the CM to the CMTS. The CMTS schedules the individual transmission bursts for all the CMs and communicates this via a bandwidth allocation map (MAP) message. Each MAP message describes the upstream transmission opportunities (grants) for a time interval and is sent shortly before the interval to which it applies.

When a CM has data to send, it waits for a "contention request" transmission opportunity. During that opportunity, it sends a short request message indicating the amount of data it has to send. It then waits for a subsequent MAP message granting it a transmission opportunity in which to send its data. This time interval between the arrival of the packet at the CM and the time at which the data arrives at the CMTS on the upstream channel is known as the Request-Grant Delay (see Figure 6 in [LLD-white-paper]). In the absence of queuing delay, this delay is generally 2-8 ms.

4.1. Faster Request Grant Loop

LLD lowers the request-grant delay by requiring support for a shorter MAP Interval and a shorter MAP Processing Time (see Figure 7 in [LLD-white-paper]).

The MAP interval is the amount of time that each MAP message describes. The MAP interval is also the time interval between consecutive MAP messages. Reducing the MAP interval means that the CMTS processes incoming requests more frequently, thus shortening the amount of time that a request might wait at the CMTS before being processed. A shorter MAP interval also means that grants are not scheduled as far into the future within each MAP message.

The MAP Processing Time is the amount of time the CMTS uses to perform its scheduling calculations. With a shorter MAP Processing Time, there is less delay between a request being received at the CMTS and the resulting grant being scheduled.

The LLD specification requires support for a nominal MAP interval of 1 ms or less for OFDMA upstream channels, in place of the 2-4 ms used previously. In certain configurations, a 1 ms MAP interval may introduce tradeoffs such as upstream and/or downstream inefficiency that will need to be weighed against the latency improvement.
4.2. Proactive Grant Service

DOCSIS scheduling services are designed to customize the behavior of the request-grant process for particular traffic types. LLD introduces a new scheduling service called Proactive Grant Service (PGS), which can eliminate the request-grant loop entirely (see Figure 8 in [LLD-white-paper]).

In PGS, a CMTS proactively schedules a stream of grants to a Service Flow at a rate that is intended to match or exceed the instantaneous demand. In doing so, the vast majority of packets carried by the Service Flow can be transmitted without being delayed by the Request-Grant process. During periods when the CMTS estimates no demand for bandwidth for a particular PGS Service Flow, it can conserve bandwidth by providing periodic unicast request opportunities rather than a stream of grants.

The service parameters that are specific to PGS are Guaranteed Grant Interval (GGI), Guaranteed Grant Rate (GGR), and Guaranteed Request Interval (GRI). In addition, the traditional rate-shaping parameters, such as Maximum Sustained Traffic Rate and Peak Rate, serve as an upper bound on the grants that can be provided to a PGS Service Flow.

PGS can eliminate the delay caused by the Request-Grant loop, but it comes at the price of efficiency. Inevitably, the CMTS will not be able to exactly predict the instantaneous demand for the Service Flow, so it may overestimate the capacity needed. When the shared channel is fully utilized, this could reduce the capacity available to other Service Flows.

The PGS scheduling type may appear at first to be similar to an existing DOCSIS upstream scheduling type "UGS/AD." The main differences with PGS are that it sets a minimum floor on the level of granting (minimum grant spacing and minimum granted bandwidth) rather than setting a fixed grant pattern (fixed grant size and precise grant spacing), it supports the "Continuous Concatenation and Fragmentation" method of filling grants (where a contiguous sequence of bytes are dequeued to fill the grant, regardless of packet boundaries) rather than only carrying a single packet in each grant, and the CM is expected to continue to send Requests to the CMTS to inform it of packets that might be waiting in the queue.

5. Low Latency DOCSIS Performance

CableLabs has developed a simulator using the NS3 platform (<https://www.nsnam.org>) in order to evaluate the performance of different aspects of LLD. The simulator models a DOCSIS 3.1 link
(OFDM/A channel types) between the CM and the CMTS and can be configured to enable or disable various components of the technology.

Because the latency performance of the service depends on the mix of applications in use by the customer, we have developed a set of 10 traffic mix scenarios that represent what we believe to be common busy-hour behaviors for a cable customer. All traffic mixes include two bidirectional UDP sessions that are modeled after online games, but they could also represent VoIP or video conferencing/chatting applications. One of the sessions has its packets marked as NQB and the other does not, allowing us to see the benefit that the low-latency queue provides.

In addition, each traffic mix has a set of other applications that create background load, as summarized in Table 2 (see Appendix B for details on the traffic types). All of this background load traffic utilizes the classic queue.

Some of these traffic mixes represent behaviors that may be very common for broadband users during busy hour, whereas others represent more extreme behaviors that users may occasionally engage in. When generating an overall view of the performance across all of the traffic mixes, we model the fact that they may not all be equally likely to occur by giving the more common mixes (1, 2, and 8) ten times the weight that we give to each of the other less common mixes.

| Traffic Mix 1  | 1 web user                                      |
| Traffic Mix 2  | 1 web user, 1 video streaming user              |
| Traffic Mix 3  | 1 web user, 1 FTP upstream                      |
| Traffic Mix 4  | 1 web user, 1 FTP downstream                    |
| Traffic Mix 5  | 1 web user, 1 FTP upstream and 1 FTP downstream |
| Traffic Mix 6  | 1 web user, 5 FTP upstream and 5 FTP downstream |
| Traffic Mix 7  | 1 web user, 5 FTP up, 5 FTP down, and 2 video streaming users |
| Traffic Mix 8  | 5 web users                                     |
| Traffic Mix 9  | 16 TCP down (speedtest)                         |
| Traffic Mix 10 | 8 TCP up (speedtest)                            |

Table 2

Table 3 summarizes the 99th percentile per-packet latency for the NQB-marked game traffic across all ten traffic mixes, as well as the weighted overall performance, for four different systems:
1. a legacy DOCSIS 3.1 system with AQM disabled, 2 ms MAP interval;
2. a legacy DOCSIS 3.1 system with AQM enabled, 2 ms MAP interval;
3. a Low Latency DOCSIS 3.1 system without PGS, 1 ms MAP interval;
and
4. a Low Latency DOCSIS 3.1 system with PGS configured for 5 Mbps GGR, 1 ms MAP interval.

We include LLD with and without PGS because some network operators may wish to deploy LLD without the overhead that comes with PGS scheduling.

**TABLE 3. 99TH PERCENTILE ROUND-TRIP LATENCY FOR NQB-MARKED TRAFFIC BETWEEN THE CM AND CMTS**

<table>
<thead>
<tr>
<th>Traffic</th>
<th>Legacy DOCSIS 3.1 with no AQM</th>
<th>Legacy DOCSIS 3.1 with AQM</th>
<th>Low Latency DOCSIS with no PGS</th>
<th>Low Latency DOCSIS with PGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mix 1</td>
<td>7.7 ms</td>
<td>7.7 ms</td>
<td>4.7 ms</td>
<td>0.9 ms</td>
</tr>
<tr>
<td>Mix 2</td>
<td>7.7 ms</td>
<td>7.7 ms</td>
<td>4.8 ms</td>
<td>0.9 ms</td>
</tr>
<tr>
<td>Mix 3</td>
<td>159.5 ms</td>
<td>36.6 ms</td>
<td>4.7 ms</td>
<td>0.9 ms</td>
</tr>
<tr>
<td>Mix 4</td>
<td>7.8 ms</td>
<td>7.9 ms</td>
<td>4.7 ms</td>
<td>0.9 ms</td>
</tr>
<tr>
<td>Mix 5</td>
<td>159.6 ms</td>
<td>57.4 ms</td>
<td>4.7 ms</td>
<td>0.9 ms</td>
</tr>
<tr>
<td>Mix 6</td>
<td>253.7 ms</td>
<td>96.7 ms</td>
<td>4.7 ms</td>
<td>0.9 ms</td>
</tr>
<tr>
<td>Mix 7</td>
<td>253.9 ms</td>
<td>74.7 ms</td>
<td>4.7 ms</td>
<td>0.9 ms</td>
</tr>
<tr>
<td>Mix 8</td>
<td>7.7 ms</td>
<td>7.7 ms</td>
<td>4.7 ms</td>
<td>0.9 ms</td>
</tr>
<tr>
<td>Mix 9</td>
<td>259.3 ms</td>
<td>52.1 ms</td>
<td>4.8 ms</td>
<td>0.9 ms</td>
</tr>
<tr>
<td>Mix 10</td>
<td>254.0 ms</td>
<td>34.1 ms</td>
<td>4.8 ms</td>
<td>0.9 ms</td>
</tr>
<tr>
<td>Weighted Overall</td>
<td></td>
<td></td>
<td>4.7 ms</td>
<td>0.9 ms</td>
</tr>
<tr>
<td>P99</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3
As can be seen in this table, there are several traffic mixes (notably 1, 2, 4, and 8) for which the relatively light traffic load doesn’t create the conditions for TCP to cause significant queuing delay, so even the "Legacy DOCSIS 3.1 with no AQM" system results in fairly low latency. However, in the heavier traffic mixes, the benefit of AQM can be seen and the benefit of the dual-queue mechanism in LLD becomes very apparent. By separating the NQB-marked traffic from the queue-building traffic, the NQB-marked traffic is isolated from the delay created by the TCP flows entirely, and very reliable low latency is achieved. The right-most system, which additionally implements PGS, can eliminate the request-grant delay for the NQB traffic and thereby drive the round-trip latency below 1 ms at 99th percentile.

Figure 9 in [LLD-white-paper] illustrates the weighted overall latency performance across all ten traffic mixes. The plot is a log-log complementary cumulative distribution function, with the y-axis labeled with the equivalent quantile values.

Focusing, for instance, on the horizontal through the 99th percentile (P99), it can be seen that LLD with PGS holds delay below 0.9 ms for 99% of packets. In contrast, a DOCSIS 3.1 network without AQM can only hold delay below 250 ms for 99% of packets. So, P99 delay is more than 250 times better with LLD. We therefore see that LLD will bring a consistent, low-latency, responsive quality to cable broadband performance and user experiences for NBQ traffic.

6. Deployment Considerations

6.1. Device Support

Deploying LLD in the MSO network can be accomplished via software-only upgrades to the existing DOCSIS 3.1 CMs and CMTSs. Table 4 shows which LLD features need implementation on the CM side, the CMTS side, or both. The Dual Queue feature in the upstream requires an upgrade to the CM as well as to the CMTS. The other features (Dual Queue in Downstream, Upstream Scheduling improvements) only require upgrades on the CMTS, so they can be deployed to CMs that don’t support LLD (including DOCSIS 3.0 modems).
6.2. Packet Marking

The design of LLD takes the approach that applications are in the best position to determine which flows or which packets are non-queue-building. Thus, applications such as online games will be able to tag their packets with the NQB DiffServ value to indicate that they behave in a non-queue-building way, so that LLD will be able to classify them into the Low Latency Service Flow.

For these packet markings to be useful for the LLD classifiers, they will need to survive the journey from the application source to the CM or CMTS. In some cases, operators today clear the DiffServ Field in packets entering their network from an interconnecting network, which would prevent the markings making their way to the CMTS. This practice is presumably driven by the view that DiffServ Field usage is defined by each operator for use within its network, in which case preserving another network’s markings has no value. As was described in Section 3.2, it is proposed that a single globally standard value be chosen to indicate NQB so that operators that intend to support LLD can ensure that this specific value traverses their inbound interconnects and their network and then arrives at the CMTS intact.

Although application marking is preferable, some network operators might want to provide immediate benefits to applications that behave in a non-queue-building way, in advance of application developers introducing support for NQB tagging. It might be possible to...
repurpose the queue protection function to identify NQB behavior even if the packets are not tagged as NQB, e.g., by assuming that all non-TCP traffic is likely to be NQB and relying on queue protection to redirect the QB flows. This is currently an area of active research.

Further, it is possible that intermediary software or devices (either installed by the user or provided by the operator) could identify flows that are expected to be NQB and mark the packets on behalf of the application.

6.3. Provisioning Mechanisms

The LLD specifications include provisioning mechanisms to allow an MSO to deploy low-latency features with minimal operational impact. Figure 10 in [LLD-white-paper] shows all the pieces needed to build a low-latency service in the upstream and downstream direction. Although it is possible to define a Low Latency ASF, its constituent Classic and Low Latency SFs, and the associated classifiers explicitly in the CM’s configuration file, a new feature known as the Aggregate QoS Profile can make this configuration automatic in many cases. Default classifiers will be created and default parameters for AQM and queue protection will be used, or any of these can be overridden by the operator as needed.

6.3.1. Aggregate QoS Profiles

Similar to Service Class Names that are expanded by the CMTS into a set of QoS parameters for a Service Flow during the registration process, an operator can create an Aggregate QoS Profile (AQP) on the CMTS to describe the parameters of an Aggregate Service Flow, its constituent Service Flows, and the classifiers used to identify NQB traffic.

Just like with Service Class Names, the operator can also provide explicit values in the configuration file for any ASF or SF parameters that they wish to "override".

6.3.2. Migration Using Existing Configuration File and Service Class Name

One very straightforward way to migrate to LLD configurations may not involve any changes to the CM configuration file. This method involves the automatic expansion of a Service Flow definition to a Low Latency ASF via the use of a Service Class Name and matching AQP definition.

When the CMTS sees a Service Class Name in a Service Flow definition from the CM’s config file, if the CM indicates support for LLD, then
the CMTS will first use the Service Class Name as an AQP Name and look for a matching entry in the AQP Table. If it finds a matching entry, it will automatically expand the Service Flow into an ASF and two Service Flows.

This mechanism allows the operator to deploy LLD by simply updating the CMTS to support the feature and configuring AQP entries that match the Service Class Names in use in CM config files. Then, as CMs are updated over time to include support for LLD, they will automatically start being configured with a Low Latency ASF.

6.3.3. Explicit Definition of ASF in the Configuration File

An operator can also encode a Low Latency ASF in a CM configuration file directly using an Aggregate Service Flow TLV (70 or 71). The ASF TLV could have an AQP Name that is used by the CMTS to look up a definition of the ASF in its AQP Table. It could also have ASF parameters that would explicitly define the ASF or would override the AQP parameters. A configuration could also have explicit individual Service Flow TLVs (24 or 25) that are linked to the ASF via the Aggregate Service Flow Reference TLV.

6.4. Latency Histogram Reporting

As part of the AQM operation, CMs and CMTSs generate estimates of the queuing latency for the upstream and downstream Service Flows, respectively. The latency histogram reporting function exposes these estimates to the operator to provide information that can be utilized to characterize network performance, optimize configurations, or troubleshoot problems in the field.

This latency histogram reporting can be enabled via a configuration file setting or can be initiated by setting a MIB object on the device. The operator configures the bins of the histogram, and the CM or the CMTS logs the number of packets with recorded latencies into each of the bins. The CM implements histograms for upstream Service Flows, and the CMTS implements histograms for downstream Service Flows. (This function can be enabled even for Service Flows for which AQM is disabled.) The latency estimates from the AQM are represented in the form of a histogram as well as a maximum latency value. See Figure 11 in [LLD-white-paper].

7. Conclusion

LLD enables a huge leap in latency performance and will improve the Internet experience overall. With LLD, online gaming will become more responsive and video chats will cease to be "choppy." This technology will enable a range of new applications that require real-
time interface between the cyber and physical worlds, such as vehicular communications and remote health care services.

To realize the benefits of LLD, a number of parties need to take action. DOCSIS equipment manufacturers will need to develop and integrate the LLD features into software updates for CMTSs and CMs. Cable operators need to plan the roll-out of software updates and configurations to DOCSIS equipment and set up the network to support those services (e.g., carrying DiffServ/ECN markings through the network). Application and operating system vendors will need to adopt packet marking for NQB traffic and/or adopt the L4S congestion controller. Each element of the Internet ecosystem will make these decisions independently; the faster that all take the necessary steps, the more quickly the user experience will improve.

The cable industry has provisioned its network with substantial bandwidth and is poised to take another leap forward with its 10G networks. But more bandwidth is only part of the broadband performance story. Latency is becoming crucial to the evolution of broadband. That is why LLD is a cornerstone of cable’s 10G future.

8. Acknowledgements

CableLabs would like to thank the participants of the Low Latency DOCSIS Working Group, representing ARRIS, Broadcom, Casa, Charter, Cisco, Comcast, Cox Communications, Huawei, Intel, Liberty Global, Nokia, Rogers, Shaw, Videotron

9. IANA Considerations

None

10. Security Considerations

TBD

11. Informative References

[DOCSIS-CCAP-OSGiV3.1]
[DOCSIS-CM-OSSIv3.1]

[DOCSIS-MULPIv3.1]

[I-D.ietf-tsvwg-aqm-dualq-coupled]
Schepper, K., Briscoe, B., Bondarenko, O., and I. Tsang, "DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)", draft-ietf-tsvwg-aqm-dualq-coupled-08 (work in progress), November 2018.

[I-D.ietf-tsvwg-ecn-l4s-id]

[I-D.ietf-tsvwg-l4s-arch]

[I-D.white-tsvwg-nqb]
White, G., "Identifying and Handling Non Queue Building Flows in a Bottleneck Link", draft-white-tsvwg-nqb-00 (work in progress), October 2018.

[LLD-white-paper]

L4S involves incremental changes to the congestion controller on the sender and to the AQM at the bottleneck. The key is to indicate congestion by marking packets using Explicit Congestion Notification (ECN) rather than discarding packets. L4S uses the 2-bit ECN field in the IP header (v4 or v6) and defines each marked packet to represent a lower strength of congestion signal [RFC8311] than the original ECN standard. All the benefits of L4S follow from that.

- Low Latency: The sender’s L4S congestion controller makes small but frequent rate adjustments dependent on the proportion of ECN marked packets, and the L4S AQM starts applying ECN-marks to packets at a very shallow buffer threshold. This means an L4S queue can ripple at the very bottom of the buffer with sub-millisecond queuing delay but still fully utilize the link. Small, frequent adjustments could not even be considered if packet discards were used instead of ECN — they would induce a prohibitively high loss level. Further, AQMs could not consider a
very shallow threshold if small adjustments were not used, as severe link under-utilization would result.

- **Low Loss**: By definition, using ECN eliminates packet discard. In turn, that eliminates retransmission delays, which particularly impact the responsiveness of short web-like exchanges of data. Using ECN eliminates both the round-trip delay repairing a loss and the delay while detecting a loss. In addition, an L4S AQM can immediately signal queue growth using ECN, catching queue growth early. In contrast, classic AQMs hold back from discarding a packet for 100–200 ms because if a burst subsides of its own accord, a loss in itself could cause more harm than the good it would do as a signal to slow down. Furthermore, eliminating packet discard eliminates the collateral damage caused to flows that were not significantly contributing to congestion.

- **Scalable Throughput**: Existing congestion control algorithms don’t scale, so applications need to open many simultaneous connections to fully utilize today's broadband connections. An L4S congestion controller can rapidly ramp up its sending rate to match any link capacity. This is because L4S uses a “scalable congestion controller” that maintains the same frequency of control signals (2 ECN marks per round trip on average) regardless of flow rate. With classic congestion controllers, the faster they try to go, the longer they run blind without any control signals.

The technology behind L4S isn’t new; it is based on a scalable congestion control called Data Center TCP (DCTCP) that is currently used in data centers to get very high throughputs with ultra-low delay and loss. What is new is the development of a way that scalable traffic can coexist with the existing TCP and QUIC traffic on the Internet - the key that unlocks a transition to L4S. Until now, DCTCP has been confined to data centers because it would starve any classic flows sharing a link.

Separation into two queues serves two purposes: (1) it isolates L4S flows from the queuing of classic TCP and QUIC and (2) it sends each type of traffic appropriately scaled congestion signals. This results in any number of application flows (of either type) all getting roughly equal bandwidth each, as if there were just one aggregate pool of bandwidth, with no division between the Service Flows.

The approach couples the levels of ECN and drop signaling, as shown in Figure 12 in [LLD-white-paper]. The packet rate of today’s classic congestion controls conforms to the well-known square-root rule (on the left of the figure). So, the classic AQM applies a drop level to Classic traffic that is coupled to the square of the ECN.
marking level being applied to Low Latency traffic. The squaring in the network counterbalances the square root at the sender, so the packet rates of the two types of flow turn out roughly the same.

Supporting L4S in LLD is relatively straightforward. All that is needed is to classify L4S flows into the Low Latency SF and support the logic in the Low Latency SF to perform immediate ECN marking of packets (see Section 3.2).

Appendix B. Simulation Details

For the results reported in this paper, we set up the following network with 5 types of client devices behind the CM and a set of servers north of the CMTS. See Figure 13 in [LLD-white-paper]. The link delays shown are 1-way values. The DOCSIS link is configured in the most latency-efficient manner (short interleavers, small OFDMA frame sizes) and models a plant distance of 8 km. The service is configured with a Maximum Sustained Traffic Rate (rate limit) of 50 Mbps in the upstream direction and 200 Mbps in the downstream direction.

The upstream game traffic model involves normally distributed packet interarrival times (\(\mu=33\) ms, \(\sigma=3\) ms) and normally distributed packet sizes (\(\mu=110\) bytes, \(\sigma=20\) bytes) constrained to discard draws of packet size <32 bytes or >188 bytes. The downstream game traffic model involves normally distributed packet interarrival times (\(\mu=33\) ms, \(\sigma=5\) ms) and normally distributed packet sizes (\(\mu=432\) bytes, \(\sigma=20\) bytes) constrained to discard draws of packet size <32 bytes or >832 bytes.

The background load traffic is configured as follows. The web user is based on the 3GPP standardized web user model [web-user-model]. The video streaming model is an abstracted model of a Dynamic Adaptive Streaming over HTTP (DASH) streaming video user where the video stream is 6 Mbps and is implemented as a 3.75 MB file download every 5 seconds. Each FTP session involves the sender selecting a file size using a log-normal random variable (\(\mu=14.8\), \(\sigma=2.0\), leading to a median file size of 2.7 MB), opening a TCP connection, sending the file, closing the TCP connection, then pausing for 100 ms before repeating the process. Although we refer to this model as an FTP model, the intention is that it models TCP usage across all applications other than web browsing and video streaming.

Authors’ Addresses
Greg White
CableLabs
858 Coal Creek Circle
Louisville, CO 80027
US
Email: g.white@cablelabs.com

Karthik Sundaresan
CableLabs
858 Coal Creek Circle
Louisville, CO 80027
US
Email: k.sundaresan@cablelabs.com

Bob Briscoe
CableLabs
UK
Email: b.briscoe-contractor@cablelabs.com
Identifying and Handling Non Queue Building Flows in a Bottleneck Link

draft-white-tsvwg-nqb-01

Abstract

This draft proposes the definition of a standardized DSCP to identify Non-Queue-Building flows, along with a Per-Hop-Behavior (PHB) that provides a separate queue for such flows.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 12, 2019.

Copyright Notice

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

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

Residential broadband internet services are commonly configured with a single bottleneck link (the access network link) upon which the service definition is applied. The service definition, typically an upstream/downstream data rate tuple, is implemented as a configured pair of rate shapers that are applied to the user’s traffic. In such networks, the quality of service that each application receives, and as a result, the quality of experience that it generates for the user is influenced by the characteristics of the access network link.

The vast majority of packets that are carried by residential broadband access networks are managed by an end-to-end congestion control algorithm, such as Reno, Cubic or BBR. These congestion control algorithms attempt to seek the available capacity of the end-to-end path (which in the case of residential broadband networks, can frequently be the access network link capacity), and in doing so generally overshoot the available capacity, causing a queue to build-up at the bottleneck link. This queue build up results in queuing delay that the application experiences as variable latency, and commonly results in packet loss as well.

In contrast to congestion-controlled applications, there are a variety of relatively low data rate applications that do not materially contribute to queueing delay, but are nonetheless subjected to it by sharing the same bottleneck link in the access network. Many of these applications may be sensitive to latency or latency variation, as well as packet loss, and thus produce a poor quality of experience in such conditions.

Active Queue Management (AQM) mechanisms (such as PIE [RFC8033], DOCSIS-PIE [RFC8034], or CoDel [RFC8289]) can improve the quality of
experience for latency sensitive applications, but there are practical limits to the amount of improvement that can be achieved without impacting the throughput of capacity-seeking applications.

This document considers differentiating between these two classes of traffic in bottleneck links in order that both classes can deliver exceptional quality of experience for their applications.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Non-Queue Building Flows

There are many applications that send traffic at relatively low data rates and/or in a fairly smooth and consistent manner such that they are highly unlikely to exceed the available capacity of the network path between source and sink. Such applications are ideal candidates to be queued separately from the capacity-seeking applications that cause queue buildups and latency.

These Non-queue-building (NQB) flows are typically UDP flows, which send traffic at a lower data rate and don’t seek the capacity of the link (examples: online games, voice chat, dns lookups). Here the data rate is essentially limited by the Application itself. In contrast, Queue-building (QB) flows include traffic which uses the Traditional TCP, QUIC, BBR or other TCP variants.

There are a lot of great examples of applications that fall very neatly into these two categories, but there are also application flows that may be in a gray area in between (e.g. they are NQB on high-speed links, but QB on low-speed links).

4. Endpoint Marking and Queue Protection

This memo proposes that application endpoints apply a marking, utilizing the Diffserv field of the IP header, to packets of NQB flows that could then be used by the network to differentiate between QB and NQB flows. It is important for such a marking to be universally agreed upon, rather than being locally defined by the network operator, such that applications could be written to apply the marking without regard to local network policies.

Some questions that arise when considering endpoint marking are: How can an application determine whether it is queue building or not, given that the sending application is generally not aware of the
available capacity of the path to the receiving endpoint? Even in cases where an application is aware of the capacity of the path, how can it be sure that the available capacity (considering other flows that may be sharing the path) would be sufficient to result in the application’s traffic not causing a queue to form? In an unmanaged environment, how can networks trust endpoint marking, why wouldn’t all applications mark their packets as NQB?

As an answer the last question, it would be worthwhile to note that the NQB designation and marking would be intended to convey verifiable traffic behavior, not needs or wants. Also, it would be important that incentives are aligned correctly, i.e. that there is a benefit to the application in marking its packets correctly, and no benefit for an application in intentionally mismarking its traffic. Thus, a useful property of nodes that support separate queues for NQB and QB flows would be that for NQB flows, the NQB queue provides better performance (considering latency, loss and throughput) than the QB queue; and for QB flows, the QB queue provides better performance (considering latency, loss and throughput) than the NQB queue.

Even so, it is possible that due to an implementation error or misconfiguration, a QB flow would end up getting mismarked as NQB, or vice versa. In the case of an NQB flow that isn’t marked as NQB and ends up in the QB queue, it would only impact its own quality of service, and so it seems to be of lesser concern. However, a QB flow that is mismarked as NQB would cause queuing delays for all of the other flows that are sharing the NQB queue.

To prevent this situation from harming the performance of the real NQB flows, it would likely be valuable to support a "queue protection" function that could identify QB flows that are mismarked as NQB, and reclassify those flows/packets to the QB queue. This would benefit the reclassified flow by giving it access to a large buffer (and thus lower packet loss rate), and would benefit the actual NQB flows by preventing harm (increased latency variability) to them. Such a function should be implemented in an objective and verifiable manner, basing its decisions upon the behavior of the flow rather than on application-level constructs.

5. Non Queue Building PHB and DSCP

This section uses the DiffServ nomenclature of per-hop-behavior (PHB) to describe how a network node could provide better quality of service for NQB flows without reducing performance of QB flows.

A node supporting the NQB PHB MUST provide a queue for non-queue-building traffic separate from the queue used for queue-building
traffic. This queue SHOULD support a latency-based queue protection mechanism that is able to identify queue-building behavior in flows that are classified into the queue, and to redirect flows causing queue build-up to a different queue. One example algorithm can be found in Annex P of [DOCSIS-MULPIv3.1].

While there may be some similarities between the characteristics of NQB flows and flows marked with the Expedited Forwarding DSCP, the NQB PHB would differ from the Expedited Forwarding PHB in several important ways.

- 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 sub-millisecond queuing delays 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 (101110b) in the NQB queue. The choice of the 0x2A codepoint (101010b) for NQB would conveniently allow a node to select these two codepoints using a single mask pattern of 101x10b.

Additionally, WiFi routers and APs that support WiFi MultiMedia commonly use the upper three bits of the DiffServ Field to select the WiFi User Priority. In the case of the 0x2A codepoint, this would map to UP_5 which is in the "Video" Access Category (one level above "Best Effort").

6. End-to-end Support

In contrast to the existing standard DSCPs, which are typically only enforced within a DiffServ Domain (e.g. an AS), this DSCP would be intended for end-to-end usage across the Internet. Some access network service providers bleach the DiffServ field on ingress into their network, and in some cases apply their own DSCP for internal usage. Access networks that support the NQB PHB would need to permit the NQB PHB to pass through this bleaching operation such that the PHB can be provided at the access network link.
7. Relationship to L4S

The dual-queue mechanism described in this draft is similar to, and is intended to be compatible with [I-D.ietf-tsvwg-l4s-arch].

8. Comparison to Existing Approaches

Traditional QoS mechanisms focus on prioritization in an attempt to achieve two goals, reduced latency for "latency-sensitive" traffic, and increased bandwidth availability for "important" applications. Applications are generally given priority in proportion to some combination of latency-sensitivity and importance.

Downsides to this approach include the difficulties in sorting out what priority level each application should get (making the value judgement as to latency-sensitivity and importance), associating packets to priority levels (lots of classifier state, or trusting endpoint markings and the value judgements that they convey), ensuring that high priority traffic doesn’t starve lower priority traffic (admission control, weighted scheduling, etc. are possible solutions). This solution can work in a managed network, where the network operator can control the usage of the QoS mechanisms, but has not been adopted end-to-end across the internet.

Flow queueing approaches (such as fq_codel [RFC8290]), on the other hand, achieve latency improvements by associating packets into "flow" queues and then prioritizing "sparse flows", i.e. packets that arrive to an empty flow queue. Flow queueing does not attempt to differentiate between flows on the basis of value (importance or latency-sensitivity), it simply gives preference to sparse flows, and tries to guarantee that the non-sparse flows all get an equal share of the remaining channel capacity and are interleaved with one another. As a result, fq mechanisms could be considered more appropriate for unmanaged environments and general internet traffic.

Downsides to this approach include loss of low latency performance due to the possibility of hash collisions (where a sparse flow shares a queue with a bulk data flow), complexity in managing a large number of queues in certain implementations, and the DRR scheduling, which enforces that each non-sparse flow gets an equal fraction of link bandwidth, causes problems with VPNs and other tunnels, exhibits poor behavior with less-aggressive CA algos, e.g. LEDBAT, and exhibits poor behavior with RMCAT CA algos. In effect the network element is making a decision as to what constitutes a flow, and then forcing all such flows to take equal bandwidth at every instant.

The Dual-queue approach achieves the main benefit of fq_codel: latency improvement without value judgements, without the downsides.
The distinction between NQB flows and QB flows is similar to the distinction made between "sparse flow queues" and "non-sparse flow queues" in fq_codel. In fq_codel, a flow queue is considered sparse if it is drained completely by each packet transmission, and remains empty for at least one cycle of the round robin over the active flows (this is approximately equivalent to saying that it utilizes less than its fair share of capacity). While this definition is convenient to implement in fq_codel, it isn’t the only useful definition of sparse flows.

The Linux Heavy-Hitter Filter ([HHF]) qdisc and the Cisco Dynamic Packet Prioritization ([DPP]) feature both categorize application flows into "mice" and "elephants", and provide a separate queue that gives high priority to the "mice" flows. In both of these implementations, the definition of a mouse flow is one that falls below a defined number of bytes or packets (respectively). In essence, the first N bytes or packets of every new flow are queued separately, and given priority over other traffic. The HHF implementation defaults to using 128KB for N, whereas the DPP documentation discusses using 120 packets.

This approach is relatively simple to implement, but it is making the wrong distinction between flows. To illustrate, an hour-long 60 kbps multiplayer online gaming flow sending 60 packets per second would be classified as an elephant after the first 17 seconds using HHF or 2 seconds using DPP, whereas it should be considered as NQB for the entire duration.

9. Acknowledgements

TBD

10. IANA Considerations

This draft proposes the registration of a standardized DSCP = 0x2A to denote Non-Queue-Building behavior.

11. Security Considerations

There is no incentive for an application to mismark its packets as NQB (or vice versa). If a queue-building flow were to mark its packets as NQB, it could experience excessive packet loss (in the case that queue-protection is not supported by a node) or it could receive no benefit (in the case that queue-protection is supported). If a non-queue-building flow were to fail to mark its packets as NQB, it could suffer the latency and loss typical of sharing a queue with capacity seeking traffic.
The NQB signal is not integrity protected and could be flipped by an on-path attacker. This might negatively affect the QoS of the tampered flow.

12. Informative References

[DOCSIS-MULPIv3.1]  

[DPP]  

[HHF]  

[I-D.ietf-tsvwg-l4s-arch]  

[RFC2119]  

[RFC8033]  

[RFC8034]  


Author’s Address

Greg White
CableLabs
858 Coal Creek Circle
Louisville, CO 80027
US

Email: g.white@cablelabs.com