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

Abstract

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

Status of This Memo

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Explicit congestion notification (ECN [RFC3168]) allows a forwarding element to notify downstream devices of the onset of congestion without having to drop packets. Coupled with a means to feedback 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 shows an SFC domain for the purpose of illustrating the use of NSH. Traffic passes through a sequence of Service Function Forwarders (SFFs) each of which sends the traffic to one or more Service Functions (SFs). Each SF performs some operation on the traffic, for example firewall or Network Address Translation (NAT), and then returns it to the SFF from which it was received.

Logically, during the transit of each SFF, the outer transport header that got the packet to the SFF is stripped, the SFF decides on the next forwarding step, either adding a transport header or, if the SFF is the exit/egress, removing the NSH header. The transport headers
added may be different in different regions of the SFC domain. For example, IP could be used for some SFF-to-SFF communication and MPLS used for other such communication.

1.2 ECN Background

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

1.3 Tunnel Congestion Feedback Background

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

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

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

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

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

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

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.

```
+-----------------------------------+
| Transport Encapsulation           |
+-----------------------------------+
| Network Service Header (NSH)      |
|   | Base Header                    |
|   | Service Path Header            |
|   | Metadata (Context Header(s))   |
| +------------------------------+  |
| Original Packet / Frame          |
+-----------------------------------+
```

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 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

^ ^

| NSH ECN|
field|
```

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 3. Exit ECN Fields Merger

<table>
<thead>
<tr>
<th>Arriving</th>
<th>Arriving Outer Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inner Header</td>
<td>Not-ECT</td>
</tr>
<tr>
<td>Not-ECT</td>
<td>Not-ECT</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>

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

### 3.4 Conservation of Packets

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

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

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

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

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

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

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

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

Table of Contents

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

1. Introduction

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

The current specification for UDP permits sending datagrams with surplus data, but are not commonly observed, and many network devices assume that IP payload length is equal to UDP length and have used this value when calculating UDP checksums. This leads to the case where some middlebox devices (e.g. Firewalls, 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 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 to 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>
<tr>
<td>---</td>
<td></td>
</tr>
<tr>
<td>Sum:</td>
<td>d6d0</td>
</tr>
<tr>
<td>CCO:</td>
<td>292f</td>
</tr>
</tbody>
</table>

Figure 5: Checksum calculations

In the second example the UDP Payload length is odd and a MSS Option has been already placed in UDP Options area. The available space for CCO starts at an odd byte (NOP padding before CCO) and also UDP options space starts at odd byte (left zero padding of first byte). CCO value is initialized with UDP Options Length (0x0009).
UDP Length: Odd
Preceding UDP Options: MSS (kind 5, len 4, val 0xc0)
Following UDP Options: None

NOP Padding before CCO: Yes
Total UDP Options Length: 9

UDP Options bytes 0: 0005
UDP Options bytes 1/2: 0405
UDP Options bytes 3/4: c001
UDP Options bytes 5/6: cc04
UDP Options bytes 7/8: 0009

Sum: 9019

CCO: 6fe6

Figure 6: Checksum calculations

4.4. Interaction with other UDP Options

Interaction with other UDP Options

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

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

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

5. Acknowledgements

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

6. IANA Considerations

This memo includes no requests to IANA
7. Security Considerations

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

8. Normative References

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


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
Abstract

IP is designed for use in Best Effort Networks, which are networks that provide no guarantee that data is delivered, or that delivery meets any specified quality of service parameters. However, there are new applications requiring IP to provide deterministic services in terms of bandwidth and latency, such as network-based AR/VR (Augmented Reality and Virtual Reality), industrial internet. This document proposes a solution in IPv6 that can be used by transport layer protocols to guarantee certain levels of service quality. This new service is fine-grained and could apply to individual or aggregated TCP/UDP flow(s).

Status of This Memo

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

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

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

This Internet-Draft will expire on October 19, 2019.
Copyright Notice

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

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

Table of Contents

1. Introduction .................................................. 3
2. Terminology .................................................. 4
3. Overview ..................................................... 6
   3.1. Design Targets ............................................ 6
   3.2. Scope and Assumptions .................................... 7
   3.3. Sub-layer in IP for Transport Control ..................... 7
   3.4. IP In-band signaling ....................................... 8
   3.5. IPv6 Approach ............................................. 10
4. Key Messages and Parameters ................................... 11
   4.1. Setup and Setup State Report messages .................... 11
   4.2. Forwarding State and Forwarding State Report messages 12
   4.3. Hop Number ................................................. 13
   4.4. Flow Identifying Method and Service ID ................... 13
   4.5. QoS State and life of Time ................................ 14
   4.6. Authentication ............................................... 14
5. Packet Forwarding ............................................. 15
   5.1. Basic Hardware Capability .................................. 15
   5.2. Flow Identification in Packet Forwarding .................. 16
   5.3. QoS Forwarding State Detection and Failure Handling ..... 16
6. Details of Working with Transport Layer ...................... 17
   6.1. Working with TCP .......................................... 17
   6.2. Working with UDP and other Protocols ..................... 20
7. Additional Considerations ..................................... 20
   7.1. User and Application driven ................................ 20
   7.2. Traffic Management in Host ................................ 21
   7.3. Heterogeneous Network ..................................... 22
   7.4. Proxy Control ............................................... 22
8. IANA Considerations ............................................. 22
9. Security Considerations ........................................ 24
10. References .................................................. 25
    10.1. Normative References .................................... 25
10.2. Informative References .................................................. 26
Appendix A. Acknowledgements ............................................... 28
Appendix B. Message Objects .................................................. 29
  B.1. Setup State Object ...................................................... 29
  B.2. Bandwidth Object .................................................... 31
  B.3. Burst Msg ............................................................ 31
  B.4. Latency Object ....................................................... 32
  B.5. Authentication Object ................................................ 32
  B.6. OAM Object .......................................................... 33
  B.7. Forwarding State Object ............................................. 34
  B.8. Setup State Report Object ......................................... 34
Authors’ Addresses ............................................................. 36

1. Introduction

Recently, more and more new applications for The Internet are
emerging. These applications have a number of key requirements that
are common to all such as that their required bandwidth is very high
and/or latency is very low compared with traditional applications
like most of web and video applications.

For example, network based Augmented Reality (AR) or Virtual Reality
(VR) applications may need hundreds of Mbps bandwidth (throughput)
and a low single digit millisecond latency. Moreover, the difference
between mean bit rate and peak bit rate may be significant due to the
choice of compression algorithm [I-D.han-iccrg-arvr-transport-problem]. This may result in large
bursts, and make traffic management more difficult.

Some future applications may expect networks to provide a bounded
latency service. One such example is tactile network [Tactile].

With the technology development in 5G [HU5G][QU2016] and beyond, the
wireless access network is also increasing the demand for the Ultra-
Reliable and Low-Latency Communications (URLLC). This also leads to
the question of whether IP can provide such service in an Evolved
Packet Core (EPC)[EPC] network. IP is becoming more and more
important in the EPC when the Multi-access Edge Computing (MEC)[MEC]
for 5G requires the cloud and data service to move closer to eNodeB
[eNodeB].

[I-D.ietf-detnet-use-cases] identifies some use cases from different
industries which have a common need for "deterministic flows". Such
flows require guaranteed bandwidth and bounded latency.

Traditionally, an IP network provides an unreliable or best-effort
datagram service over a collection of underlying networks (i.e.:
ethernet, ATM, etc...). Integrated services (IntServ) [RFC3175] specifies a fine-grained QoS system, which requires all routers along the traffic path to support it and maintain the states for resource reserved IP flow(s), so it is difficult to scale up to keep track of all the reservations. Differentiated services (DiffServ) [RFC2475] specifies a simple and scalable mechanism to classify traffic and provide more coarse QoS, however because it can only specify per-hop behaviors (PHBs), and how individual routers deal with the DS [RFC2474] field is configuration specific. It is difficult to provide consistent resource reservation for specified class of traffic, thus hard to support the end-to-end bandwidth or latency guarantee.

The transport layer (TCP/UDP) on top of IP is based on the best-effort-only service, which has influenced the transport layer evolution for quite long time, and results in some widely accepted assumptions and solutions, such as:

1. The IP layer can only provide basic P2P (point to point) or P2MP (point to multi-point) end-to-end connectivity in the Internet, but the connectivity is not reliable and does not guarantee any quality of service to end-user or application, such as bandwidth, packet loss, latency etc. Due to this assumption, the transport layer or application must have its own control mechanism in congestion and flow to obtain the reliable and satisfactory service to cooperate with the under layer network quality.

2. The transport layer assumes that the IP layer can only process all IP flows equally in the hardware since the best effort service is actually an un-differentiated service. The process includes scheduling, queuing and forwarding. Thus, the transport layer must behave nicely and friendly to make sure all flows will only obtain its own faired share of resource, and no one could consume more and no one could be starved.

This document proposes a new IP transport service that guarantees bandwidth and latency for new applications. The scope and criteria for the new technology will also be discussed. This new IP transport service is designed to be supplementary to regular IP transport services, only meant to be used for special applications that are bandwidth and/or latency sensitive.

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...
RFC 2119 [RFC2119] RFC 8174 [RFC8174] when, and only when, they appear in all capitals, as shown here.

Abbreviations used in this documents:

E2E
   End-to-end.

EH
   IPv6 Extension Header or Extension Option.

QoS
   Quality of Service.

OAM
   Operation and Management.

In-band Signaling
   In telecommunications, in-band signaling is sending control information within the same band or channel used for voice or video.

Out-of-band Signaling
   out-of-band signaling is that the control information sent over a different channel, or even over a separate network.

IP flow
   For non-IPSec, an IP flow is identified by the source, destination IP address, the protocol number, the source and destination port number.

IP path
   An IP path is the route that IP flow will traverse. It could be the shortest path determined by routing protocols (IGP or BGP), or the explicit path such as segment routing [I-D.ietf-spring-segment-routing].

QoS channel
   A forwarding channel that is QoS guaranteed. It provides additional QoS service to IP forwarding. A QoS channel can be used for one or multiple IP flows depending on the granularity of in-band signaling.

CIR
   Committed Information Rate.

PIR
   Peak Information Rate.
3. Overview

Semiconductor chip technology has advanced significantly in the last decade, and as such the widely used network processing and forwarding process can now not only forward packets at line speed, but also easily support other feature processing such as QoS for DiffServ/MPLS, Access Control List (ACL), firewall, and Deep Packet Inspection (DPI).

This advancement enables network processors to do the general process to handle simple control messages for traffic management, such as signaling for hardware programming, congestion state report, OAM, etc. So now it’s possible to treat some TCP/IP flows differently from others and give them specified resource are feasible now by using network processor.

This document proposes a deterministic IP transport service, which can provide guaranteed bandwidth and latency. The solution is based on the QoS implemented in network processor through in-band signaling.

3.1. Design Targets

The proposed transport service is expected to satisfy the following criteria:

- End user or application may directly use the new service.
- The new service can coexist with the current transport service and is backward compatible.
- Service providers can manage the new service.
- Performance and scalability targets of this new service are practical for vendors to achieve.
- The new service is transport agnostic. TCP, UDP and other transport protocols on top of IP can use it.
3.2. Scope and Assumptions

The initial aim is to propose a solution for IPv6. To limit the scope of the document and simplify the design and solution, the following constraints are given:

1. The new service with QoS is aimed to be supplementary to regular IP service. It is targeted for the applications that are bandwidth and/or latency sensitive. It is not intended to replace the TCP/IP variants that have been proved to be efficient and successful for current applications.

2. The new service is limited within one administrative domain, even it does not exclude the possibilities of extending the mechanism for inter-domain scenarios. Currently only inter-domain security is considered, and the inter-domain SLA, accounting and other issues are not discussed.

3. Due to high bandwidth requirement of new service for individual flow, the total number of the flows with the new service cannot be high for a port, or a system. From another point of view, the new service is targeted for applications that really need it, the number of supported applications/users should be controlled and cannot be unlimited. Hence the scalability requirement for the new service is limited.

4. The new service must be able to coexist with the regular transport service in the same hardware, and be backward compatible. Also, a transport flow can switch between regular transport and new service without service interruption.

3.3. Sub-layer in IP for Transport Control

In order to provide some new features for the layer above IP, it is very useful to introduce an additional sub-layer, Transport Control, between layer 3 (IP) and layer 4 (TCP/UDP). The new layer belongs to IP, and is present only when the system needs to provide extra control for the upper layer, in addition to the normal IP forwarding. Fig 1. illustrates a new stack with the sub-layer.
Figure 1: The new stack with a sub-layer in Layer 3

The new sub-layer is always bound with IP layer and can provide support of the features for upper layer, such as:

**In-band Signaling**

The IP header with the new sub-layer can carry the signaling information for the devices on the IP path. The information may include all QoS related parameters used for hardware programming.

**Congestion control**

The congestion state in each device on the path can be detected and notified to the source of flows by the sub-layer; The dynamic congestion control instruction can also be carried by the sub-layer and examined by network devices on the IP path.

**IP Path OAM**

The OAM instruction can be carried in the sub-layer, and the OAM state can be notified to the source of flows by the sub-layer. The OAM includes the path and device property detection, QoS forwarding diagnosis and report.

**IPv4** could use the IP option for the purpose of the sub-layer. But due to the limit size of the IP option, the functionalities, scalability of the layer is restricted.

**IPv6** can realize the sub-layer easily using IPv6 extension header [RFC8200]. The document will focus on the solution for IPv6 using different IPv6 extension headers.

**3.4. IP In-band signaling**

In-band signaling messages are carried along with the payload. It is guaranteed that the signaling follows the same path as the data flow, and this can bring up some advantages that other methods can hardly provide:
Diagnosis
The in-band signaling message takes the same path, same hops, same processing at each hop as the data packet, this will make the diagnosis for both signaling and data path easier.

Simplicity
The in-band signaling message is forwarded with the normal data packet, it does not need to run a separate protocol. This will dramatically reduce the complexity of the control.

Performance and scalability
Due to the simplicity of in-band signaling for control, it is easier to provide a better performance and scalability for a new future.

There have been similar works done or proposed in the industry for quite some time. The in-band QoS signaling for IPv6 was discussed by Lawrence Roberts in 2005 [I-D.roberts-inband-qos-ipv6]. The requirements of IP in-band signaling was proposed by Jon Haper in 2007 [I-D.harper-inband-signalling-requirements]. Telecommunications Industry Association (TIA) published a standard for "QoS Signaling for IP QoS Support and Sender Authentication" in 2006 [TIA].

This document proposes an optimized solution for QoS service using in-band signaling, and it also tries to address issues raised by previous proposals, such as security, scalability and performance.

The major differences from the previous works are:

1. Focus on IPv6 only.

2. The proposed solution could be driven by end-user operating system’s protocol stack such as TCP, UDP or other protocols, or by network device working as a proxy.

3. Simplified signaling process with minimal information carried, reduced QoS state maintenance at network devices.

4. Use different IPv6 options for signaling and signaling state report.

5. Support both bandwidth reservation and latency expectation at each hop.


7. Support dynamic QoS forwarding state monitoring.
IPv6 extension header is used for signaling. There are two types of extension header used for the purpose of transport QoS control, one is the hop-by-hop EH (HbH-EH) and another is the destination EH (Dst-EH).

The HbH-EH may be examined and processed by the nodes that are explicitly configured to do so [RFC8200], and these nodes are called HbH-EH-aware nodes. Note, not all nodes along a path need to HbH-EH-aware. HbH-EH is used to carry the QoS requirement for dedicated flow(s) and then the information is intercepted by HbH-EH-aware nodes on the path to program hardware accordingly.

The destination EH will only be examined and processed by the destination device that is associated with the destination IPv6 address in the IPv6 header. This EH is used to send the QoS related report information directly to the source of the signaling at other end.

The following figure illustrates the path setup process:

Setup Message (bandwidth, latency, setup state)

+------------------------HbH-EH------ -----------------+
|             |                          |             |
|             |                          |             |
+------+   +------------+   +-----+   +------------+   +------+
|      |   |            |   |     |   |            |   |      |
| host |---|     R1     |---| R2  |---|    R3      |---|server|
|      |   |HbH-EH-aware|   |     |   |HbH-EH-aware|   |      |
+------+   +------------+   +-----+   +------------+   +------+
|                                                       |
|                                                       |
+----------------------- Dst-EH ------------------------+

Setup State Report Message (setup state report)

Figure 2: Path Setup

Using the figure.2 for illustration, to set up a path with resource reservation, a setup message including QoS requirements, such as max/min bandwidth, burst size, the latency, and the setup state is sent from the host to the server. After each HbH-EH-aware node along the path receives the message, it reads the QoS information and programs the hardware for resource reservation, queuing management etc. The setup state object is updated at each HbH-EH-aware node to include
the QoS programming and provisioning result and the necessary hardware reference information for IP forwarding with QoS. After the setup message reaches the server, the server will send a setup state report message encoded as Dst-EH to the host. The setup state report message carries the path setup results from the setup state object.

4. Key Messages and Parameters

4.1. Setup and Setup State Report messages

Setup message is intended to program the hardware for QoS channel on the IP path from the source to the destination expressed in IPv6 header. It is embedded as the HbH-EH in an IPv6 packet and will be processed at each HbH-EH-aware node. For the simplicity, performance and scalability purpose, not all routers along the path need to do the processing or be HbH-EH-aware. For different QoS requirements and scenarios, different criteria can be used to configure HbH-EH-aware nodes.

A throttle router is the device that an interested TCP/UDP session cannot get the enough bandwidth to support its application, and it will also contribute more to latency than non-throttle routers. The regular throttle routers include the BRAS (broadband remote access server) in broadband access network, the PGW (PDN Gateway) in LTE network etc. In more general case, any routers which aggregated traffic may become as a throttle router. Throttle routers should be configured to process HbH-EH when:

- **Reserved bandwidth is required:** The throttle router is the critical point to be configured to process the hop-by-hop EH for the bandwidth reservation. Moreover, the direction of congestion must be considered.

- **Bounded latency is required:** In theory, each router and switch could contribute some delay to the end-to-end latency, but the throttle router will contribute more than non-throttle routers, and slow device will contribute more than fast device. We can use OAM to detect the latency contribution in a network, and configure those worst-case devices to process the HbH-EH.

Setup State Report message is the message sent from the destination host to the source host (from the point of view of the Setup message). The message is embedded into the Dst-EH in any data packet. The Setup State Report in the message is just a copy from the Setup message received at the destination host for a typical TCP session. The message is used at the source host to forward the packet later and to do the congestion control.
<Setup Message> ::= <Setup State Object> [ <Bandwidth Object> ] [ <Burst Object> ] [ <Latency Object> ] [ <OAM Object> ] [ <Authentication Object> ]

<Setup State Report Message> ::= <Setup State Report Object> [ <OAM Object> ]

4.2. Forwarding State and Forwarding State Report messages

After the QoS is programmed by the in-band signaling, the specified IP flows can be processed and forwarded for the QoS requirement. There are two ways for host to use the QoS channel for associated TCP session:

1. Host directly send the IP packet without any changes to the packet, this is for the following cases:

   * The hardware was programmed to use the tuples in IP header as identification for QoS process (SIS = 0), and

   * The packet does not function to collect the QoS forwarding state on the path.

2. Host add the Forward State message into a data packet’s IP header as Hbh-EH and send the packet, this is for the cases:

   * The hardware was programmed to use the Service ID as identification for QoS process (SIS != 0).

   * The hardware was programmed to use the tuples in IP header as identification for QoS process (SIS = 0), and the data packet functions to collect the QoS forwarding state on the path. This is the situation that host wants to detect the QoS forwarding state for the purpose of failure handling (See section 4.3).

Forwarding State message format is shown in the Section 6.7. It is used to notify the service ID and also update QoS forwarding state for the hops that are Hbh-EH-aware nodes.

After Forwarding State message is reaching the destination host, the host is supposed to retrieve it and form a Forwarding State Report message, and carry it in any data packet as the Dst-EH, then send it to the host in the reverse direction.
4.3. Hop Number

This is the parameter for total number of HbH-EH-aware nodes on the path. It is the field "Hop_num" in Setup message, and is used to locate the bit position for "Setup State" and the "Service ID" in "Service ID List". The value of "Hop_num" must be decremented at each HbH-EH-aware node. At the receiving host of the in-band signaling, the Hop_num must be zero.

The source host must know the exact hop number, and setup the initial value in the Setup message. The exact hop number can be detected using OAM message.

4.4. Flow Identifying Method and Service ID

A QoS channel might be enforced for a group of flows or a delicate flow, and flow identifying method means the way of identifying a flow or a group of flows that can use a HW programmed QoS channel. Different levels of flow granularities to support QoS are defined as below:

Flow level
The flow identification could be 5 tuples for non IPSec IPv6 packet: the source, destination IP address, protocol number, source and destination port number, and also could be 3 tuples for IPSec IPv6 packet: the source, destination IP address and the flow label.

Address level In-band Signaling
A flow of packets share the same source, destination IP address, but with different protocol number. This is the scenario that the signaling is for the aggregated flows which have the same source, destination address. i.e, All TCP/UDP flows between the same client and same server (only one address for client and one for server)

Transport level In-band Signaling
Packets share the same source, destination IP address, protocol number, but with different source or destination port number (non-IPSec) or different flow label (IPSec). This could be for the
aggregated TCP or UDP flows that started and terminated at the same IP addresses.

DiffServ level In-band Signaling
Packets share the same DSCP value. This means aggregated differentiated service flows that have the same DSCP value. The DSCP value is determined by the 6 most-significant bits in 8-bits DiffServ field for IPv4 or 8-bits Traffic Class field for IPv6.

There are two ways for flow identifying. One is by tuple or DSCP value in IP header, another is by a local significant number, called service ID, generated and maintained in a router. When "Service ID Size" (SIS) is zero, it means the "Flow identification method" (FI) is used for both control plane and data plane. When "SIS" is not zero, it means "FI" is only used in signaling of setting up the QoS channel, and the data plane will only use the "Service ID". The use of local generated number to identify flow is to speed up the flow lookup and QoS process for data plane.

The "Service ID List" is a list of "Service ID" for all hops that are HbH-EH-aware nodes on the IP path. When a router receives a HbH-EH, it may generate a service ID for the flow(s) that is defined by the Flow Identifying Method in "FI". Then the router must attach the service ID value to the end of the Service ID List. After the packet reaches the destination host, the Service ID List will be that the 1st router’s service ID as the list header, and the last router’s service ID as the list tail.

4.5. QoS State and life of Time

After a router is programmed for a QoS, a QoS state is created. The QoS state life is determined by the "Time" in the Setup message. Whenever there is a packet processed by a QoS state, the associated timer for the QoS state is reset. If the timer of a QoS state is expired, the QoS state will be erased and the associated resource will be released.

In order to keep the QoS state active, an application at source host can send some zero size of data to refresh the QoS state.

When the Time is set to zero, it means the life of the QoS State will be kept until the de-programming message is received.

4.6. Authentication

The in-band signaling is designed to have a basic security mechanism to protect the integrity of a signaling message. The Authentication message is to attach to a signaling message, the source host
calculates the harsh value of a key and all invariable part of a signaling message (Setup message: ver, FI, R, SIS, P, Time; Bandwidth message, Latency message, Burst message). The key is only known to the hosts and all HbH-EH-aware nodes. The securely distribution of the key is out the scope of the document.

5. Packet Forwarding

To achieve the required QoS, after the path setup with guaranteed bandwidth there are some requirements to be met during data forwarding. These include the hardware capability, the scheme for the data forwarding, QoS processing, state report, etc.

5.1. Basic Hardware Capability

Section 4 explains how QoS guaranteed path can be set up and the corresponding messages used, however different implementations may vary in details. To achieve the satisfactory targets for performance and scalability, the protocol must be cooperated with capable hardware to provide the desired fine-grained QoS for different transport.

In our experiment to implement the feature for TCP, we used a network processor with traffic management feature. The traffic management can provide the fine-grained QoS for any configured flow(s).

The following capabilities are RECOMMENDED:

1. The in-banding signaling is processed in network processor without punting to controller CPU for help

2. The QoS forwarding state is kept and maintained in network processor without the involvement from controller CPU.

3. The QoS state has a life of a pre-configured time and will be automatically deleted if there is no data packet processed by that QoS state. The timer can be changed on the fly.

4. The data forwarding does not need to be done at the controller CPU, or so called slow path. It is at the same hardware as the normal IP forwarding. For any IP packet, the QoS forwarding is executed first. Normal forwarding will be executed if there is no QoS state associated with the identification of the flow.

5. The QoS forwarding and normal forwarding can be switched on the fly.
The details of data plane and hardware related implementations, such as traffic classification, shaping, queuing and scheduling, are out of scope of this document. The report of [NGP] has given some experiments and results by using commercial hardwares.

5.2. Flow Identification in Packet Forwarding

Flow identification in Packet Forwarding is same as the QoS channel establishment by Setup message. It is to forward a packet with a specified QoS process if the packet is identified to be belonging to specified flow(s).

There are two method used in data forwarding to identify flows:

1. Hardware was programmed to use tuples in IP header implicitly. This is indicated by that the "SIS" is zero or the Service ID is not used. When a packet is received, its tuples are looked up according to the value of "FI". If there is a QoS table has match for the packet, the packet will be processed by the QoS state found in the QoS table. This method does not need any EH added into the data packet unless the data packet function to collect the QoS forwarding state on the path.

2. Hardware was programmed to use service ID to identify flows. This is indicated by that the "SIS" is not zero. When a packet is received, the service ID associated with the hop is retrieved and looked up for the QoS table. If it has match for the packet, the packet will be processed by the QoS state entry found in the QoS table.

5.3. QoS Forwarding State Detection and Failure Handling

QoS forwarding may fail due to different reasons:

1. Hardware failure in HbH-EH-aware node.

2. IP path change due to link failure, node failure or routing changes; And the IP path change has impact to the HbH-EH-aware node.

3. Network topology change; and the change leads to the changes of HbH-EH-aware nodes.

Application may need to be aware of the service status of QoS guarantee when the application is using a TCP session with QoS. In order to provide such feature, the TCP stack in the source host can detect the QoS forwarding state by sending TCP data packet with Forwarding State message coded as HbH-EH. After the TCP data packet
reaches the destination host, the host will copy the forwarding state into a Forwarding State Report message, and send it with another TCP packet (for example, TCP-ACK) in reverse direction to the source host. Thereafter, the source host can obtain the QoS forwarding state on all HbH-EH-aware nodes.

A host can do the QoS forwarding state detection by three ways: on demand, periodically or constantly.

After a host detects that there is QoS forwarding state failure, it can repair such failure by sending another Setup message embedded into a HbH-EH of any TCP packet. This repairing can handle all failure case mentioned above.

If a failure cannot be repaired, host will be notified, and appropriate action can be taken, see section 7.1

6. Details of Working with Transport Layer

The proposed new IP service is transport agnostic, which means any transport layer protocol can use it.

6.1. Working with TCP

Considering TCP as the most widely used transport layer protocol, this document uses TCP as an example of transport protocol to show how it works with the proposed IP service.

The following is the list of messages for signaling and associated data forwarding.

- Setup: This is for the setup of QoS channel through the IP path.
- Bandwidth: This is the required bandwidth for the QoS channel. It has minimum (CIR) and maximum bandwidth (PIR).
- Latency: This is the required latency for the QoS channel, it is the bounded latency for each hop on the path. This is not the end to end latency.
- Burst: This is the required burst for the QoS channel, it is the maximum burst size.
- Authentication: This is the security message for a in-band signaling.
- OAM: This is the Operation and Management message for the QoS channel.
Setup State Report: This is the state report of a setup message.

Forwarding State: This is the forwarding state message used for data packet.

Forwarding State Report: This is the forwarding state report of a QoS channel.

There are three scenarios of QoS signaling for TCP session setup with QoS:

1. Upstream: This is for the direction of client to server. A application decides to open a TCP session with upstream QoS (for uploading), it will call TCP API to open a socket and connect to a server. The client host will form a TCP SYN packet with the HbH-EH in the IPv6 header. The EH includes Setup message and Bandwidth message, and optionally Latency, Burst, Authentication and OAM messages. The packet is forwarded at each hop. Each HbH-EH-aware nodes will process the signaling message to finish the following tasks before forwarding the packet to next hop:

   * Retrieve the QoS parameters to program the Hardware, it includes: FL, Time, Bandwidth, Latency, Burst
   * Update the field in the EH, it includes: Hop_number, Total_latency, and possibly Service ID List

When the server receives the TCP SYN, the Host kernel will also check the HbH-EH while punting the TCP packet to the TCP stack for processing. If the HbH-EH is present and the Report bit is set, the Host kernel must form a new Setup State Report message, all fields in the message must be copied from the Setup message in the HbH-EH. When the TCP stack is sending the TCP-SYNACK to the client, the kernel must add the Setup State Report message as a Dst-EH in the IPv6 header. After this, the IPv6 packet is complete and can be sent to wire; When the client receives the TCP-SYNACK, the Host kernel will check the Dst-EH while punting the TCP packet to the TCP stack for processing. If the Dst-EH is present and the Setup State Report message is valid, the kernel must read the Setup State Report message. Depending on the setup state, the client will operate according to description in section 7.1

2. Downstream: This is for the direction of server to client. A application decides to open a TCP session with downstream QoS (for downloading), it will call TCP API to open a socket and connect to a server. The client host will form a TCP SYN packet with the Dst-EH in the IPv6 header. The EH includes Bandwidth
message, and optionally Latency, Burst messages. The packet is forwarded at each hop. Each hop will not process the Dst-EH. When the server receives the TCP SYN, the Host kernel will check the Dst-EH while punting the TCP packet to the TCP stack for processing. If the Dst-EH is present, the Host kernel will retrieve the QoS requirement information from Bandwidth, Latency and Burst message, and check the QoS policy for the user. If the user is allowed to get the service with the expected QoS, the server will form a Setup message similar to the case of client to server, and add it as the HbH-EH in the IPv6 header, and send the TCP-SYNACK to client. Each HbH-EH-aware nodes on the path from server to client will process the message similar to the case of client to server. After the client receives the TCP-SYNACK, The client will send the Setup State Report message to server as the Dst-EH in the TCP-ACK. Finally the server receives the TC-ACK and Setup State Report message, it can send the data to the established session according to the pre-negotiated QoS requirements.

3. Bi-direction: This is the case that the client wants to setup a session with bi-direction QoS guarantee. The detailed operations are actually a combination of Upstream and Downstream described above.

After a QoS channel is setup, the in-band signaling message can still be exchanged between two hosts, there are two scenarios for this.

1. Modify QoS on the fly: When the pre-set QoS parameters need to be adjusted, the application at source host can re-send a new in-band signaling message, the message can be embedded into any TCP packet as a IPv6 HbH-EH. The QoS modification should not impact the established TCP session and programmed QoS service. Thus, there is no service impacted during the QoS modification. Depending on the hardware performance, the signaling message can be sent with TCP packet with different data size. If the performance is high, the signaling message can be sent with any TCP packet; otherwise, the signaling message should be sent with small size TCP packet or zero-size TCP packet (such as TCP ACK). Modification of QoS on the fly is a very critical feature for the so called "Application adaptive QoS transport service". With this service, an application (or the proxy from a service provider) could setup an optimized CIR for different stage of application for the economical and efficient purpose. For example, in the transport of compressed video, the I-frame has big size and cannot be lost, but P-frame and B-frame both have smaller size and can tolerate some loss. There are much more P-frame and B-frame than I-frame in videos with smooth changes and variations in images [I-D.han-iccrg-arvr-transport-problem].
Based on this characteristics, application can request a relatively small CIR for the time of P-frame and P-frame, and request a big CIR for the time of I-frame.

2. Repairing of the QoS channel: This is the case the QoS channel was broken and need to be repaired, see section 5.3.

6.2. Working with UDP and other Protocols

There are other transport layer protocols, such as UDP, QUICK and SCTP, and for these protocols similar strategy as TCP can be applied. The to establish a closed-loop for the transport control.

For protocols with natively bi-directional control mechanism such as SCTP, only some QoS control functionalities for the protocol need to be added. The mechanism for TCP can be borrowed for such job. There will be the QoS setup for one directional data stream, and QoS setup state report for another directional data stream. The protocol may also have functionalities in the stack to handle the adjustment of the behaviour for different QoS setup and setup states.

For protocols that natively lack the feed-back control mechanism to form a closed-loop such as UDP, this mechanism needs to be added into the streams. There are two options to realize this:

1. Modify the protocol itself to have some state machine to establish the closed-loop for the protocol. This can be done in the kernel of the OS by modifying the protocol stack.

2. Modify the user data stream to introduce the closed-loop scheme, this becomes as application work. It is up to application to add or modify codes for the state machine of the closed-loop control.

7. Additional Considerations

This document only covers the details of setting up a path with QoS using IPv6, and TCP is used as an example of transport layer protocol to achieve flow level service. Only basic scenarios are covered, and there are lots of open issues to be researched. The following is a non-comprehensive list, and they can be addressed in separate drafts.

7.1. User and Application driven

The QoS transport service is initiated and controlled by end user’s application. Following tasks are done in host:

1. The detailed QoS parameters in signaling message are set by end user application. New socket option must be added, the option is
a place holder for QoS parameters (Setup, Bandwidth, etc.), Setup State Report and Forwarding State Report messages.

2. The Setup State Report and Forwarding State Report message received at host are processed by transport service in kernel. The Setup State Report message processed at host can result in the notification to the application whether the setup is successful. If the setup is successful, the application can start to use the socket having the QoS support; If the setup is failed, the application may have three choices:

* Lower the QoS requirement and re-setup a new QoS channel with new in-band signaling message.

* Use the TCP session as traditional transport without any QoS support.

* Lookup the service provider for help to locate the problem in network.

7.2. Traffic Management in Host

In order to better accommodate this new IP in-band service, the OS on a host may be changed in traffic management related areas. There are two parts for traffic management to be changed: one is to manage traffic going out a host’s shared links, and the other is congestion control for TCP flows.

1. For current traffic management in a host, all TCP/UDP sessions will share the bandwidth for all egress links. For the purpose to work with the differentiated service provided by under layer network in bandwidth and latency, the kernel may allocate expected resource to applications that are using the QoS transport service. For example, kernel can queue different packets from different applications or users to different queue and schedule them in different priority. Only after this change, some application can use more bandwidth and get less queuing delay for a link than others.

2. The congestion control in a host manages the behavior of TCP flow(s). This includes important features like slow start, AIMD, fast retransmit, selective ACK, etc. To accommodate the benefit of the QoS guaranteed transport service, the congestion control can be much simpler [I-D.han-tsvwg-cc]. The new congestion control is related to the implementation of QoS guarantee. Following is a simple congestion control algorithm assuming that the CIR is guaranteed and PIR is shared between flows:
* There is no slow start, the TCP can start sending traffic at the rate of CIR.

* The AIMD is kept, but the range of the sawtooth pattern should be maintained between CIR and PIR.

* Other congestion control features can be kept.

7.3. Heterogeneous Network

When an IP network is connected with a non-IP network, such as MPLS or Ethernet network, the in-band signaling should also work in that network to achieve an end-to-end connection. The behavior, protocol and rules in the interworking with non-IP network is out of the scope of this draft, and further research needs to be done to solve the problem.

7.4. Proxy Control

It is expected that in a real service provider network, the in-band signaling will be checked, filtered and managed at proxy routers. It serves the following purposes:

1. A proxy can check if the in-band signaling from an end user meets the SLA compliance. This adds extra security and DOS attack prevention.

2. A proxy can collect the statistics for user’s TCP flows and check the in-band signaling for accounting and charging.

3. A proxy can insert and process appropriate in-band signaling for TCP flows if the host does not support this new feature. This can provide backward compatibility, also enable the host to use the new feature.

8. IANA Considerations

This document defines a new option type for the Hop-by-Hop Options header and the Destination Options header. According to [RFC8200], the detailed value are:
Figure 3: The New Option Type

1. The highest-order 2 bits: 00, indicating if the processing IPv6 node does not recognize the Option type, skip over this option and continue processing the header.

2. The third-highest-order bit: 0, indicating the Option Data does not change en route.

3. The low-order 5 bits: 10000, assigned by IANA.

This document also defines a 4-bit subtype field, for which IANA will create and maintain a new sub-registry entitled "In-band signaling Subtypes" under the "Internet Protocol Version 6 (IPv6) Parameters" [IPv6_Parameters] registry. Initial values for the subtype registry are given below.
9. Security Considerations

It is important to guarantee that the resource reservation is used by authenticated users, and false signaling should not be accepted or processed. The following aspects may be considered:

Authentication of user

If an user is interested in using this new service, the user should sign up to a service provider. Service provider should do the proper authentication check for a new user, and establish account for the user.

After the sign up, a user should provide a security key to the service provider through a secured channel (https, registered mail, etc.), or the key could be generated and given to user by the service provider. Service provider should distribute the security key of the user to different network device. More specifically, the security key should be distributed securely to all HbH-EH-aware nodes for an open network, or the proxy for a closed network.

Proxy
Proxy or gateway is the 1st network device connecting to
customer’s devices (Host, phone, etc.) that can generate the
signaling for resource reservation. The functionality of the
Proxy is to check if the signaling is allowed to go through
SP’s network. This can be done by checking the signaling
integrity and other info associated with the user, such as the
source/destination IP address, the account balance, the user’s
privilege, etc.

Authentication of signaling message

The signaling for resource reservation should be checked at
each HbH-EH-aware nodes or a proxy node.

Service ID is originally used for performance improvement of
forwarding with QoS, and it can also provide additional security
protection of forwarding resource in data plane. Service ID in each
HbH-EH-aware node is to represent an IP flow with programmed QoS
service, and it is a local significant number generated by a router
to identify a flow that was offered QoS service. So, the router can
periodically change the number for the same flow to protect any
middle box sniffing for DOS attacking. It can be done by host
periodical send out in-band signaling with the same QoS parameters
and obtain the new Service ID and Service ID List for the use of next
data forwarding.

10. References

10.1. Normative References

[RFC2119]  Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,

Control", RFC 2581, DOI 10.17487/RFC2581, April 1999,
<https://www.rfc-editor.org/info/rfc2581>.

[RFC8174]  Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC
2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174,

(IPv6) Specification", STD 06, RFC 8200,
DOI 10.17487/RFC8200, July 2017,

Han, et al.          Expires October 19, 2019          [Page 25]
10.2. Informative References


[I-D.ietf-aqm-pie]

[I-D.ietf-detnet-use-cases]

[I-D.ietf-spring-segment-routing]

[I-D.ietf-tcpm-dctcp]

[I-D.roberts-inband-qos-ipv6]

[I-D.sridharan-tcpm-ctcp]

[IPv6_Parameters]

[MEC]

[NGP]
Appendix A. Acknowledgements

The authors are very grateful to Fred Baker for his valuable contributions to this document.

We appreciate the following people who made lots of contributions to this draft: Guoping Li, Boyan Tu, and Xuefei Tan, and thank Huawei Nanjing research team led by Feng Li to provide the Product on
Concept (POC) development and test, the team members include Fengxin Sun, Xingwang Zhou, and Weiguang Wang. We also like to thank other people involved in the discussion of solution: Tao Ma from Future Network Strategy dept.

Appendix B. Message Objects

This section defines detailed objects used in different messages.

B.1. Setup State Object
Type = 0, Setup state;

Version: The version of the protocol for the QoS

FI: Flow identification method,
   0: 5 tuples; 1: src,dst,port; 2: src,dst; 3: DSCP

R: If the destination host report the received Setup state to
   the src address by Destination EH. 0: dont report; 1: report

SIS: Service ID size; 0: 0bits, 1: 16bits, 2: 20bits, 3: 32bits

P: 0: program HW for the QoS from src to dst;
   1: De-program HW for the QoS from src to dst

Time: The life time of QoS forwarding state in second.

Hop_num: The total hop number on the path set by host. It must be
decremented at each hop after the processing.

u: the unit of latency, 0: ms; 1: us

Total_latency : Latency accumulated from each hop, each hop will
add the latency in the device to this value.

Figure 5: The Setup State Object

Setup state for each hop index: each bit is the setup state on each
hop on the path, 0: failed; 1: success. The 1st hop is at the most
significant bit.
Service ID list for hops: it is for all hops on the path, each service ID bit size is defined in SIS. The 1st service ID is at the top of the stack. Each hop add its service ID at the correct position indexed by the current hop number for the router.

The Setup object is embedded into the hop-by-hop EH to setup the QoS in the device on the IP forwarding path. To keep the whole setup message size unchanged at each hop, the total hop number must be known at the source host. The total hop number can be detected by OAM. The service ID list is empty before the 1st hop receives the in-band signaling. Each hop then fill up the associated service ID into the correct place determined by the index of the hop.

B.2. Bandwidth Object

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0 0 0 1|      reserved         |       Minimum bandwidth       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Maximum bandwidth      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type = 1,

Minimum bandwidth : The minimum bandwidth required, or CIR, unit Mbps

Maximum bandwidth : The maximum bandwidth required, or PIR, unit Mbps

Figure 6: The Bandwidth Object

B.3. Burst Msg
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
0 0 1 0 | Burst size |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Type = 2,
Burst size : The burst size, unit M bytes

Figure 7: The burst message

B.4. Latency Object

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
0 0 1 1 |u| Latency |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Type = 3,
u: the unit of the latency
0: ms; 1: us
Latency: Expected maximum latency for each hop

Figure 8: The Latency Object

B.5. Authentication Object
Type = 4,
MAC_ALG: Message Authentication Algorithm

0: MD5; 1:SHA-0; 2: SHA-1; 3: SHA-256; 4: SHA-512

MAC data: Message Authentication Data;
Res: Reserved bits
Size of signaling data (opt_len): Size of MAC data + 2
MD5: 18; SHA-0: 22; SHA-1: 22; SHA-256: 34; SHA-512: 66

Figure 9: The Authentication Object

B.6. OAM Object

Type = 5,
OAM_t : OAM type
OAM_len : 8-bit unsigned integer. Length of the OAM data, in octets;
OAM data: OAM data, details of OAM data are TBD.

Figure 10: The OAM Object
B.7. Forwarding State Object

```
+-----------------------------+-----------------------------+
| 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 |
+------------------------------------------------------------------+
| 0 1 1 0 ver | FI | R | SIS | P | Time  | Hop_num | u | Total_latency |
+------------------------------+-----------------------------------+
| Forwarding state for each hop index                             |
+------------------------------+-----------------------------------+
| Service ID list for hops                                         |
+------------------------------+-----------------------------------+
```

Type = 6, Forwarding state;

All parameter definitions and process in the 1st row are same in the setup message.

Forward state for each hop index: each bit is the fwd state on each hop on the path, 0: failed; 1: success; The 1st hop is at the most significant bit.

Service ID list for hops: it is for all hops on the path, each index bit size is defined in SIS. The list is from the setup report message.

Figure 11: The Forwarding State Object

B.8. Setup State Report Object
Type = 7, Setup state report;

H: Hop number bit. When a host receives a setup message and form a setup report message, it must check if the Hop_num in setup message is zero. If it is zero, the H bit is set to one, and if it is not zero, the H bit is clear. This will notify the source of setup message that if the original Hop_num was correct. Following are directly copied from the setup message:
u, Total_latency;
State for each hop index
Service ID list for hops.

Figure 12: The Setup State Report Object

Type = 8, Forwarding state report;

H: Hop number bit. When a host receives a Forward State message
and form a Forward State Report message, it must check if the
Hop_num in Forward State message is zero. If it is zero, the H bit
is set to one, and if it is not zero, the H bit is clear.
This will notify the source of Forward State message that if the
original Hop_num was set correct.
Following are directly copied from the Forward State message:
u, Total_latency;
Forwarding State for each hop index

Figure 13: The Fwd State Report Object

Authors’ Addresses

Lin Han
Huawei Technologies
2330 Central Expressway
Santa Clara, CA 95050
USA

Phone: +10 408 330 4613
Email: lin.han@huawei.com
As communication devices become more hybrid, smart devices include more media-rich communication applications, and the boundaries between telecommunication and other applications becomes less clear. Simultaneously, as the end-devices become more mobile, application traffic transits more often between enterprise networks, the Internet, and cellular telecommunication networks. In this context, it is crucial that quality of service be aligned between these different environments. However, this is not always the case by default, and cellular communication networks use a different QoS nomenclature from the Internet and enterprise networks. This document specifies a set of 3rd Generation Partnership Project (3GPP) Quality of Service (QoS) Class Identifiers (QCI) to Differentiated Services Code Point (DSCP) mappings, to reconcile the marking recommendations offered by the 3GPP with the recommendations offered by the IETF, so as to maintain a consistent QoS treatment between cellular networks and the Internet.
# Table of Contents

1. Introduction ................................................. 3  
   1.1. Related Work ............................................. 4  
   1.2. Applicability Statement ................................. 4  
   1.3. Document Organization .................................. 5  
   1.4. Requirements language .................................. 5  
   1.5. Terminology Used in this Document ..................... 5  
2. Service Comparison and Default Interoperation of Diffserv and 3GPP LTE .................................................. 7  
   2.1. Diffserv Domain Boundaries ............................... 7  
   2.2. QCI and Bearer Model in 3GPP ............................ 7  
   2.3. QCI Definition and Logic ................................ 9  
   2.3.1. Conversational ........................................ 9  
   2.3.2. Streaming ............................................. 9  
   2.3.3. Interactive ........................................... 10  
   2.3.4. Background ............................................ 10  
   2.4. QCI implementations ..................................... 13  
   2.5. GSMA IPX Guidelines Interpretation and Conflicts ....... 13  
3. P-GW Device Marking and Mapping Capability Recommendations . 14  
4. DSCP-to-QCI Mapping Recommendations .......................... 15  
   4.1. Control Traffic .......................................... 15  
   4.1.1. Network Control Protocols .............................. 15  
   4.1.2. Operations, Administration, and Maintenance (OAM) ... 16  
   4.2. User Traffic ............................................. 16  
   4.2.1. Telephony ............................................. 16  
   4.2.2. Signaling ................................................ 17  
   4.2.3. Multimedia Conferencing ............................... 17  
   4.2.4. Real-Time Interactive ................................ 18  
   4.2.5. Multimedia Streaming ................................ 18  
   4.2.6. Broadcast Video ........................................ 19  
   4.2.7. Low-Latency Data .................................... 19  
   4.2.8. High-Throughput Data .................................. 20
1. Introduction

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

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


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

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

1.2. Applicability Statement

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

1.3. Document Organization

This document is organized as follows:

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

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

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

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

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

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

1.4. Requirements language

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

1.5. Terminology Used in this Document

Key terminology used in this document includes:

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

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

UE: User Equipment, the end-device.

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

SAE: System Architecture Evolution.

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

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

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

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

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

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

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

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

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

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

2.1. Diffserv Domain Boundaries

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

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

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

2.2. QCI and Bearer Model in 3GPP

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

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

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

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

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

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

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

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

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

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

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

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

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

2.3. QCI Definition and Logic

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

2.3.1. Conversational

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

2.3.2. Streaming

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

2.3.3. Interactive

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

2.3.4. Background

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

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

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

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

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

2.5. GSMA IPX Guidelines Interpretation and Conflicts

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

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

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

- Recommending EF for real-time (conversational) video, for which [RFC4594] recommends AF41.

- Recommending AF31 for DNS traffic, for which [RFC4594] recommends the standard service class (DF)

- Recommending AF31 for all types of signaling traffic, thus losing the ability to differentiate between the various types of signaling flows, as recommended in[RFC4594] section 5.1.
o Recommending AF21 for WAP browsing and WEB browsing, for which [RFC4594] recommends the High Throughput data class.

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

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

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

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

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

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

3. P-GW Device Marking and Mapping Capability Recommendations

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

o mark DSCP, per Diffserv standards

o mark QCI, per the [TS 23.203] standard

o support fully-configurable mappings between DSCP and QCI

o process DSCP markings set by cellular endpoint devices

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

o mark DSCP, per Diffserv standards
o mark QCI, per the [TS 23.203] standard

o support fully-configurable mappings between DSCP (set by applications in software) and QCI (set by the operating system and/or the LTE infrastructure)

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

4. DSCP-to-QCI Mapping Recommendations

4.1. Control Traffic

4.1.1. Network Control Protocols

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

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

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

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

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

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

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

4.2. User Traffic

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

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

4.2.1. Telephony

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

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

4.2.2. Signaling

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

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

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

4.2.3. Multimedia Conferencing

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

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

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

4.2.4. Real-Time Interactive

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

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

4.2.5. Multimedia Streaming

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

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

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

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

4.2.6. Broadcast Video

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

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

4.2.7. Low-Latency Data

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

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

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

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

4.2.8. High-Throughput Data

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

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

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

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

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

4.2.9. Standard

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

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

4.2.10. Low-Priority Data

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

Note: This marking recommendation may change in the future, as [LE-PHB] defines a Lower Effort (LE) PHB for Low-Priority Data traffic and recommends an additional DSCP for this traffic.

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

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

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

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

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

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

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

5.1. QCI and Diffserv Logic Reconciliation

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

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

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

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

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

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

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

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

5.2. Voice QCI [1]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5.6. Live streaming and interactive gaming [7]

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

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

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

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

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

5.8. V2X messaging [75,3,9]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5.11. Mission-critical data [70]

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

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

<table>
<thead>
<tr>
<th>QCI</th>
<th>Resource Type</th>
<th>Priority Level</th>
<th>Example Services</th>
<th>Recommended DSCP (PHB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GBR</td>
<td>2</td>
<td>Conversational Voice</td>
<td>44 (VA)</td>
</tr>
<tr>
<td>2</td>
<td>GBR</td>
<td>4</td>
<td>Conversational Video (Live Streaming)</td>
<td>35 (N.A.)</td>
</tr>
<tr>
<td>3</td>
<td>GBR</td>
<td>3</td>
<td>Real Time Gaming, V2X messages, Electricity distribution (medium voltage) Process automation (monitoring)</td>
<td>19 (N.A.)</td>
</tr>
<tr>
<td>4</td>
<td>GBR</td>
<td>5</td>
<td>Non-Conversational Video (Buffered Streaming)</td>
<td>37 (N.A.)</td>
</tr>
<tr>
<td>65</td>
<td>GBR</td>
<td>0.7</td>
<td>Mission Critical user plane Push To Talk voice (e.g., MCPTT)</td>
<td>42 (N.A.)</td>
</tr>
<tr>
<td>66</td>
<td>GBR</td>
<td>2</td>
<td>Non-Mission-Critical user plane Push To Talk voice</td>
<td>43 (N.A.)</td>
</tr>
<tr>
<td>67</td>
<td>GBR</td>
<td>1.5</td>
<td>Mission Critical Video user plane</td>
<td>33 (N.A.)</td>
</tr>
<tr>
<td>75</td>
<td>GBR</td>
<td>2.5</td>
<td>V2X messages</td>
<td>17 (N.A.)</td>
</tr>
<tr>
<td>82</td>
<td>GBR</td>
<td>1.9</td>
<td>Discrete automation (small packets)</td>
<td>27 (N.A.)</td>
</tr>
<tr>
<td>83</td>
<td>GBR</td>
<td>2.2</td>
<td>Discrete automation (large packets)</td>
<td>29 (N.A.)</td>
</tr>
<tr>
<td>84</td>
<td>GBR</td>
<td>2.4</td>
<td>Intelligent Transport Systems</td>
<td>31 (N.A.)</td>
</tr>
<tr>
<td>85</td>
<td>GBR</td>
<td>2.1</td>
<td>Electricity Distribution - High</td>
<td>25 (N.A.)</td>
</tr>
</tbody>
</table>
### Voltage

<table>
<thead>
<tr>
<th>QCI</th>
<th>DSCP</th>
<th>VRF</th>
<th>Application Description</th>
<th>Codepoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Non-GBR</td>
<td>1</td>
<td>IMS Signalling</td>
<td>40 (CS5)</td>
</tr>
<tr>
<td>6</td>
<td>Non-GBR</td>
<td>6</td>
<td>Video (Buffered Streaming) TCP-based (e.g. www, email, chat, ftp, p2p file sharing, progressive video)</td>
<td>10 (AF11)</td>
</tr>
<tr>
<td>7</td>
<td>Non-GBR</td>
<td>7</td>
<td>Voice, Video (live streaming), interactive gaming</td>
<td>38 (AF43)</td>
</tr>
<tr>
<td>8</td>
<td>Non-GBR</td>
<td>8</td>
<td>Video (buffered streaming) TCP-based (e.g. www, email, chat, ftp, p2p file sharing, progressive video)</td>
<td>12 (AF12)</td>
</tr>
<tr>
<td>9</td>
<td>Non-GBR</td>
<td>9</td>
<td>Same as 8</td>
<td>14 (AF13)</td>
</tr>
<tr>
<td>69</td>
<td>Non-GBR</td>
<td>0.5</td>
<td>Mission Critical delay sensitive signalling (e.g., MC-PTT signalling, MC Video signalling)</td>
<td>41 (N.A.)</td>
</tr>
<tr>
<td>70</td>
<td>Non-GBR</td>
<td>5.5</td>
<td>Mission Critical Data (e.g. example services are the same as QCI 6/8/9)</td>
<td>20 (AF22)</td>
</tr>
<tr>
<td>79</td>
<td>Non-GBR</td>
<td>6.5</td>
<td>V2X messages</td>
<td>21 (N.A.)</td>
</tr>
<tr>
<td>80</td>
<td>Non-GBR</td>
<td>6.8</td>
<td>Low latency eMMB applications (TCP/UDP-based); augmented reality</td>
<td>32 (CS4)</td>
</tr>
</tbody>
</table>

### 6. IANA Considerations

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

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

8. Security Recommendations for General QoS

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

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

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

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

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

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

9. References

9.1. Normative References


9.2. Informative References


Henry & Szigeti Expires October 15, 2019 [Page 36]
Internet-Draft                DIFFSERV-QCI                    April 2019


Authors' Addresses

Jerome Henry
Cisco

Email: jerhenryl@cisco.com
Packetization Layer Path MTU Discovery for Datagram Transports
draft-ietf-tsvwg-datagram-plpmtud-08

Abstract

This document describes a robust method for Path MTU Discovery (PMTUD) for datagram Packetization Layers (PLs). It describes an extension to RFC 1191 and RFC 8201, which specifies ICMP-based Path MTU Discovery for IPv4 and IPv6. The method allows a PL, or a datagram application that uses a PL, to discover whether a network path can support the current size of datagram. This can be used to detect and reduce the message size when a sender encounters a network black hole (where packets are discarded). The method can probe a network path with progressively larger packets to discover whether the maximum packet size can be increased. This allows a sender to determine an appropriate packet size, providing functionally for datagram transports that is equivalent to the Packetization Layer PMTUD specification for TCP, specified in RFC 4821.

The document also provides implementation notes for incorporating Datagram PMTUD into IETF datagram transports or applications that use datagram transports.

When published, this specification updates RFC 4821.

Status of This Memo

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

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

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

This Internet-Draft will expire on 7 December 2019.

Copyright Notice

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

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

Table of Contents

1. Introduction .................................................. 3
   1.1. Classical Path MTU Discovery .............................. 4
   1.2. Packetization Layer Path MTU Discovery .................. 6
   1.3. Path MTU Discovery for Datagram Services ................. 7
2. Terminology ................................................... 7
3. Features Required to Provide Datagram PLPMTUD .................. 10
4. DPLPMTUD Mechanisms ........................................... 12
   4.1. PLPMTU Probe Packets .................................. 12
   4.2. Confirmation of Probed Packet Size ...................... 14
   4.3. Detection of Unsupported PLPMTU Size, aka Black Hole Detection ............................................ 14
   4.4. Response to PTB Messages ................................ 15
   4.4.1. Validation of PTB Messages ............................ 15
   4.4.2. Use of PTB Messages .................................. 16
5. Datagram Packetization Layer PMTUD ............................. 17
   5.1. DPLPMTUD Components .................................... 18
   5.1.1. Timers .............................................. 18
   5.1.2. Constants .......................................... 19
   5.1.3. Variables .......................................... 20
   5.1.4. Overview of DPLPMTUD Phases .......................... 21
   5.2. State Machine .......................................... 23
   5.3. Search to Increase the PLPMTU ............................ 26
   5.3.1. Probing for a larger PLPMTU .......................... 26
   5.3.2. Selection of Probe Sizes .............................. 27
   5.3.3. Resilience to Inconsistent Path Information ............ 27
   5.4. Robustness to Inconsistent Paths ......................... 28
1. Introduction

The IETF has specified datagram transport using UDP, SCTP, and DCCP, as well as protocols layered on top of these transports (e.g., SCTP/UDP, DCCP/UDP, QUIC/UDP), and direct datagram transport over the IP network layer. This document describes a robust method for Path MTU Discovery (PMTUD) that may be used with these transport protocols (or the applications that use their transport service) to discover an appropriate size of packet to use across an Internet path.
1.1. Classical Path MTU Discovery

Classical Path Maximum Transmission Unit Discovery (PMTUD) can be used with any transport that is able to process ICMP Packet Too Big (PTB) messages (e.g., [RFC1191] and [RFC8201]). In this document, the term PTB message is applied to both IPv4 ICMP Unreachable messages (type 3) that carry the error Fragmentation Needed (Type 3, Code 4) [RFC0792] and ICMPv6 packet too big messages (Type 2) [RFC4443]. When a sender receives a PTB message, it reduces the effective MTU to the value reported as the Link MTU in the PTB message, and a method that from time-to-time increases the packet size in attempt to discover an increase in the supported PMTU. The packets sent with a size larger than the current effective PMTU are known as probe packets.

Packets not intended as probe packets are either fragmented to the current effective PMTU, or the attempt to send fails with an error code. Applications are sometimes provided with a primitive to let them read the Maximum Packet Size (MPS), derived from the current effective PMTU.

Classical PMTUD is subject to protocol failures. One failure arises when traffic using a packet size larger than the actual PMTU is black-holed (all datagrams sent with this size, or larger, are discarded). This could arise when the PTB messages are not delivered back to the sender for some reason (see for example [RFC2923]).

Examples where PTB messages are not delivered include:

* The generation of ICMP messages is usually rate limited. This could result in no PTB messages being generated to the sender (see section 2.4 of [RFC4443])

* ICMP messages can be filtered by middleboxes (including firewalls) [RFC4890]. A stateful firewall could be configured with a policy to block incoming ICMP messages, which would prevent reception of PTB messages to a sending endpoint behind this firewall.

* When the router issuing the ICMP message drops a tunneled packet, the resulting ICMP message will be directed to the tunnel ingress. This tunnel endpoint is responsible for forwarding the ICMP message and also processing the quoted packet within the payload field to remove the effect of the tunnel, and return a correctly formatted ICMP message to the sender [I-D.ietf-intarea-tunnels]. Failure to do this prevents the PTB message reaching the original sender.
Asymmetry in forwarding can result in there being no return route to the original sender, which would prevent an ICMP message being delivered to the sender. This issue can also arise when policy-based routing is used, Equal Cost Multipath (ECMP) routing is used, or a middlebox acts as an application load balancer. An example is where the path towards the server is chosen by ECMP routing depending on bytes in the IP payload. In this case, when a packet sent by the server encounters a problem after the ECMP router, then any resulting ICMP message needs to also be directed by the ECMP router towards the original sender.

There are additional cases where the next hop destination fails to receive a packet because of its size. This could be due to misconfiguration of the layer 2 path between nodes, for instance the MTU configured in a layer 2 switch, or misconfiguration of the Maximum Receive Unit (MRU). If the packet is dropped by the link, this will not cause a PTB message to be sent to the original sender.

Another failure could result if a node that is not on the network path sends a PTB message that attempts to force a sender to change the effective PMTU [RFC8201]. A sender can protect itself from reacting to such messages by utilising the quoted packet within a PTB message payload to validate that the received PTB message was generated in response to a packet that had actually originated from the sender. However, there are situations where a sender would be unable to provide this validation. Examples where validation of the PTB message is not possible include:

When a router issuing the ICMP message implements RFC792 [RFC0792], it is only required to include the first 64 bits of the IP payload of the packet within the quoted payload. There could be insufficient bytes remaining for the sender to interpret the quoted transport information.

Note: The recommendation in RFC1812 [RFC1812] is that IPv4 routers return a quoted packet with as much of the original datagram as possible without the length of the ICMP datagram exceeding 576 bytes. IPv6 routers include as much of the invoking packet as possible without the ICMPv6 packet exceeding 1280 bytes [RFC4443].

The use of tunnels/encryption can reduce the size of the quoted packet returned to the original source address, increasing the risk that there could be insufficient bytes remaining for the sender to interpret the quoted transport information.

Even when the PTB message includes sufficient bytes of the quoted packet, the network layer could lack sufficient context to
validate the message, because validation depends on information about the active transport flows at an endpoint node (e.g., the socket/address pairs being used, and other protocol header information).

* When a packet is encapsulated/tunneled over an encrypted transport, the tunnel/encapsulation ingress might have insufficient context, or computational power, to reconstruct the transport header that would be needed to perform validation.

1.2. Packetization Layer Path MTU Discovery

The term Packetization Layer (PL) has been introduced to describe the layer that is responsible for placing data blocks into the payload of IP packets and selecting an appropriate MPS. This function is often performed by a transport protocol, but can also be performed by other encapsulation methods working above the transport layer.

In contrast to PMTUD, Packetization Layer Path MTU Discovery (PLPMTUD) [RFC4821] does not rely upon reception and validation of PTB messages. It is therefore more robust than Classical PMTUD. This has become the recommended approach for implementing PMTU discovery with TCP.

It uses a general strategy where the PL sends probe packets to search for the largest size of unfragmented datagram that can be sent over a network path. Probe packets are sent with a progressively larger packet size. If a probe packet is successfully delivered (as determined by the PL), then the PLPMTU is raised to the size of the successful probe. If no response is received to a probe packet, the method reduces the probe size. The result of probing with the PLPMTU is used to set the application MPS.

PLPMTUD introduces flexibility in the implementation of PMTU discovery. At one extreme, it can be configured to only perform ICMP black Hole Detection and recovery to increase the robustness of Classical PMTUD, or at the other extreme, all PTB processing can be disabled and PLPMTUD can completely replace Classical PMTUD.

PLPMTUD can also include additional consistency checks without increasing the risk that data is lost when probing to discover the path MTU. For example, information available at the PL, or higher layers, enables received PTB messages to be validated before being utilized.
1.3. Path MTU Discovery for Datagram Services

Section 5 of this document presents a set of algorithms for datagram protocols to discover the largest size of unfragmented datagram that can be sent over a network path. The method described relies on features of the PL described in Section 3 and applies to transport protocols operating over IPv4 and IPv6. It does not require cooperation from the lower layers, although it can utilize PTB messages when these received messages are made available to the PL.

The UDP Usage Guidelines [RFC8085] state "an application SHOULD either use the Path MTU information provided by the IP layer or implement Path MTU Discovery (PMTUD)", but does not provide a mechanism for discovering the largest size of unfragmented datagram that can be used on a network path. Prior to this document, PLPMTUD had not been specified for UDP.

Section 10.2 of [RFC4821] recommends a PLPMTUD probing method for the Stream Control Transport Protocol (SCTP). SCTP utilizes probe packets consisting of a minimal sized HEARTBEAT chunk bundled with a PAD chunk as defined in [RFC4820], but RFC4821 does not provide a complete specification. The present document provides the details to complete that specification.

The Datagram Congestion Control Protocol (DCCP) [RFC4340] requires implementations to support Classical PMTUD and states that a DCCP sender "MUST maintain the MPS allowed for each active DCCP session". It also defines the current congestion control MPS (CCMPS) supported by a network path. This recommends use of PMTUD, and suggests use of control packets (DCCP-Sync) as path probe packets, because they do not risk application data loss. The method defined in this specification could be used with DCCP.

Section 6 specifies the method for a set of transports, and provides information to enable the implementation of PLPMTUD with other datagram transports and applications that use datagram transports.

2. Terminology

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

Other terminology is directly copied from [RFC4821], and the definitions in [RFC1122].
Actual PMTU: The Actual PMTU is the PMTU of a network path between a sender PL and a destination PL, which the DPLPMTUD algorithm seeks to determine.

Black Hole: A Black Hole is encountered when a sender is unaware that packets are not being delivered to the destination 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.
layer. Examples are Ethernet LANs and Internet (or higher) layer and tunnels.

Link MTU: The Link Maximum Transmission Unit (MTU) is the size in bytes of the largest IP packet, including the IP header and payload, that can be transmitted over a link. Note that this could more properly be called the IP MTU, to be consistent with how other standards organizations use the acronym. This includes the IP header, but excludes link layer headers and other framing that is not part of IP or the IP payload. Other standards organizations generally define the link MTU to include the link layer headers.

MAX_PMTU: The MAX_PMTU is the largest size of PLPMTU that DPLPMTUD will attempt to use.

MPS: The Maximum Packet Size (MPS) is the largest size of application data block that can be sent across a network path by a PL. In DPLPMTUD this quantity is derived from the PLPMTU by taking into consideration the size of the lower protocol layer headers. Probe packets generated by DPLPMTUD can have a size larger than the MPS.

MIN_PMTU: The MIN_PMTU is the smallest size of PLPMTU that DPLPMTUD will attempt to use.

Packet: A Packet is the IP header plus the IP payload.

Packetization Layer (PL): The Packetization Layer (PL) is the layer of the network stack that places data into packets and performs transport protocol functions.

Path: The Path is the set of links and routers traversed by a packet between a source node and a destination node by a particular flow.

Path MTU (PMTU): The Path MTU (PMTU) is the minimum of the Link MTU of all the links forming a network path between a source node and a destination node.

PTB_SIZE: The PTB_SIZE is a value reported in a validated PTB message that indicates next hop link MTU of a router along the path.

PLPMTU: The Packetization Layer PMTU is an estimate of the actual PMTU provided by the DPLPMTUD algorithm.

PLPMTUD: Packetization Layer Path MTU Discovery (PLPMTUD), the
method described in this document for datagram PLs, which is an extension to Classical PMTU Discovery.

Probe packet: A probe packet is a datagram sent with a purposely chosen size (typically the current PLPMTU or larger) to detect if packets of this size can be successfully sent end-to-end across the network path.

3. Features Required to Provide Datagram PLPMTUD

TCP PLPMTUD has been defined using standard TCP protocol mechanisms. All of the requirements in [RFC4821] also apply to the use of the technique with a datagram PL. Unlike TCP, some datagram PLs require additional mechanisms to implement PLPMTUD.

There are eight requirements for performing the datagram PLPMTUD method described in this specification:

1. PMTU parameters: A DPLPMTUD sender is RECOMMENDED to provide information about the maximum size of packet that can be transmitted by the sender on the local link (the local Link MTU). It MAY utilize similar information about the receiver when this is supplied (note this could be less than EMTU_R). This avoids implementations trying to send probe packets that can not be transmitted by the local link. Too high of a value could reduce the efficiency of the search algorithm. Some applications also have a maximum transport protocol data unit (PDU) size, in which case there is no benefit from probing for a size larger than this (unless a transport allows multiplexing multiple applications PDUs into the same datagram).

2. PLPMTU: A datagram application using a PL not supporting fragmentation is REQUIRED to be able to choose the size of datagrams sent to the network, up to the PLPMTU, or a smaller value (such as the MPS) derived from this. This value is managed by the DPLPMTUD method. The PLPMTU (specified as the effective PMTU in Section 1 of [RFC1191]) is equivalent to the EMTU_S (specified in [RFC1122]).

3. Probe packets: On request, a DPLPMTUD sender is REQUIRED to be able to transmit a packet larger than the PLPMTU. This is used to send a probe packet. In IPv4, a probe packet MUST be sent with the Don’t Fragment (DF) bit set in the IP header, and without network layer endpoint fragmentation. In IPv6, the probe packet is always sent without source fragmentation (as specified in section 5.4 of [RFC8201]).
4. Processing PTB messages: A DPLPMTUD sender MAY optionally utilize PTB messages received from the network layer to help identify when a network path does not support the current size of probe packet. Any received PTB message MUST be validated before it is used to update the PLPMTU discovery information [RFC8201]. This validation confirms that the PTB message was sent in response to a packet originating by the sender, and needs to be performed before the PLPMTU discovery method reacts to the PTB message. A PTB message MUST NOT be used to increase the PLPMTU [RFC8201].

5. Reception feedback: The destination PL endpoint is REQUIRED to provide a feedback method that indicates to the DPLPMTUD sender when a probe packet has been received by the destination PL endpoint. The mechanism needs to be robust to the possibility that packets could be significantly delayed along a network path. The local PL endpoint at the sending node is REQUIRED to pass this feedback to the sender DPLPMTUD method.

6. Probe loss recovery: It is RECOMMENDED to use probe packets that do not carry any user data. Most datagram transports permit this. If a probe packet contains user data requiring retransmission in case of loss, the PL (or layers above) are REQUIRED to arrange any retransmission/repair of any resulting loss. DPLPMTUD is REQUIRED to be robust in the case where probe packets are lost due to other reasons (including link transmission error, congestion).

7. Probing and congestion control: The DPLPMTUD sender treats isolated loss of a probe packet (with or without a corresponding PTB message) as a potential indication of a PMTU limit for the path. Loss of a probe packet SHOULD NOT be treated as an indication of congestion and the loss SHOULD NOT directly trigger a congestion control reaction [RFC4821].

8. Shared PLPMTU state: The PLPMTU value could also be stored with the corresponding entry in the destination cache and used by other PL instances. The specification of PLPMTUD [RFC4821] states: "If PLPMTUD updates the MTU for a particular path, all Packetization Layer sessions that share the path representation (as described in Section 5.2 of [RFC4821]) SHOULD be notified to make use of the new MTU". Such methods MUST be robust to the wide variety of underlying network forwarding behaviors, PLPMTU adjustments based on shared PLPMTU values should be incorporated in the search algorithms. Section 5.2 of [RFC8201] provides guidance on the caching of PMTU information and also the relation to IPv6 flow labels.
In addition, the following principles are stated for design of a DPLPMTUD method:

* MPS: A method is **REQUIRED** to signal an appropriate MPS to the higher layer using the PL. The value of the MPS can change following a change to the path. It is **RECOMMENDED** that methods avoid forcing an application to use an arbitrary small MPS (PLPMTU) for transmission while the method is searching for the currently supported PLPMTU. Datagram PLs do not necessarily support fragmentation of PDUs larger than the PLPMTU. A reduced MPS can adversely impact the performance of a datagram application.

* Path validation: It is **RECOMMENDED** that methods are robust to path changes that could have occurred since the path characteristics were last confirmed, and to the possibility of inconsistent path information being received.

* Datagram reordering: A method is **REQUIRED** to be robust to the possibility that a flow encounters reordering, or the traffic (including probe packets) is divided over more than one network path.

* When to probe: It is **RECOMMENDED** that methods determine whether the path has changed since it last measured the path. This can help determine when to probe the path again.

4. DPLPMTUD Mechanisms

This section lists the protocol mechanisms used in this specification.

4.1. PLPMTU Probe Packets

The DPLPMTUD method relies upon the PL sender being able to generate probe packets with a specific size. TCP is able to generate these probe packets by choosing to appropriately segment data being sent [RFC4821]. In contrast, a datagram PL that needs to construct a probe packet has to either request an application to send a data block that is larger than that generated by an application, or to utilize padding functions to extend a datagram beyond the size of the application data block. Protocols that permit exchange of control messages (without an application data block) could alternatively prefer to generate a probe packet by extending a control message with padding data.

A receiver needs to be able to distinguish an in-band data block from
any added padding. This is needed to ensure that any added padding is not passed on to an application at the receiver.

This results in three possible ways that a sender can create a probe packet listed in order of preference:

Probing using padding data: A probe packet that contains only control information together with any padding, which is needed to be inflated to the size required for the probe packet. Since these probe packets do not carry an application-supplied data block, they do not typically require retransmission, although they do still consume network capacity and incur endpoint processing.

Probing using application data and padding data: A probe packet that contains a data block supplied by an application that is combined with padding to inflate the length of the datagram to the size required for the probe packet. If the application/transport needs protection from the loss of this probe packet, the application/transport could perform transport-layer retransmission/repair of the data block (e.g., by retransmission after loss is detected or by duplicating the data block in a datagram without the padding data).

Probing using application data: A probe packet that contains a data block supplied by an application that matches the size required for the probe packet. This method requests the application to issue a data block of the desired probe size. If the application/transport needs protection from the loss of an unsuccessful probe packet, the application/transport needs then to perform transport-layer retransmission/repair of the data block (e.g., by retransmission after loss is detected).

A PL that uses a probe packet carrying an application data block, could need to retransmit this application data block if the probe fails. This could need the PL to re-fragment the data block to a smaller packet size that is expected to traverse the end-to-end path (which could utilize endpoint network-layer or PL fragmentation when these are available).

DPLPMTUD MAY choose to use only one of these methods to simplify the implementation.

Probe messages sent by a PL MUST contain enough information to uniquely identify the probe within Maximum Segment Lifetime, while being robust to reordering and replay of probe response and PTB messages.
4.2. Confirmation of Probed Packet Size

The PL needs a method to determine (confirm) when probe packets have been successfully received end-to-end across a network path.

Transport protocols can include end-to-end methods that detect and report reception of specific datagrams that they send (e.g., DCCP and SCTP provide keep-alive/heartbeat features). When supported, this mechanism SHOULD also be used by DPLPMTUD to acknowledge reception of a probe packet.

A PL that does not acknowledge data reception (e.g., UDP and UDP-Lite) is unable itself to detect when the packets that it sends are discarded because their size is greater than the actual PMTU. These PLs need to either rely on an application protocol to detect this loss, or make use of an additional transport method such as UDP-Options [I-D.ietf-tsvwg-udp-options].

Section 6 specifies this function for a set of IETF-specified protocols.

4.3. Detection of Unsupported PLPMTU Size, aka Black Hole Detection

A PL sender needs to reduce the PLPMTU when it discovers the actual PMTU supported by a network path is less than the PLPMTU. This can be triggered when a validated PTB message is received, or by another event that indicates the network path no longer sustains the current packet size, such as a loss report from the PL, or repeated lack of response to probe packets sent to confirm the PLPMTU. Detection is followed by a reduction of the PLPMTU.

This is performed by sending packet probes of size PLPMTU to verify that a network path still supports the last acknowledged PLPMTU size. There are two alternative mechanisms:

* A PL can rely upon a mechanism implemented within the PL to detect excessive loss of data sent with a specific packet size and then conclude that this excessive loss could be a result of an invalid PMTU (as in PLPMTUD for TCP [RFC4821]).

* A PL can use the DPLPMTUD probing mechanism to periodically generate probe packets of the size of the current PLPMTU (e.g., using the confirmation timer Section 5.1.1). A timer tracks whether acknowledgments are received. Successive loss of probes is an indication that the current path no longer supports the PLPMTU (e.g., when the number of probe packets sent without receiving an acknowledgement, PROBE_COUNT, becomes greater than MAX_PROBES).
A PL MAY inhibit sending probe packets when no application data has been sent since the previous probe packet. A PL preferring to use an up-to-date PLPMTU once user data is sent again, MAY choose to continue PLPMTU discovery for each path. However, this may result in additional packets being sent.

When the method detects the current PLPMTU is not supported, DPLPMTUD sets a lower MPS. The PL then confirms that the updated PLPMTU can be successfully used across the path. The PL could need to send a probe packet with a size less than the size of the data block generated by an application. In this case, the PL could provide a way to fragment a datagram at the PL, or use a control packet as the packet probe.

4.4. Response to PTB Messages

This method requires the DPLPMTUD sender to validate any received PTB message before using the PTB information. The response to a PTB message depends on the PTB_SIZE indicated in the PTB message, the state of the PLPMTUD state machine, and the IP protocol being used.

Section 4.4.1 first describes validation for both IPv4 ICMP Unreachable messages (type 3) and ICMPv6 packet too big messages, both of which are referred to as PTB messages in this document.

4.4.1. Validation of PTB Messages

This section specifies utilization of PTB messages.

* A simple implementation MAY ignore received PTB messages and in this case the PLPMTU is not updated when a PTB message is received.

* An implementation that supports PTB messages MUST validate messages before they are further processed.

A PL that receives a PTB message from a router or middlebox, performs ICMP validation as specified in Section 5.2 of [RFC8085][RFC8201]. Because DPLPMTUD operates at the PL, the PL needs to check that each received PTB message is received in response to a packet transmitted by the endpoint PL performing DPLPMTUD.

The PL MUST check the protocol information in the quoted packet carried in an ICMP PTB message payload to validate the message originated from the sending node. This validation includes determining that the combination of the IP addresses, the protocol, the source port and destination port match those returned in the
The validation SHOULD utilize information that it is not simple for an off-path attacker to determine [RFC8085]. For example, by checking the value of a protocol header field known only to the two PL endpoints. A datagram application that uses well-known source and destination ports ought to also rely on other information to complete this validation.

These checks are intended to provide protection from packets that originate from a node that is not on the network path. A PTB message that does not complete the validation MUST NOT be further utilized by the DPLPMTUD method.

PTB messages that have been validated MAY be utilized by the DPLPMTUD algorithm, but MUST NOT be used directly to set the PLPMTU. A method that utilizes these PTB messages can improve the speed at which the algorithm detects an appropriate PLPMTU, compared to one that relies solely on probing. Section 4.4.2 describes this processing.

4.4.2. Use of PTB Messages

A set of checks are intended to provide protection from a router that reports an unexpected PTB_SIZE. The PL also needs to check that the indicated PTB_SIZE is less than the size used by probe packets and larger than minimum size accepted.

This section provides a summary of how PTB messages can be utilized. This processing depends on the PTB_SIZE and the current value of a set of variables:

\[ MIN_{PMTU} < PTB\_SIZE < BASE_{PMTU} \]

* A robust PL MAY enter an error state (see Section 5.2) for an IPv4 path when the PTB_SIZE reported in the PTB message is larger than or equal to 68 bytes and when this is less than the BASE_PMTU.

* A robust PL MAY enter an error state (see Section 5.2) for an IPv6 path when the PTB_SIZE reported in the PTB message is larger than or equal to 1280 bytes and when this is less than the BASE_PMTU.

\[ PTB\_SIZE = PLPMTU \]

* Completes the search for a larger PLPMTU.

\[ PTB\_SIZE > PROBED\_SIZE \]

* Inconsistent network signal.
* PTB message ought to be discarded without further processing (e.g. PLPMTU not modified).

* The information could be utilized as an input to trigger enabling a resilience mode.

BASE_PMTU <= PTB_SIZE < PLPMTU
* Black Hole Detection is triggered and the PLPMTU ought to be set to BASE_PMTU.

* The PL could use the PTB_SIZE reported in the PTB message to initialize a search algorithm.

PLPMTU < PTB_SIZE < PROBED_SIZE
* The PLPMTU continues to be valid, but the last PROBED_SIZE searched was larger than the actual PMTU.

* The PLPMTU is not updated.

* The PL can use the reported PTB_SIZE from the PTB message as the next search point when it resumes the search algorithm.

xxx Author Note: Do we want to specify how to handle PTB Message with PTB_SIZE = 0? xxx

5. Datagram Packetization Layer PMTUD

This section specifies Datagram PLPMTUD (DPLPMTUD). The method can be introduced at various points (as indicated with * in the figure below) in the IP protocol stack to discover the PLPMTU so that an application can utilize an appropriate MPS for the current network path. DPLPMTUD SHOULD NOT be used by an application if it is already used in a lower layer.
The central idea of DPLPMTUD is probing by a sender. Probe packets are sent to find the maximum size of a user message that can be completely transferred across the network path from the sender to the destination.

The following sections identify the components needed for implementation, provides an overview of the phases of operation, and specifies the state machine and search algorithm.

5.1. DPLPMTUD Components

This section describes the timers, constants, and variables of DPLPMTUD.

5.1.1. Timers

The method utilizes up to three timers:

PROBE_TIMER: The PROBE_TIMER is configured to expire after a period longer than the maximum time to receive an acknowledgment to a probe packet. This value MUST NOT be smaller than 1 second, and SHOULD be larger than 15 seconds. Guidance on selection of the timer value are provided in section 3.1.1 of the UDP Usage Guidelines [RFC8085].

If the PL has a path Round Trip Time (RTT) estimate and timely acknowledgements the
PROBE_TIMER can be derived from the PL RTT estimate.

PMTU_RAISE_TIMER: The PMTU_RAISE_TIMER is configured to the period a sender will continue to use the current PLPMTU, after which it re-enters the Search phase. This timer has a period of 600 secs, as recommended by PLPMTUD [RFC4821].

DPLPMTUD MAY inhibit sending probe packets when no application data has been sent since the previous probe packet. A PL preferring to use an up-to-data PMTU once user data is sent again, can choose to continue PMTU discovery for each path. However, this may result in sending additional packets.

CONFIRMATION_TIMER: When an acknowledged PL is used, this timer MUST NOT be used. For other PLs, the CONFIRMATION_TIMER is configured to the period a PL sender waits before confirming the current PLPMTU is still supported. This is less than the PMTU_RAISE_TIMER and used to decrease the PLPMTU (e.g., when a black hole is encountered). Confirmation needs to be frequent enough when data is flowing that the sending PL does not black hole extensive amounts of traffic. Guidance on selection of the timer value are provided in section 3.1.1 of the UDP Usage Guidelines [RFC8085].

DPLPMTUD MAY inhibit sending probe packets when no application data has been sent since the previous probe packet. A PL preferring to use an up-to-data PMTU once user data is sent again, can choose to continue PMTU discovery for each path. However, this may result in sending additional packets.

An implementation could implement the various timers using a single timer.

5.1.2. Constants

The following constants are defined:

MAX_PROBES: The MAX_PROBES is the maximum value of the PROBE_COUNT
counter (see Section 5.1.3). The default value of MAX_PROBES is 10.

**MIN_PMTU:** The MIN_PMTU is the smallest allowed probe packet size. For IPv6, this value is 1280 bytes, as specified in [RFC2460]. For IPv4, the minimum value is 68 bytes.

Note: An IPv4 router is required to be able to forward a datagram of 68 bytes without further fragmentation. This is the combined size of an IPv4 header and the minimum fragment size of 8 bytes. In addition, receivers are required to be able to reassemble fragmented datagrams at least up to 576 bytes, as stated in section 3.3.3 of [RFC1122].

**MAX_PMTU:** The MAX_PMTU is the largest size of PLPMTU. This has to be less than or equal to the minimum of the local MTU of the outgoing interface and the destination PMTU for receiving. An application, or PL, MAY reduce the MAX_PMTU when there is no need to send packets larger than a specific size.

**BASE_PMTU:** The BASE_PMTU is a configured size expected to work for most paths. The size is equal to or larger than the MIN_PMTU and smaller than the MAX_PMTU. In the case of IPv6, this value is 1280 bytes [RFC2460]. When using IPv4, a size of 1200 bytes is RECOMMENDED.

### 5.1.3. Variables

This method utilizes a set of variables:

**PROBED_SIZE:** The PROBED_SIZE is the size of the current probe packet. This is a tentative value for the PLPMTU, which is awaiting confirmation by an acknowledgment.

**PROBE_COUNT:** The PROBE_COUNT is a count of the number of unsuccessful probe packets that have been sent with a size of PROBED_SIZE. The value is initialized to zero when a particular size of PROBED_SIZE is first attempted.

The figure below illustrates the relationship between the packet size constants and variables at a point of time when the DPLPMTUD algorithm performs path probing to increase the size of the PLPMTU. A probe packet has been sent of size PROBED_SIZE. Once this is acknowledged, the PLPMTU will raise to PROBED_SIZE allowing the
DPLPMTUD algorithm to further increase PROBED_SIZE towards the actual PMTU.

```
MIN_PMTU<--------------------------MAX_PMTU
  |       |       |
  v       |       v
BASE_PMTU|       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 v
      and BASE_PMTU v Error |
      confirmed ++-+++

v
PLPMTU v
confirmation failed v

v
PLPMTU v
Search v
confirmed

v
PLPMTU v
Connection or BASE_PMTU confirmed

v
PLPMTU v
Consistent connectivity v
and BASE_PMTU v
confirmed

v
PLPMTU v
Search algorithm completed

v
PLPMTU 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 SEARCH_COMPLETE state, a CONFIRMATION_TIMER periodically resets the PROBE_COUNT and schedules a probe packet with the size of the PLPMTU. If the probe packet fails to be acknowledged after MAX_PROBES attempts, the method enters the BASE state. When used with an acknowledged PL (e.g., SCTP), DPLPMTUD SHOULD NOT continue to generate PLPMTU probes in this state.

ERROR: The ERROR state represents the case where either the network path is not known to support a PLPMTU of at least the BASE_PMTU size or when there is contradictory information about the network path that would otherwise result in excessive variation in the MPS signalled to the higher layer. The state implements a method to mitigate oscillation in the state-event engine. It signals a conservative value of the MPS to the higher layer by the PL. The state is exited when packet probes no longer detect the error or when the PL indicates that connectivity has been lost.

Implementations are permitted to enable endpoint fragmentation if the DPLPMTUD is unable to validate MIN_PMTU within PROBE_COUNT probes. If DPLPMTUD is unable to validate MIN_PMTU the implementation should transition to the DISABLED state.

5.3. Search to Increase the PLPMTU

This section describes the algorithms used by DPLPMTUD to search for a larger PLPMTU.

5.3.1. Probing for a larger PLPMTU

Implementations use a search algorithm across the search range to determine whether a larger PLPMTU can be supported across a network path.

The method discovers the search range by confirming the minimum PLPMTU and then using the probe method to select a PROBED_SIZE less than or equal to MAX_PMTU. MAX_PMTU is the minimum of the local MTU and EMTU_R (learned from the remote endpoint). The MAX_PMTU MAY be reduced by an application that sets a maximum to the size of datagrams it will send.

The PROBE_COUNT is initialized to zero when a probe packet is first sent with a particular size. A timer is used by the search algorithm
to trigger the sending of probe packets of size PROBED_SIZE, larger than the PLPMTU. Each probe packet successfully sent to the remote peer is confirmed by acknowledgement at the PL, see Section 4.1.

Each time a probe packet is sent to the destination, the PROBE_TIMER is started. The timer is canceled when the PL receives acknowledgment that the probe packet has been successfully sent across the path Section 4.1. This confirms that the PROBED_SIZE is supported, and the PROBED_SIZE value is then assigned to the PLPMTU. The search algorithm can continue to send subsequent probe packets of an increasing size.

If the timer expires before a probe packet is acknowledged, the probe has failed to confirm the PROBED_SIZE. Each time the PROBE_TIMER expires, the PROBE_COUNT is incremented, the PROBE_TIMER is reinitialized, and a probe packet of the same size is retransmitted (the replicated probe improve the resilience to loss). The maximum number of retransmissions for a particular size is configured (MAX_PROBES). If the value of the PROBE_COUNT reaches MAX_PROBES, probing will stop, and the PL sender enters the SEARCH_COMPLETE state.

5.3.2. Selection of Probe Sizes

The search algorithm needs to determine a minimum useful gain in PLPMTU. It would not be constructive for a PL sender to attempt to probe for all sizes. This would incur unnecessary load on the path and has the undesirable effect of slowing the time to reach a more optimal MPS. Implementations SHOULD select the set of probe packet sizes to maximize the gain in PLPMTU from each search step.

Implementations could optimize the search procedure by selecting step sizes from a table of common PMTU sizes. When selecting the appropriate next size to search, an implementor ought to also consider that there can be common sizes of MPS that applications seek to use, and their could be common sizes of MTU used within the network.

5.3.3. Resilience to Inconsistent Path Information

A decision to increase the PLPMTU needs to be resilient to the possibility that information learned about the network path is inconsistent. A path is inconsistent, when, for example, probe packets are lost due to other reasons (i.e. not packet size) or due to frequent path changes. Frequent path changes could occur by unexpected "flapping" - where some packets from a flow pass along one path, but other packets follow a different path with different properties.
A PL sender is able to detect inconsistency from the sequence of PLPMTU probes that it sends or the sequence of PTB messages that it receives. When inconsistent path information is detected, a PL sender could use an alternate search mode that clamps the offered MPS to a smaller value for a period of time. This avoids unnecessary loss of packets due to MTU limitation.

5.4. Robustness to Inconsistent Paths

Some paths could be unable to sustain packets of the BASE_PMTU size. To be robust to these paths an implementation could implement the Error State. This allows fallback to a smaller than desired PLPMTU, rather than suffer connectivity failure. This could utilize methods such as endpoint IP fragmentation to enable the PL sender to communicate using packets smaller than the BASE_PMTU.


This section specifies protocol-specific details for datagram PLPMTUD for IETF-specified transports.

The first subsection provides guidance on how to implement the DPLPMTUD method as a part of an application using UDP or UDP-Lite. The guidance also applies to other datagram services that do not include a specific transport protocol (such as a tunnel encapsulation). The following subsections describe how DPLPMTUD can be implemented as a part of the transport service, allowing applications using the service to benefit from discovery of the PLPMTU without themselves needing to implement this method.

6.1. Application support for DPLPMTUD with UDP or UDP-Lite

The current specifications of UDP [RFC0768] and UDP-Lite [RFC3828] do not define a method in the RFC-series that supports PLPMTUD. In particular, the UDP transport does not provide the transport layer features needed to implement datagram PLPMTUD.

The DPLPMTUD method can be implemented as a part of an application built directly or indirectly on UDP or UDP-Lite, but relies on higher-layer protocol features to implement the method [RFC8085].

Some primitives used by DPLPMTUD might not be available via the Datagram API (e.g., the ability to access the PLPMTU cache, or interpret received PTB messages).

In addition, it is desirable that PMTU discovery is not performed by multiple protocol layers. An application SHOULD avoid using DPLPMTUD when the underlying transport system provides this capability. To
use common method for managing the PLPMTU has benefits, both in the ability to share state between different processes and opportunities to coordinate probing.

6.1.1. Application Request

An application needs an application-layer protocol mechanism (such as a message acknowledgement method) that solicits a response from a destination endpoint. The method SHOULD allow the sender to check the value returned in the response to provide additional protection from off-path insertion of data [RFC8085], suitable methods include a parameter known only to the two endpoints, such as a session ID or initialized sequence number.

6.1.2. Application Response

An application needs an application-layer protocol mechanism to communicate the response from the destination endpoint. This response may indicate successful reception of the probe across the path, but could also indicate that some (or all packets) have failed to reach the destination.

6.1.3. Sending Application Probe Packets

A probe packet that may carry an application data block, but the successful transmission of this data is at risk when used for probing. Some applications may prefer to use a probe packet that does not carry an application data block to avoid disruption to data transfer.

6.1.4. Validating the Path

An application that does not have other higher-layer information confirming correct delivery of datagrams SHOULD implement the CONFIRMATION_TIMER to periodically send probe packets while in the SEARCH_COMPLETE state.

6.1.5. Handling of PTB Messages

An application that is able and wishes to receive PTB messages MUST perform ICMP validation as specified in Section 5.2 of [RFC8085]. This requires that the application to check each received PTB messages to validate it is received in response to transmitted traffic and that the reported PTB_SIZE is less than the current probed size (see Section 4.4.2). A validated PTB message MAY be used as input to the 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]


[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC
2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174,

"Path MTU Discovery for IP version 6", STD 87, RFC 8201,
DOI 10.17487/RFC8201, July 2017,

[RFC8261] Tuexen, M., Stewart, R., Jesup, R., and S. Loreto,
"Datagram Transport Layer Security (DTLS) Encapsulation of
SCTP Packets", RFC 8261, DOI 10.17487/RFC8261, November

10.2. Informative References

[I-D.ietf-intarea-tunnels]
Touch, J. and M. Townsley, "IP Tunnels in the Internet
Architecture", draft-ietf-intarea-tunnels-09 (work in
progress), 19 July 2018,
<http://www.ietf.org/internet-drafts/draft-ietf-intarea-
tunnels-09.txt>.

[I-D.ietf-tsvwg-udp-options]
Touch, J., "Transport Options for UDP", draft-ietf-tsvwg-
udp-options-07 (work in progress), 8 March 2019,
<http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-udp-
options-07.txt>.

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5,
RFC 792, DOI 10.17487/RFC0792, September 1981,
<https://www.rfc-editor.org/info/rfc792>.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts -
Communication Layers", STD 3, RFC 1122,
DOI 10.17487/RFC1122, October 1989,

RFC 1812, DOI 10.17487/RFC1812, June 1995,

[RFC2923] Lahey, K., "TCP Problems with Path MTU Discovery",
RFC 2923, DOI 10.17487/RFC2923, September 2000,

Congestion Control Protocol (DCCP)", RFC 4340,
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 supseding 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.
Copyright Notice

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

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

Table of Contents

1. Introduction .................................................. 3
   1.1. Update to RFC 3819 ....................................... 5
   1.2. Scope ................................................... 5
2. Terminology ................................................... 7
3. Modes of Operation ............................................ 9
   3.1. Feed-Forward-and-Up Mode ................................. 9
   3.2. Feed-Up-and-Forward Mode ................................. 11
   3.3. Feed-Backward Mode .................................... 12
   3.4. Null Mode ................................................. 14
4. Feed-Forward-and-Up Mode: Guidelines for Adding Congestion Notification ................................................. 14
   4.1. IP-in-IP Tunnels with Shim Headers ........................ 15
   4.2. Wire Protocol Design: Indication of ECN Support ............. 16
   4.3. Encapsulation Guidelines .................................. 18
   4.4. Decapsulation Guidelines .................................. 20
   4.5. Sequences of Similar Tunnels or Subnets .................... 22
   4.6. Reframing and Congestion Markings ......................... 22
5. Feed-Up-and-Forward Mode: Guidelines for Adding Congestion Notification ................................................. 23
6. Feed-Backward Mode: Guidelines for Adding Congestion Notification ................................................. 24
7. IANA Considerations (to be removed by RFC Editor) .......... 25
8. Security Considerations ....................................... 26
9. Conclusions .................................................... 26
10. Acknowledgements .............................................. 27
11. Comments Solicited ............................................ 27
12. References .................................................... 27
   12.1. Normative References .................................... 27
   12.2. Informative References .................................. 28
Appendix A. Changes in This Version (to be removed by RFC Editor) ................................................. 33
1. Introduction

The benefits of Explicit Congestion Notification (ECN) described in [RFC8087] and summarized below can only be fully realized if support for ECN is added to the relevant subnetwork technology, as well as to IP. When a lower layer buffer drops a packet obviously it does not just drop at that layer; the packet disappears from all layers. In contrast, when active queue management (AQM) at a lower layer marks a packet with ECN, the marking needs to be explicitly propagated up the layers. The same is true if AQM marks the outer header of a packet that encapsulates inner tunnelled headers. Forwarding ECN is not as straightforward as other headers because it has to be assumed ECN may be only partially deployed. If a lower layer header that contains ECN congestion indications is stripped off by a subnet egress that is not ECN-aware, or if the ultimate receiver or sender is not ECN-aware, congestion needs to be indicated by dropping a packet, not marking it.

The purpose of this document is to guide the addition of congestion notification to any subnet technology or tunnelling protocol, so that lower layer AQM algorithms can signal congestion explicitly and it will propagate consistently into encapsulated (higher layer) headers, otherwise the signals will not reach their ultimate destination.

ECN is defined in the IP header (v4 and v6) [RFC3168] to allow a resource to notify the onset of queue build-up without having to drop packets, by explicitly marking a proportion of packets with the congestion experienced (CE) codepoint.

Given a suitable marking scheme, ECN removes nearly all congestion loss and it cuts delays for two main reasons:

- It avoids the delay when recovering from congestion losses, which particularly benefits small flows or real-time flows, making their delivery time predictably short [RFC2884];
- As ECN is used more widely by end-systems, it will gradually remove the need to configure a degree of delay into buffers before they start to notify congestion (the cause of bufferbloat). This is because drop involves a trade-off between sending a timely signal and trying to avoid impairment, whereas ECN is solely a signal not an impairment, so there is no harm triggering it earlier.

Some lower layer technologies (e.g. MPLS, Ethernet) are used to form subnetworks with IP-aware nodes only at the edges. These networks
are often sized so that it is rare for interior queues to overflow. However, until recently this was more due to the inability of TCP to saturate the links. For many years, fixes such as window scaling [RFC1323] (now [RFC7323]) proved hard to deploy. And the Reno variant of TCP has remained in widespread use despite its inability to scale to high flow rates. However, now that modern operating systems are finally capable of saturating interior links, even the buffers of well-provisioned interior switches will need to signal episodes of queuing.

Propagation of ECN is defined for MPLS [RFC5129], and is being defined for TRILL [RFC7780], [I-D.ietf-trill-ecn-support], but it remains to be defined for a number of other subnetwork technologies.

Similarly, ECN propagation is yet to be defined for many tunneling 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 tunneling 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 tunneling protocols, and a companion standards track specification [I-D.ietf-tsvwg-rfc6040-update-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 [RFCXXX], 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 [RFCXXX] 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.

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.

Figure 1: Feed-Forward-and-Up Mode
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-backward mode.

9. Conclusions

Following the guidance in the document enables ECN support to be extended to numerous protocols that encapsulate IP (v4 & v6) in a consistent way, so that IP continues to fulfil its role as an end-to-end interoperability layer. This includes:

- A wide range of tunnelling protocols including those with various forms of shim header between two IP headers, possibly also separated by a L2 header;

- A wide range of subnet technologies, particularly those that work in the same ‘feed-forward-and-up’ mode that is used to support ECN in IP and MPLS.
Guidelines have been defined for supporting propagation of ECN between Ethernet and IP on so-called Layer-3 Ethernet switches, using a ‘feed-up-and-forward’ mode. This approach could enable other subnet technologies to pass ECN signals into the IP layer, even if they do not support ECN natively.

Finally, attempting to add ECN to a subnet technology in feed-backward mode is deprecated except in special cases, due to its likely sluggish response to congestion.

10. Acknowledgements

Thanks to Gorry Fairhurst and David Black for extensive reviews. Thanks also to the following reviewers: Joe Touch, Andrew McGregor, Richard Scheffenegger, Ingemar Johansson, Piers O’Hanlon and Michael Welzl, who pointed out that lower layer congestion notification signals may have different semantics to those in IP. Thanks are also due to the tsvwg chairs, TSV ADs and IETF liaison people such as Eric Gray, Dan Romascanu and Gonzalo Camarillo for helping with the liaisons with the IEEE and 3GPP. And thanks to Georg Mayer and particularly to Erik Guttman for the extensive search and categorisation of any 3GPP specifications that cite ECN specifications.

Bob Briscoe was part-funded by the European Community under its Seventh Framework Programme through the Trilogy project (ICT-216372) for initial drafts and through the Reducing Internet Transport Latency (RITE) project (ICT-317700) subsequently. The views expressed here are solely those of the authors.

11. Comments Solicited

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

12. References

12.1. Normative References


12.2. Informative References


[I-D.ietf-intarea-gue]

[I-D.ietf-nvo3-geneve]

[I-D.ietf-trill-ecn-support]

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

[I-D.ietf-tsvwg-rfc6040update-shim]

[IEEE802.1Q]

[ITU-T.I.371]

[Leiserson85]


Appendix A. Changes in This Version (to be removed by RFC Editor)

From ietf-12 to ietf-13

* Following 3rd tsvwg WGLC:

  + Formalized update to RFC 3819 in its own subsection (1.1) and referred to it in the abstract

  + Scope: Clarified that the specification of alternative ECN semantics using ECT(1) was not in RFC 4774, but rather in RFC 8311, and that the problem with using a DSCP to indicate alternative semantics has issues at domain boundaries as well as tunnels.

  + Terminology: tightened up definitions of ECN-PDU and Not-ECN-PDU, and removed definition of Congestion Baseline, given it was only used once.

  + Mentioned QCN where feedback is first introduced (S.3), referring forward to where it is discussed more deeply (S.4).

  + Clarified that IS-IS solution to adding ECN support to TRILL was not pursued

  + Completely rewrote the rationale for the guideline about a Standard Congestion Monitoring Baseline, to focus on standardization of the otherwise unknown scenario used, rather than the relative usefulness of the info in each approach

  + Explained the re-framing problem better and added fragmentation as another possible cause of the problem

  + Acknowledged new reviewers

  + Updated references, replaced citations of 802.1Qau and 802.1ah with rolled up 802.1Q, and added citations of Fat trees and Clos Networks

  + Numerous other editorial improvements

From ietf-11 to ietf-12

* Updated references

From ietf-10 to ietf-11
Removed short section (was 3) ‘Guidelines for All Cases’ because it was out of scope, being covered by RFC 4774. Expanded the Scope section (1.2) to explain all this. Explained that the default encap/decap rules already support certain alternative semantics, particularly all three of the alternative semantics for ECT(1): equivalent to ECT(0), higher severity than ECT(0), and unmarked but implying different marking semantics from ECT(0).

* Clarified why the QCN example was being given even though not about increment deployment of ECN

* Pointed to the spoofing issue with feed-backward mode from the Security Considerations section, to aid security review.

* Removed any ambiguity in the word ‘transport’ throughout

From ietf-09 to ietf-10

* Updated section 5.1 on "IP-in-IP tunnels with Shim Headers" to be consistent with updates to draft-ietf-tsvwg-rfc6040update-shim.

* Removed reference to the ECN nonce, which has been made historic by RFC 8311

* Removed "Open Issues" Appendix, given all have been addressed.

From ietf-08 to ietf-09

* Updated para in Intro that listed all the IP-in-IP tunnelling protocols, to instead refer to draft-ietf-tsvwg-rfc6040update-shim

* Updated section 5.1 on "IP-in-IP tunnels with Shim Headers" to summarize guidance that has evolved as rfc6040update-shim has developed.

From ietf-07 to ietf-08: Refreshed to avoid expiry. Updated references.

From ietf-06 to ietf-07:

* Added the people involved in liaisons to the acknowledgements.

From ietf-05 to ietf-06:
* Introduction: Added GUE and Geneve as examples of tightly coupled shims between IP headers that cite RFC 6040. And added VXLAN to list of those that do not.

* Replaced normative text about tightly coupled shims between IP headers, with reference to new draft-ietf-tsvwg-rfc6040update-shim

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

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

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

* Updated references.

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

From ietf-04 to ietf-05:

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

* Updated references

From ietf-03 to ietf-04:

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

From ietf-02 to ietf-03:

* Updated references, ad cited RFC4774.

From ietf-01 to ietf-02:

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

* Updated references.

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

From briscoe-03 to 04:

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

* Added applicability to 3GPP TS 36.300.

From briscoe-02 to 03:

* Scope section:
  + Added dependence on correct propagation of traffic class information
  + For the feed-backward mode, deemed multicast and anycast out of scope

* Ensured all guidelines referring to subnet technologies also refer to tunnels and vice versa by adding applicability sentences at the start of sections 4.1, 4.2, 4.3, 4.4, 4.6 and 5.

* Added Security Considerations on ensuring congestion signal fields are classed as immutable and on using end-to-end congestion signal integrity technologies rather than hop-by-hop.

From briscoe-01 to 02:

* Added authors: JK & PT

* Added

  + Section 4.1 "IP-in-IP Tunnels with Tightly Coupled Shim Headers"
  + Section 4.5 "Sequences of Similar Tunnels or Subnets"
  + roadmap at the start of Section 4, given the subsections have become quite fragmented.
  + Section 9 "Conclusions"
* Clarified why transports are starting to be able to saturate interior links

* Under Section 1.1, addressed the question of alternative signal semantics and included multicast & anycast.

* Under Section 3.1, included a 3GPP example.

* Section 4.2. "Wire Protocol Design":
  + Altered guideline 2. to make it clear that it only applies to the immediate subnet egress, not later ones
  + Added a reminder that it is only necessary to check that ECN propagates at the egress, not whether interior nodes mark ECN
  + Added example of how QCN uses 802.1p to indicate support for QCN.

* Added references to Appendix C of RFC6040, about monitoring the amount of congestion signals introduced within a tunnel

* Appendix A: Added more issues to be addressed, including plan to produce a standards track update to IP-in-IP tunnel protocols.

* Updated acks and references

From briscoe-00 to 01:

* Intended status: BCP (was Informational) & updates 3819 added.

* Briefier 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-06

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 September 12, 2019.

Copyright Notice

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

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

Table of Contents

1. Introduction .................................................. 3
   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 ............................................ 16
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 Centre 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 Centre 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:
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 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).
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 mark the ECN field as CE for an increasing proportion of packets, 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-asm-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 \( p_L \) (ToDo cross-ref to new section in l4s-arch that explains the rationale for the square).

That is

\[
p_C \approx \left( \frac{p_L}{k} \right)^2
\]
The constant of proportionality \( k \) does not have to be standardised for interoperability, but a value of 2 is RECOMMENDED.

[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, 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) and Voice-Admit service classes or equivalent local-use DSCP s (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.

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 (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 re-mark the end-to-end L4S 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 ‘Global-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
      (Global-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
   (Global-use or Local-use)
   
   A. PHBs Before L4S ECN Classification
   
   B. PHBs After L4S ECN Classification

6. L4S Experiments

   [I-D.ietf-tsvwg-aqm-dualq-coupled] sets operational and management requirements for experiments with DualQ Coupled AQMs. General operational and management requirements for experiments with L4S congestion controls are given in Section 4 and Section 5 above, e.g. co-existence and scaling requirements, incremental deployment arrangements. The specification of each scalable congestion control will need to include protocol-specific requirements for configuration and monitoring performance during experiments. Appendix A of [RFC5706] provides a helpful checklist.

7. IANA Considerations

   This specification contains no IANA considerations.

8. Security Considerations

   Approaches to assure the integrity of signals using the new identifier are introduced in Appendix C.1.

   The requirement to detect loss in time units prevents the ACK-splitting attacks described in [Savage-TCP].

9. Acknowledgements

   Thanks to Richard Scheffenegger, John Leslie, David Taeht, 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, David Black and Gorry Fairhurst for providing help and reviewing this draft and to Ingemar Johansson for reviewing and
providing substantial text. Appendix A listing the Prague L4S Requirements is based on text authored by Marcelo Bagnulo Braun that was originally an appendix to [I-D.ietf-tsvwg-l4s-arch]. That text was in turn based on the collective output of the attendees listed in the minutes of a ‘bar BoF’ on DCTCP Evolution during IETF-94 [TCPPrague].

The authors’ contributions were part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). Bob Briscoe was also part-funded by the Research Council of Norway through the TimeIn project. The views expressed here are solely those of the authors.

10. References

10.1. Normative References


10.2. Informative References


[I-D.briscoe-tsvwg-l4s-diffserv]
Briscoe, B., "Interactions between Low Latency, Low Loss, Scalable Throughput (L4S) and Differentiated Services", draft-briscoe-tsvwg-l4s-diffserv-02 (work in progress), November 2018.

[I-D.ietf-avtcore-cc-feedback-message]

[I-D.ietf-quic-transport]

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

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

[I-D.ietf-tcpm-rack]

[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.


De Schepper & Briscoe Expires September 12, 2019
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 queuing delay so, over the same path, the RTT becomes lower. Then this problem becomes surprisingly common [TCP-sub-mss-w]. This is because, for the same link capacity, smaller RTT implies a smaller window. For instance, consider a residential setting with an upstream broadband Internet access of 8 Mb/s, assuming a max segment size of 1500 B. Two upstream flows will each have the minimum window of 2 MSS if the RTT is 6ms or less, which is quite common when accessing a nearby data centre. So, any more than two such parallel TCP flows will become unresponsive and increase queuing delay.

Unless scalable congestion controls are required to comply with this requirement from the start, they will frequently become unresponsive, negating the low latency benefit of L4S, for themselves and for others. One possible sub-MSS window mechanism is described in [TCP-sub-mss-w], and other approaches are likely to be feasible.
A.1.7. Measuring Reordering Tolerance in Time Units

Description: A scalable congestion control needs to detect loss by counting in time-based units, which is scalable, rather than counting in units of packets, which is not.

Motivation: If it is known that all L4S senders using a link obey this rule, then link technologies that support L4S can remove the head-of-line blocking delay they have to introduce while trying to keep packets in tight order to avoid triggering loss detection based on counting packets.

End-systems cannot know whether a missing packet is due to loss or reordering, except in hindsight - if it appears later. If senders deem that loss has occurred by counting reordered packets (e.g. the 3 Duplicate ACK rule of Classic TCP), the time over which the network has to keep packets in order scales down as packet rates scale up over the years. In contrast, if senders allow a reordering window in time-based units before they deem there has been a loss, the time over which the network has to keep packets in order stays constant.

The potential benefit for links comes in two parts: i) switching the unit from packet count to time-based; ii) potentially relaxing the amount of time available for re-ordering. The initial switch to time-based units offers no immediate benefit, but as the years progress it stops the reordering requirement becoming tighter. The secondary relaxation might be possible where some transport protocols find they can tolerate more re-ordering (e.g. more than the 3 DupACK rule, perhaps because it was reasonable when packet rates were low, but it is now far too tight).

Tolerance of reordering over a small duration could allow parallel (e.g. bonded-channel) link technologies to relax their need to deliver packets strictly in order. Such links typically give arriving packets a link-level sequence number and introduce delay while buffering packets at the receiving end until they can be delivered in the same order. For radio links, this delay usually includes the time allowed for link-layer retransmissions.

Note, however, that a link technology will only be able to relax its ordering requirement if it is certain that it will not degrade the transport /most/ sensitive to reordering that might use the link. Also note that in some controlled environments no reordering is tolerated by some transports (e.g. RoCEv2 ussed 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-agm-dualq-coupled]) can know for certain that all L4S traffic is using the scalable time-based approach.

This will allow networks to remove the head-of-line blocking delay of resequencing straight away, but only for L4S traffic. Classic traffic will have to wait for many years until incremental deployment of RACK has become near-universal. Nonetheless, experience with RACK will determine how much reordering tolerance networks will be reasonable for L4S traffic.

Performance Optimization as well as Safety Improvement: The delay benefit would be lost if any L4S sender did not follow the time-based approach. Therefore, the time-based approach is made a normative requirement (a necessary safety improvement). Nonetheless, the time-based approach also enables a throughput benefit that a flow can enjoy independently of others (a performance optimization), explained next.

Given the requirement for a scalable congestion control to fall-back to Reno or Cubic on a loss (see Appendix A.1.3), it is important that a scalable congestion control does not deem that a loss has occurred too soon. If, later within the same round trip, an out-of-order acknowledgement fills the gap, the sender would have halved its rate spuriously (as well as retransmitting spuriously). With a RACK-like approach, allowing longer before a loss is deemed to have occurred maintains higher throughput in the presence of reordering (ToDo: Quantify this statement).
On the other hand, it is also important not to wait too long before deeming that a gap is due to a loss (termed a long reordering window), otherwise loss recovery would be slow.

The speed of loss recovery is much more significant for short flows than long, therefore a good compromise would adapt the reordering window; from a small fraction of the RTT at the start of a flow, to a larger fraction of the RTT for flows that continue for many round trips. This is the approach adopted by TCP RACK (Recent ACKnowledgements) [I-D.ietf-tcpm-rack] and recommended for all L4S senders, whether using TCP or another transport protocol.

The requirement to detect loss in time units also prevents the ACK-splitting attacks described in [Savage-TCP].

A.2. Scalable Transport Protocol Optimizations

A.2.1. Setting ECT in TCP Control Packets and Retransmissions

Description: This item only concerns TCP and its derivatives (e.g. SCTP), because the original specification of ECN for TCP precluded the use of ECN on control packets and retransmissions. To improve performance, scalable transport protocols ought to enable ECN at the IP layer in TCP control packets (SYN, SYN-ACK, pure ACKs, etc.) and in retransmitted packets. The same is true for derivatives of TCP, e.g. SCTP.

Motivation: RFC 3168 prohibits the use of ECN on these types of TCP packet, based on a number of arguments. This means these packets are not protected from congestion loss by ECN, which considerably harms performance, particularly for short flows. [I-D.ietf-tcpm-generalized-ecn] counters each argument in RFC 3168 in turn, showing it was over-cautious. Instead it proposes experimental use of ECN on all types of TCP packet as long as AccECN feedback [I-D.ietf-tcpm-accurate-ecn] is available (which is itself a prerequisite for using a scalable congestion control).

A.2.2. Faster than Additive Increase

Description: It would improve performance if scalable congestion controls did not limit their congestion window increase to the traditional additive increase of 1 MSS per round trip [RFC5681] during congestion avoidance. The same is true for derivatives of TCP congestion control, including similar approaches used for real-time media.

Motivation: As currently defined, DCTCP uses the traditional TCP Reno additive increase in congestion avoidance phase. When the available
capacity suddenly increases (e.g. when another flow finishes, or if radio capacity increases) it can take very many round trips to take advantage of the new capacity. In the steady state, DCTCP induces about 2 ECN marks per round trip, so it should be possible to quickly detect when these signals have disappeared and seek available capacity more rapidly. It will of course be necessary to minimize the impact on other flows (classic and scalable).

TCP Cubic was designed to solve this problem, but as flow rates have continued to increase, the delay accelerating into available capacity has become prohibitive. For instance, with RTT=20 ms, to increase flow rate from 100Mb/s to 200Mb/s Cubic takes between 50 and 100 round trips. Every 8x increase in flow rate leads to 2x more acceleration delay.

A.2.3. Faster Convergence at Flow Start

Description: Particularly when a flow starts, scalable congestion controls need to converge (reach their steady-state share of the capacity) at least as fast as classic TCP and preferably faster. This does not just affect TCP Prague, but also the flow start behaviour of any L4S congestion control derived from a Classic transport that uses TCP slow start, including those for real-time media.

Motivation: As an example, a new DCTCP flow takes longer than classic TCP to obtain its share of the capacity of the bottleneck when there are already ongoing flows using the bottleneck capacity. In a data centre environment DCTCP takes about a factor of 1.5 to 2 longer to converge due to the much higher typical level of ECN marking that DCTCP background traffic induces, which causes new flows to exit slow start early [Alizadeh-stability]. In testing for use over the public Internet the convergence time of DCTCP relative to regular TCP is even less favourable [Paced-Chirping]). It is exacerbated by the typically greater mismatch between the link rate of the sending host and typical Internet access bottlenecks, in combination with the shallow ECN marking threshold needed for L4S. This problem is detrimental in general, but would particularly harm the performance of short flows relative to classic TCP.

Appendix B. Alternative Identifiers

This appendix is informative, not normative. It records the pros and cons of various alternative ways to identify L4S packets to record the rationale for the choice of ECT(1) (Appendix B.1) as the L4S identifier. At the end, Appendix B.6 summarises the distinguishing features of the leading alternatives. It is intended to supplement, not replace the detailed text.
The leading solutions all use the ECN field, sometimes in combination with the Diffserv field. Both the ECN and Diffserv fields have the additional advantage that they are no different in either IPv4 or IPv6. A couple of alternatives that use other fields are mentioned at the end, but it is quickly explained why they are not serious contenders.

B.1. ECT(1) and CE codepoints

Definition:

Packets with ECT(1) and conditionally packets with CE would signify L4S semantics as an alternative to the semantics of classic ECN [RFC3168], specifically:

* The ECT(1) codepoint would signify that the packet was sent by an L4S-capable sender;

* Given shortage of codepoints, both L4S and classic ECN sides of an AQM would have to use the same CE codepoint to indicate that a packet had experienced congestion. If a packet that had already been marked CE in an upstream buffer arrived at a subsequent AQM, this AQM would then have to guess whether to classify CE packets as L4S or classic ECN. Choosing the L4S treatment would be a safer choice, because then a few classic packets might arrive early, rather than a few L4S packets arriving late;

* Additional information might be available if the classifier were transport-aware. Then it could classify a CE packet for classic ECN treatment if the most recent ECT packet in the same flow had been marked ECT(0). However, the L4S service ought not to need transport-layer awareness;

Cons:

Consumes the last ECN codepoint: The L4S service is intended to supersede the service provided by classic ECN, therefore using ECT(1) to identify L4S packets could ultimately mean that the ECT(0) codepoint was ‘wasted’ purely to distinguish one form of ECN from its successor;

ECN hard in some lower layers: It is not always possible to support ECN in an AQM acting in a buffer below the IP layer [I-D.ietf-tsvwg-ecn-encap-guidelines]. In such cases, the L4S service would have to drop rather than mark frames even though they might contain an ECN-capable packet. However, such cases would be unusual.
Risk of reordering classic CE packets: Having to classify all CE packets as L4S risks some classic CE packets being wrongly classified as L4S and arriving early, which is a form of reordering. Reordering can cause the TCP sender to retransmit spuriously. However, the risk of spurious retransmissions would be extremely low, because:

1. it is quite unusual to experience more than one bottleneck queue on a path.

2. It would be even more unusual for the first bottleneck to support classic ECN marking and for the second to support L4S ECN marking.

3. even then, reordering would only occur if there was simultaneous mixing of classic and L4S traffic, which would be more unlikely in an access link, which is where most bottlenecks are located;

4. even then, spurious retransmissions would only occur if a contiguous sequence of three or more packets in one classic ECN flow were all CE-marked at the first bottleneck;

5. even then, a spurious retransmission would only occur if the source did not support RACK [I-D.ietf-tcpm-rack], which is already widely supported. Otherwise a whole reordering window within one classic ECN flow would have to be marked CE at the first bottleneck to cause a spurious retransmission.

It is extremely unlikely that a set of 5 eventualities that are each unusual in themselves would all happen simultaneously. But, even if they did, it would only cause spurious retransmission of a packet.

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;

Global-use 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 global-use DSCPs.

Note that today a global-use DSCP gives little more assurance of end-to-end service than a local-use DSCP. In future the global-use range might give more assurance of end-to-end service than local-use, but it is unlikely that either
assurance will be high, particularly given the hosts are included in the end-to-end path.

Not all tunnels: Diffserv codepoints are often not propagated to the outer header when a packet is encapsulated by a tunnel header. DSCPs are propagated to the outer of uniform mode tunnels, but not pipe mode [RFC2983], and pipe mode is fairly common.

ECN hard in some lower layers:: Because this approach uses both the Diffserv and ECN fields, an AQM will only work at a lower layer if both can be supported. If individual network operators wished to deploy an AQM at a lower layer, they would usually propagate an IP Diffserv codepoint to the lower layer, using for example IEEE 802.1p. However, the ECN capability is harder to propagate down to lower layers because few lower layers support it.

Pros:

Could migrate to e2e: If all usage of classic ECN migrates to usage of L4S, the DSCP would become redundant, and the ECN capability alone could eventually identify L4S packets without the interconnection problems of Diffserv detailed above, and without having permanently consumed more than one codepoint in the IP header. Although the DSCP does not generally function as an end-to-end identifier (see above), it could be used initially by individual ISPs to introduce the L4S service for their own locally generated traffic;

B.3. ECN capability alone

Definition:

This approach uses ECN capability alone as the L4S identifier. It is only feasible if classic ECN is not widely deployed. The specific definition of codepoints would be:

* Any ECN codepoint other than Not-ECT would signify an L4S-capable sender;

* ECN codepoints would not be used for classic [RFC3168] ECN, and the classic network service would only be used for Not-ECT packets.

This approach would only be feasible if

A. it was generally agreed that there was little chance of any classic [RFC3168] ECN deployment in any network nodes;
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></td>
<td>eventual</td>
<td>eventual</td>
<td>eventual</td>
</tr>
<tr>
<td>end-to-end</td>
<td>N . .</td>
<td>. ? .</td>
<td>. Y</td>
</tr>
<tr>
<td>tunnels</td>
<td>. O .</td>
<td>. O .</td>
<td>. ?</td>
</tr>
<tr>
<td>reordering</td>
<td>. . Y</td>
<td>. . Y</td>
<td>. Y</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.

Status of This Memo

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

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

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

This Internet-Draft will expire on July 15, 2019.

Copyright Notice

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

Table of Contents

1. Introduction ........................................... 2
2. Definitions and Abbreviations .......................... 4
3. Summary of Architecture Overview ...................... 7
4. Procedural Overview ..................................... 10
   4.1. General ........................................... 10
   4.2. Sender Operation with Sliding Window FEC Codes .... 10
   4.3. Receiver Operation with Sliding Window FEC Codes .... 13
5. Protocol Specification ................................. 15
   5.1. General ........................................... 15
   5.2. FEC Framework Configuration Information .......... 16
   5.3. FEC Scheme Requirements ............................ 16
6. Feedback ................................................ 16
7. Transport Protocols ...................................... 17
8. Congestion Control ....................................... 17
9. Implementation Status .................................... 17
10. Security Considerations ............................... 17
11. Operations and Management Considerations .......... 18
12. IANA Considerations .................................... 18
13. Acknowledgments ........................................ 18
14. References ............................................. 18
   14.1. Normative References ............................... 18
   14.2. Informative References ............................. 19
Appendix A. About Sliding Encoding Window Management (informational) ............................. 20

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)
Internet-Draft           FEC Framework Extension            January 2019

[RFC8406]. This encoding window, either of fixed or variable size, slides over the set of source symbols. FEC encoding is launched whenever needed, from the set of source symbols present in the sliding encoding window at that time. This approach significantly reduces FEC-related latency, since repair symbols can be generated and passed to the transport layer on-the-fly, at any time, and can be regularly received by receivers to quickly recover packet losses. Using sliding window FEC codes is therefore highly beneficial to real-time flows, one of the primary targets of FECFRAME. [RLC-ID] provides an example of such FEC Scheme for FECFRAME, built upon the simple sliding window Random Linear Codes (RLC).

This document is fully backward compatible with [RFC6363]. Indeed:

- this FECFRAME update does not prevent nor compromise in any way the support of block FEC codes. Both types of codes can nicely co-exist, just like different block FEC schemes can co-exist;
- each sliding window FEC Scheme is associated to a specific FEC Encoding ID subject to IANA registration, just like block FEC Schemes;
- any receiver, for instance a legacy receiver that only supports block FEC schemes, can easily identify the FEC Scheme used in a FECFRAME session. Indeed, the FEC Encoding ID that identifies the FEC Scheme is carried in the FEC Framework Configuration Information (see section 5.5 of [RFC6363]). For instance, when the Session Description Protocol (SDP) is used to carry the FEC Framework Configuration Information, the FEC Encoding ID can be communicated in the "encoding-id=" parameter of a "fec-repair-flow" attribute [RFC6364]. This mechanism is the basic approach for a FECFRAME receiver to determine whether or not it supports the FEC Scheme used in a given FECFRAME session;

This document leverages on [RFC6363] and re-uses its structure. It proposes new sections specific to sliding window FEC codes whenever required. The only exception is Section 3 that provides a quick summary of FECFRAME in order to facilitate the understanding of this document to readers not familiar with the concepts and terminology.

2. Definitions and Abbreviations

The following list of definitions and abbreviations is copied from [RFC6363], adding only the Block/sliding window FEC Code and Encoding/Decoding Window definitions (tagged with "ADDED"):

Application Data Unit (ADU): The unit of source data provided as payload to the transport layer. For instance, it can be a
payload containing the result of the RTP packetization of a
compressed video frame.

ADU Flow: A sequence of ADUs associated with a transport-layer flow
identifier (such as the standard 5-tuple {source IP address,
source port, destination IP address, destination port, transport
protocol}).

AL-FEC: Application-layer Forward Error Correction.

Application Protocol: Control protocol used to establish and control
the source flow being protected, e.g., the Real-Time Streaming
Protocol (RTSP).

Content Delivery Protocol (CDP): A complete application protocol
specification that, through the use of the framework defined in
this document, is able to make use of FEC schemes to provide FEC
capabilities.

FEC Code: An algorithm for encoding data such that the encoded data
flow is resilient to data loss. Note that, in general, FEC codes
may also be used to make a data flow resilient to corruption, but
that is not considered in this document.

Block FEC Code: (ADDED) An FEC Code that operates on blocks, i.e.,
for which the input flow MUST be segmented into a sequence of
blocks, FEC encoding and decoding being performed independently
on a per-block basis.

Sliding Window FEC Code: (ADDED) An FEC Code that can generate
repair symbols on-the-fly, at any time, from the set of source
symbols present in the sliding encoding window at that time.
These codes are also known as convolutional codes.

FEC Framework: A protocol framework for the definition of Content
Delivery Protocols using FEC, such as the framework defined in
this document.

FEC Framework Configuration Information: Information that controls
the operation of the FEC Framework.

FEC Payload ID: Information that identifies the contents and
provides positional information of a packet with respect to the
FEC Scheme.

FEC Repair Packet: At a sender (respectively, at a receiver), a
payload submitted to (respectively, received from) the transport
protocol containing one or more repair symbols along with a
Repair FEC Payload ID and possibly an RTP header.

FEC Scheme: A specification that defines the additional protocol
aspects required to use a particular FEC code with the FEC
Framework.

FEC Source Packet: At a sender (respectively, at a receiver), a
payload submitted to (respectively, received from) the transport
protocol containing an ADU along with an optional Explicit Source
FEC Payload ID.

Repair Flow: The packet flow carrying FEC data.

Repair FEC Payload ID: A FEC Payload ID specifically for use with
repair packets.

Source Flow: The packet flow to which FEC protection is to be
applied. A source flow consists of ADUs.

Source FEC Payload ID: A FEC Payload ID specifically for use with
source packets.

Source Protocol: A protocol used for the source flow being
protected, e.g., RTP.

Transport Protocol: The protocol used for the transport of the
source and repair flows, using an unreliable datagram service
such as UDP.

Encoding Window: (ADDED) Set of Source Symbols available at the
sender/coding node that are used to generate a repair symbol,
with a Sliding Window FEC Code.

Decoding Window: (ADDED) Set of received or decoded source and
repair symbols available at a receiver that are used to decode
erased source symbols, with a Sliding Window FEC Code.

Code Rate: The ratio between the number of source symbols and the
number of encoding symbols. By definition, the code rate is such
that 0 < code rate <= 1. A code rate close to 1 indicates that a
small number of repair symbols have been produced during the
encoding process.

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

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

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

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

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

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

3. Summary of Architecture Overview

The architecture of [RFC6363], Section 3, equally applies to this FECFRAME extension and is not repeated here. However, we provide hereafter a quick summary to facilitate the understanding of this document to readers not familiar with the concepts and terminology.
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.

```
+----------------------+
|     Application      |
+----------------------+
    |(6) ADUs
    +----------------------+
    | FEC Framework        |
    |                      |<--------------------------| FEC Scheme |
    |(2) Extract FEC Payload|(5) ADUs                  |(4) FEC Decoding |
    | IDs and pass IDs &    |--------------------------|
    | payloads to FEC       |(3) Explicit Source FEC   |
    | scheme                |            Payload IDs     |
    +----------------------+    Repair FEC Payload IDs |
                      | Source payloads          |
                      | Repair payloads          |
    ^                    |(1) FEC Source           |
                      | and Repair Packets      |
    +----------------------+
    | Transport Protocol    |
    +----------------------+
```

Figure 4: Receiver Operation with Sliding Window FEC Codes
5. Protocol Specification

5.1. General

This section discusses the protocol elements for the FEC Framework specific to Sliding Window FEC schemes. The global formats of source data packets (i.e., [RFC6363], Figure 6) and repair data packets (i.e., [RFC6363], Figures 7 and 8) remain the same with Sliding Window FEC codes. They are not repeated here.
5.2. FEC Framework Configuration Information

The FEC Framework Configuration Information considerations of [RFC6363], Section 5.5, equally applies to this FECFRAME extension and is not repeated here.

5.3. FEC Scheme Requirements

The FEC Scheme requirements of [RFC6363], Section 5.6, mostly apply to this FECFRAME extension and are not repeated here. An exception though is the "full specification of the FEC code", item (4), that is specific to block FEC codes. The following item (4-bis) applies in case of Sliding Window FEC schemes:

4-bis. A full specification of the Sliding Window FEC code

This specification MUST precisely define the valid FEC-Scheme-Specific Information values, the valid FEC Payload ID values, and the valid packet payload sizes (where packet payload refers to the space within a packet dedicated to carrying encoding symbols).

Furthermore, given valid values of the FEC-Scheme-Specific Information, a valid Repair FEC Payload ID value, a valid packet payload size, and a valid encoding window (i.e., a set of source symbols), the specification MUST uniquely define the values of the encoding symbol (or symbols) to be included in the repair packet payload with the given Repair FEC Payload ID value.

Additionally, the FEC Scheme associated to a Sliding Window FEC Code:

- MUST define the relationships between ADUs and the associated source symbols (mapping);
- MUST define the management of the encoding window that slides over the set of ADUs. Appendix A provides non normative hints about what FEC Scheme designers need to consider;
- MUST define the management of the decoding window. This usually consists in managing a system of linear equations (in case of a linear FEC code);

6. Feedback

The discussion of [RFC6363], Section 6, equally applies to this FECFRAME extension and is not repeated here.
7. Transport Protocols

The discussion of [RFC6363], Section 7, equally applies to this FECFRAME extension and is not repeated here.

8. Congestion Control

The discussion of [RFC6363], Section 8, equally applies to this FECFRAME extension and is not repeated here.

9. Implementation Status

Editor’s notes: RFC Editor, please remove this section motivated by RFC 7942 before publishing the RFC. Thanks!

An implementation of FECFRAME extended to Sliding Window codes exists:

- Organisation: Inria
- Description: This is an implementation of FECFRAME extended to Sliding Window codes and supporting the RLC FEC Scheme [RLC-ID]. It is based on: (1) a proprietary implementation of FECFRAME, made by Inria and Expway for which interoperability tests have been conducted; and (2) a proprietary implementation of RLC Sliding Window FEC Codes.
- Maturity: the basic FECFRAME maturity is "production", the FECFRAME extension maturity is "under progress".
- Coverage: the software implements a subset of [RFC6363], as specialized by the 3GPP eMBMS standard [MBMSTS]. This software also covers the additional features of FECFRAME extended to Sliding Window codes, in particular the RLC FEC Scheme.
- Licensing: proprietary.
- Implementation experience: maximum.
- Information update date: March 2018.
- Contact: vincent.roca@inria.fr

10. Security Considerations

This FECFRAME extension does not add any new security consideration. All the considerations of [RFC6363], Section 9, apply to this document as well. However, for the sake of completeness, the
following goal can be added to the list provided in Section 9.1 "Problem Statement" of [RFC6363]:

- Attacks can try to corrupt source flows in order to modify the receiver application’s behavior (as opposed to just denying service).

11. Operations and Management Considerations

This FECFRAME extension does not add any new Operations and Management Consideration. All the considerations of [RFC6363], Section 10, apply to this document as well.

12. IANA Considerations

No IANA actions are required for this document.

A FEC Scheme for use with this FEC Framework is identified via its FEC Encoding ID. It is subject to IANA registration in the "FEC Framework (FECFRAME) FEC Encoding IDs" registry. All the rules of [RFC6363], Section 11, apply and are not repeated here.

13. Acknowledgments

The authors would like to thank Christer Holmberg, David Black, Gorry Fairhurst, and Emmanuel Lochin, Spencer Dawkins, Ben Campbell, Benjamin Kaduk, Eric Rescorla, Adam Roach, and Greg Skinner for their valuable feedback on this document. This document being an extension to [RFC6363], the authors would also like to thank Mark Watson as the main author of that RFC.

14. References

14.1. Normative References


14.2. Informative References


Appendix A. About Sliding Encoding Window Management (informational)

The FEC Framework does not specify the management of the sliding encoding window which is the responsibility of the FEC Scheme. This annex only provides a few informational hints.

Source symbols are added to the sliding encoding window each time a new ADU is available at the sender, after the ADU-to-source-symbol mapping specific to the FEC Scheme.

Source symbols are removed from the sliding encoding window, for instance:

- after a certain delay, when an "old" ADU of a real-time flow times out. The source symbol retention delay in the sliding encoding window should therefore be initialized according to the real-time features of incoming flow(s) when applicable;

- once the sliding encoding window has reached its maximum size (there is usually an upper limit to the sliding encoding window size). In that case the oldest symbol is removed each time a new source symbol is added.

Several considerations can impact the management of this sliding encoding window:

- at the source flows level: real-time constraints can limit the total time source symbols can remain in the encoding window;

- at the FEC code level: theoretical or practical limitations (e.g., because of computational complexity) can limit the number of source symbols in the encoding window;

- at the FEC Scheme level: signaling and window management are intrinsically related. For instance, an encoding window composed of a non-sequential set of source symbols requires an appropriate signaling to inform a receiver of the composition of the encoding window, and the associated transmission overhead can limit the maximum encoding window size. On the opposite, an encoding window always composed of a sequential set of source symbols simplifies signaling: providing the identity of the first source symbol plus their number is sufficient, which creates a fixed and relatively small transmission overhead.
Authors’ Addresses

Vincent Roca
INRIA
Univ. Grenoble Alpes
France

EMail: vincent.roca@inria.fr

Ali Begen
Networked Media
Konya
Turkey

EMail: ali.begen@networked.media
This document describes the L4S architecture for the provision of a new Internet service that could eventually replace best efforts for all traffic: Low Latency, Low Loss, Scalable throughput (L4S). It is becoming common for _all_ (or most) applications being run by a user at any one time to require low latency. However, the only solution the IETF can offer for ultra-low queuing delay is Diffserv, which only favours a minority of packets at the expense of others. In extensive testing the new L4S service keeps average queuing delay under a millisecond for _all_ applications even under very heavy load, without sacrificing utilization; and it keeps congestion loss to zero. It is becoming widely recognized that adding more access capacity gives diminishing returns, because latency is becoming the critical problem. Even with a high capacity broadband access, the reduced latency of L4S remarkably and consistently improves performance under load for applications such as interactive video, conversational video, voice, Web, gaming, instant messaging, remote desktop and cloud-based apps (even when all being used at once over the same access link). The insight is that the root cause of queuing delay is in TCP, not in the queue. By fixing the sending TCP (and other transports) queuing latency becomes so much better than today that operators will want to deploy the network part of L4S to enable new products and services. Further, the network part is simple to deploy - incrementally with zero-config. Both parts, sender and network, ensure coexistence with other legacy traffic. At the same time L4S solves the long-recognized problem with the future scalability of TCP throughput.

This document describes the L4S architecture, briefly describing the different components and how the work together to provide the aforementioned enhanced Internet service.
Status of This Memo

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

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

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

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

Copyright Notice

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

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

Table of Contents

1. Introduction .............................................. 3
2. L4S Architecture Overview ............................... 4
3. Terminology ............................................. 6
4. L4S Architecture Components ............................. 7
5. Rationale ................................................. 9
  5.1. Why These Primary Components? ........................ 9
  5.2. Why Not Alternative Approaches? ....................... 10
6. Applicability ............................................. 13
  6.1. Applications ......................................... 13
  6.2. Use Cases ............................................ 14
  6.3. Deployment Considerations ............................ 15
    6.3.1. Deployment Topology .............................. 16
    6.3.2. Deployment Sequences ............................. 17
    6.3.3. L4S Flow but Non-L4S Bottleneck ................ 19
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 [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 queuing 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 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]. 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: The L4S service traffic needs to be isolated from the queuing latency of the Classic service 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. So a ‘semi-permeable’ membrane is needed that partitions latency but not bandwidth. The Dual Queue Coupled AQM [I-D.ietf-tsvwg-aqm-dualq-coupled] is an example of such a semi-permeable membrane.

Per-flow queuing such as in [RFC8290] could be used, 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 thousands 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) codepoint of the ECN field is a workable solution.

3) Host: Scalable congestion controls already exist. They solve the scaling problem with TCP that was first pointed out in [RFC3649]. The one used most widely (in controlled environments) is Data Centre 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 (SCTP, RTP/RTCP, RMCAT, etc.)
Figure 1: Components of an L4S Solution: 1) Isolation in separate network queues; 2) Packet Identification Protocol; and 3) Scalable Sending Host

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 Centre 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 protocol changes (an unassignment and an assignment) that we describe next:

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--too often to use drops. ‘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. So the
standards track [RFC3168] needs 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].

Network components: 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. An informational appendix of the draft is provided for pseudocode examples of different possible AQM approaches. Initially a zero-config variant of RED called Curvy RED was implemented, tested and documented. 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. For instance, a variant of PIE called Dual PI Squared [PI2] has been implemented and found to perform better than Curvy RED over a wide range of conditions, so it has been documented in another appendix of [I-D.ietf-tsvwg-aqm-dualq-coupled].

Host mechanisms: The L4S architecture includes a number of mechanisms in the end host that we enumerate next:

a. Data Centre 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 more accurate ECN feedback for TCP (AccECN [I-D.ietf-tcpm-accurate-ecn]) is in progress.

* 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.

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 required with Classic ECN
Therefore, in a DualQ AQM, the L4S queue uses a new L4S variant of ECN that is not equivalent to drop [I-D.ietf-tnsnf-dualq]. 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.

5.2. Why Not Alternative Approaches?

All the following approaches address some part of the same problem space as L4S. In each case, it is shown that L4S complements them or improves on them, rather than being a mutually exclusive alternative:
Diffserv: Diffserv addresses the problem of bandwidth apportionment for important traffic as well as queuing latency for delay-sensitive traffic. L4S solely addresses the problem of queuing latency (as well as loss and throughput scaling). Diffserv will still be necessary where important traffic requires priority (e.g. for commercial reasons, or for protection of critical infrastructure traffic) - see [I-D.briscoe-tsvwg-l4s-diffserv]. Nonetheless, if there are Diffserv classes for important traffic, the L4S approach can provide low latency for _all_ traffic within each Diffserv class (including the case where there is only one Diffserv class).

Also, as already explained, Diffserv only works for a small subset of the traffic on a link. It is not applicable when all the applications in use at one time at a single site (home, small business or mobile device) require low latency. Also, because L4S is for all traffic, it needs none of the management baggage (traffic policing, traffic contracts) associated with favouring some packets over others. This baggage has held Diffserv back from widespread end-to-end deployment.

State-of-the-art AQMs: AQMs such as PIE and fq_CoDel give a significant reduction in queuing delay relative to no AQM at all. The L4S work is intended to complement these AQMs, and we definitely do not want to distract from the need to deploy them as widely as possible. Nonetheless, without addressing the large saw-tooothing 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, unlike L4S, it does not support applications that need both capacity-seeking behaviour and very low latency.

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 [I-D.ietf-tcpm-alternativebackoff-ecn] alters the host behaviour in response to ECN marking to utilize a link better and give ECN flows faster throughput, but it 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).
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 in the best effort class:

- Gaming;
- VoIP;
- Video conferencing;
- Web browsing;
- (Adaptive) video streaming;
- Instant messaging.

The significantly lower queuing latency also enables some interactive application functions to be offloaded to the cloud that would hardly even be usable today:

- Cloud based interactive video;
- Cloud based virtual and augmented reality.

The above two applications have been successfully demonstrated with L4S, both running together over a 40 Mb/s broadband access link loaded up with the numerous other latency sensitive applications in the previous list as well as numerous downloads - all sharing the same bottleneck queue simultaneously [L4Sdemo16]. For the former, a panoramic video of a football stadium could be swiped and pinched so that, on the fly, a proxy in the cloud could generate a sub-window of the match video under the finger-gesture control of each user. For the latter, a virtual reality headset displayed a viewport taken from a 360 degree camera in a racing car. The user's head movements controlled the viewport extracted by a cloud-based proxy. In both cases, with 7 ms end-to-end base delay, the additional queuing delay of roughly 1 ms was so low that it seemed the video was generated locally.

Using a swiping finger gesture or head movement to pan a video are extremely latency-demanding actions--far more demanding than VoIP.
Because human vision can detect extremely low delays of the order of single milliseconds when delay is translated into a visual lag between a video and a reference point (the finger or the orientation of the head sensed by the balance system in the inner ear (the vestibular system).

Without the low queuing delay of L4S, cloud-based applications like these would not be credible without significantly more access bandwidth (to deliver all possible video that might be viewed) and more local processing, which would increase the weight and power consumption of head-mounted displays. When all interactive processing can be done in the cloud, only the data to be rendered for the end user needs to be sent.

Other low latency high bandwidth applications such as:

- Interactive remote presence;
- Video-assisted remote control of machinery or industrial processes.

are not credible at all without very low queuing delay. No amount of extra access bandwidth or local processing can make up for lost time.

6.2. Use Cases

The following use-cases for L4S are being considered by various interested parties:

- Where the bottleneck is one of various types of access network: DSL, cable, mobile, satellite
  - Radio links (cellular, WiFi, satellite) that are distant from the source are particularly challenging. The radio link capacity can vary rapidly by orders of magnitude, so it is often desirable to hold a buffer to utilise sudden increases of capacity;
  - Cellular networks are further complicated by a perceived need to buffer in order to make hand-overs imperceptible;
  - Satellite networks generally have a very large base RTT, so even with minimal queuing, overall delay can never be extremely low;
  - Nonetheless, it is certainly desirable not to hold a buffer purely because of the sawteeth of Classic TCP, when it is more than is needed for all the above reasons.
o Private networks of heterogeneous data centres, where there is no single administrator that can arrange for all the simultaneous changes to senders, receivers and network needed to deploy DCTCP:

  * a set of private data centres interconnected over a wide area with separate administrations, but within the same company
  * a set of data centres operated by separate companies interconnected by a community of interest network (e.g. for the finance sector)
  * multi-tenant (cloud) data centres where tenants choose their operating system stack (Infrastructure as a Service - IaaS)

o Different types of transport (or application) congestion control:

  * elastic (TCP/SCTP);
  * real-time (RTP, RMCAT);
  * query (DNS/LDAP).

o Where low delay quality of service is required, but without inspecting or intervening above the IP layer [I-D.smith-encrypted-traffic-management]:

  * mobile and other networks have tended to inspect higher layers in order to guess application QoS requirements. However, with growing demand for support of privacy and encryption, L4S offers an alternative. There is no need to select which traffic to favour for queuing, when L4S gives favourable queuing to all traffic.

o If queuing delay is minimized, applications with a fixed delay budget can communicate over longer distances, or via a longer chain of service functions [RFC7665] or onion routers.

6.3. Deployment Considerations

The DualQ is, in itself, an incremental deployment framework for L4S AQMs so that L4S traffic can coexist with existing Classic "TCP-friendly" traffic. Section 6.3.1 explains why only deploying a DualQ AQM [I-D.ietf-tsvwg-aqm-dualq-coupled] in one node at each end of the access link will realize nearly all the benefit of L4S.

L4S involves both end systems and the network, so Section 6.3.2 suggests some typical sequences to deploy each part, and why there
will be an immediate and significant benefit after deploying just one part.

If an ECN-enabled DualQ AQM has not been deployed at a bottleneck, an L4S flow is required to include a fall-back strategy to Classic behaviour. Section 6.3.3 describes how an L4S flow detects this, and how to minimize the effect of false negative detection.

6.3.1. Deployment Topology

DualQ AQMs will not have to be deployed throughout the Internet before L4S will work for anyone. Operators of public Internet access networks typically design their networks so that the bottleneck will nearly always occur at one known (logical) link. This confines the cost of queue management technology to one place.

The case of mesh networks is different and will be discussed later. But the known bottleneck case is generally true for Internet access to all sorts of different ‘sites’, where the word ‘site’ includes home networks, small-to-medium sized campus or enterprise networks and even cellular devices (Figure 2). Also, this known-bottleneck case tends to be applicable whatever the access link technology; whether xDSL, cable, cellular, line-of-sight wireless or satellite.

Therefore, the full benefit of the L4S service should be available in the downstream direction when the DualQ AQM is deployed at the ingress to this bottleneck link (or links for multihomed sites). And similarly, the full upstream service will be available once the DualQ is deployed at the upstream ingress.
Deployment in mesh topologies depends on how over-booked the core is. If the core is non-blocking, or at least generously provisioned so that the edges are nearly always the bottlenecks, it would only be necessary to deploy the DualQ AQM at the edge bottlenecks. For example, some data-centre networks are designed with the bottleneck in the hypervisor or host NICs, while others bottleneck at the top-of-rack switch (both the output ports facing hosts and those facing the core).

The DualQ would eventually also need to be deployed at any other persistent bottlenecks such as network interconnections, e.g. some public Internet exchange points and the ingress and egress to WAN links interconnecting data-centres.

6.3.2. Deployment Sequences

For any one L4S flow to work, it requires 3 parts to have been deployed. This was the same deployment problem that ECN faced [RFC8170] so we have learned from this.

Firstly, L4S deployment exploits the fact that DCTCP already exists on many Internet hosts (Windows, FreeBSD and Linux); both servers and clients. Therefore, just deploying DualQ AQM at a network bottleneck immediately gives a working deployment of all the L4S parts. DCTCP needs some safety concerns to be fixed for general use over the public Internet (see Section 2.3 of [I-D.ietf-tsvwg-ecn-l4s-id]), but
DCTCP is not on by default, so these issues can be managed within controlled deployments or controlled trials.

Secondly, the performance improvement with L4S is so significant that it enables new interactive services and products that were not previously possible. It is much easier for companies to initiate new work on deployment if there is budget for a new product trial. If, in contrast, there were only an incremental performance improvement (as with Classic ECN), spending on deployment tends to be much harder to justify.

Thirdly, the L4S identifier is defined so that initially network operators can enable L4S exclusively for certain customers or certain applications. But this is carefully defined so that it does not compromise future evolution towards L4S as an Internet-wide service. This is because the L4S identifier is defined not only as the end-to-end ECN field, but it can also optionally be combined with any other packet header or some status of a customer or their access link [I-D.ietf-tsvwg-ecn-l4s-id]. Operators could do this anyway, even if it were not blessed by the IETF. However, it is best for the IETF to specify that they must use their own local identifier in combination with the IETF’s identifier. Then, if an operator enables the optional local-use approach, they only have to remove this extra rule to make the service work Internet-wide - it will already traverse middleboxes, peerings, etc.

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 [BBR]. 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 [BBR] 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 [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 will continually widen L4S applicability.

Classic ECN support is starting to materialize (in the upstream of some home routers as of early 2017), so 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]).

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 [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 re-mark 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. Allowing operators to use an additional local classifier is intended to remove any incentive 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 requires.
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 standardisation of dynamic behaviour (cf. TCP slow-start) and self-interest 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.

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. 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 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


(Under submission)


[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-01 (work in progress), July 2018.


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.

<table>
<thead>
<tr>
<th>RFC/Doc Title</th>
<th>RFC/Doc Number</th>
<th>Date</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFC8257 Data Center TCP (DCTCP): TCP Congestion Control for Data Centers</td>
<td>RFC 8257</td>
<td>October 2017</td>
<td><a href="https://www.rfc-editor.org/info/rfc8257">https://www.rfc-editor.org/info/rfc8257</a></td>
</tr>
<tr>
<td>RFC8290 The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm</td>
<td>RFC 8290</td>
<td>January 2018</td>
<td><a href="https://www.rfc-editor.org/info/rfc8290">https://www.rfc-editor.org/info/rfc8290</a></td>
</tr>
<tr>
<td>RFC8312 CUBIC for Fast Long-Distance Networks</td>
<td>RFC 8312</td>
<td>February 2018</td>
<td><a href="https://www.rfc-editor.org/info/rfc8312">https://www.rfc-editor.org/info/rfc8312</a></td>
</tr>
</tbody>
</table>

Briscoe, et al. Expires April 25, 2019
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/iccr" combination refers to the procedure typically used for congestion control changes, where tcpm owns the approval decision, but uses the iccr for expert review [NewCC_Proc];

TCP: Applicable to all forms of TCP congestion control;

DCTCP: Applicable to Data Centre TCP as currently used (in controlled environments);

DCTCP bis: Applicable to an future Data Centre TCP congestion control intended for controlled environments;

XXX Prague: Applicable to a Scalable variant of XXX (TCP/SCTP/RMCAT) congestion control.
<table>
<thead>
<tr>
<th>Req #</th>
<th>Requirement</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ARCHITECTURE</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>L4S IDENTIFIER</td>
<td>[I-D.ietf-tsvwg-ecn-l4s-id]</td>
</tr>
<tr>
<td>2</td>
<td>DUAL QUEUE AQM</td>
<td>[I-D.ietf-tsvwg-aqm-dualq-coupled]</td>
</tr>
<tr>
<td>3</td>
<td>Suitable ECN Feedback</td>
<td>[I-D.ietf-tcpm-accurate-ecn], [I-D.stewart-tsvwg-sctpecn].</td>
</tr>
<tr>
<td></td>
<td>SCALABLE TRANSPORT - 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, [RFC8257]</td>
</tr>
<tr>
<td>4-2</td>
<td>Fall back to Reno/Cubic if classic ECN bottleneck detected</td>
<td>[I-D.ietf-tsvwg-ecn-l4s-id] S.2.3</td>
</tr>
<tr>
<td>4-3</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 Congestion Window for Small Round Trip Times</td>
<td>[I-D.ietf-tsvwg-ecn-l4s-id] S.2.3, [TCP-sub-mss-w]</td>
</tr>
<tr>
<td></td>
<td>SCALABLE TRANSPORT - PERFORMANCE ENHANCEMENTS</td>
<td></td>
</tr>
<tr>
<td>5-1</td>
<td>Setting ECT in TCP Control Packets and Retransmissions</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></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></td>
</tr>
<tr>
<td>4-2</td>
<td>tcpm/</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>iccrg?</td>
<td></td>
</tr>
<tr>
<td>4-3</td>
<td>tcpm/</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>iccrg?</td>
<td></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/</td>
<td></td>
</tr>
<tr>
<td></td>
<td>iccrg?</td>
<td></td>
</tr>
<tr>
<td>5-3</td>
<td>tcpm/</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>iccrg?</td>
<td></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. Remarking to other DSCPs/PHBs ............................. 8
9. Multicast Considerations .................................. 9
10. The Update to RFC 4594 ................................... 10
11. The Update to RFC 8325 ................................... 12
12. The Update to draft-ietf-rtsec-web-qos ................... 12
13. IANA Considerations ...................................... 14
14. Security Considerations .................................. 14
15. References .................................................. 15
   15.1. Normative References ................................. 15
   15.2. Informative References ............................... 15
Appendix A. History of the LE PHB ............................ 17
Appendix B. Acknowledgments ................................ 18
1. Introduction

This document defines a Differentiated Services per-hop behavior [RFC2474] called "Lower Effort" (LE), which is intended for traffic of sufficiently low urgency that all other traffic takes precedence over the LE traffic in consumption of network link bandwidth. Low urgency traffic has a low priority for timely forwarding, which does not necessarily imply that it is generally of minor importance. From this viewpoint, it can be considered as a network equivalent to a background priority for processes in an operating system. There may or may not be memory (buffer) resources allocated for this type of traffic.

Some networks carry packets that ought to consume network resources only when no other traffic is demanding them. In this point of view, packets forwarded by the LE PHB scavenge otherwise unused resources only, which led to the name "scavenger service" in early Internet2 deployments (see Appendix A). Other commonly used names for LE PHB type services are "Lower-than-best-effort" or "Less-than-best-effort". In summary, with the mentioned feature above, the LE PHB has two important properties: it should scavenge residual capacity and it must be preemptable by the default PHB (or other elevated PHBs) in case they need more resources. Consequently, the effect of this type of traffic on all other network traffic is strictly limited ("no harm" property). This is distinct from "best-effort" (BE) traffic since the network makes no commitment to deliver LE packets. In contrast, BE traffic receives an implied "good faith" commitment of at least some available network resources. This document proposes a Lower Effort Differentiated Services per-hop behavior (LE PHB) for handling this "optional" traffic in a differentiated services node.

2. Requirements Language

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

3. Applicability

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

Just like best-effort traffic, LE traffic SHOULD be congestion controlled (i.e., use a congestion controlled transport or implement an appropriate congestion control method [RFC2914] [RFC8085]). Since LE traffic could bestarved 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.
For backup traffic or non-time critical synchronization or mirroring traffic.

- For content distribution transfers between caches.

- For preloading or prefetching objects from web sites.

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

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

- 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. Remarking to other DSCPs/PHBs

"DSCP bleaching", i.e., setting the DSCP to ‘000000’ (default PHB) is NOT RECOMMENDED for this PHB. This may cause effects that are in contrast to the original intent in protecting BE traffic from LE traffic (no harm property). In the case that a DS domain does not support the LE PHB, its nodes SHOULD treat LE marked packets with the default PHB instead (by mapping the LE DSCP to the default PHB), but they SHOULD do so without remarking to DSCP ‘000000’. The reason for this is that later traversed DS domains may then have still the possibility to treat such packets according to the LE PHB.

Operators of DS domains that forward LE traffic within the BE aggregate need to be aware of the implications, i.e., induced congestion situations and quality-of-service degradation of the
original BE traffic. In this case, the LE property of not harming other traffic is no longer fulfilled. To limit the impact in such cases, traffic policing of the LE aggregate MAY be used.

In the case that LE marked packets are effectively carried within the default PHB (i.e., 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 the 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:

<table>
<thead>
<tr>
<th>Low-Priority Data</th>
<th>CS1</th>
<th>001000</th>
<th>Any flow that has no BW assurance</th>
</tr>
</thead>
</table>

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>
</table>

<table>
<thead>
<tr>
<th>Low-Priority Data (legacy)</th>
<th>CS1</th>
<th>RFC 3662</th>
<th>1</th>
<th>AC_BK (Background)</th>
</tr>
</thead>
</table>

12. The Update to draft-ietf-tsvwg-rtcweb-qos

Section 5 of [I-D.ietf-tsvwg-rtcweb-qos] describes the Recommended DSCP Values for WebRTC Applications
This update to [I-D.ietf-tsvwg-rtcweb-qos] replaces all occurrences of CS1 with LE in Table 1:

<table>
<thead>
<tr>
<th>Flow Type</th>
<th>Very Low</th>
<th>Low</th>
<th>Medium</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
<td>LE (1)</td>
<td>DF (0)</td>
<td>EF (46)</td>
<td>EF (46)</td>
</tr>
<tr>
<td>Interactive Video with or without Audio</td>
<td>LE (1)</td>
<td>DF (0)</td>
<td>AF42, AF43 (36, 38)</td>
<td>AF41, AF42 (34, 36)</td>
</tr>
<tr>
<td>Non-Interactive Video with or without Audio</td>
<td>LE (1)</td>
<td>DF (0)</td>
<td>AF32, AF33 (28, 30)</td>
<td>AF31, AF32 (26, 28)</td>
</tr>
<tr>
<td>Data</td>
<td>LE (1)</td>
<td>DF (0)</td>
<td>AF11</td>
<td>AF21</td>
</tr>
</tbody>
</table>

and updates the following paragraph:

"The above table assumes that packets marked with CS1 are treated as "less than best effort", such as the LE behavior described in [RFC3662]. However, the treatment of CS1 is implementation dependent. If an implementation treats CS1 as other than "less than best effort", then the actual priority (or, more precisely, the per-hop-behavior) of the packets may be changed from what is intended. It is common for CS1 to be treated the same as DF, so applications and browsers using CS1 cannot assume that CS1 will be treated differently than DF [RFC7657]. However, it is also possible per [RFC2474] for CS1 traffic to be given better treatment than DF, thus caution should be exercised when electing to use CS1. This is one of the cases where marking packets using these recommendations can make things worse."

as follows:

"The above table assumes that packets marked with LE are treated as lower effort (i.e., "less than best effort"), such as the LE behavior described in [RFCXXXX]. However, the treatment of LE is implementation dependent. If an implementation treats LE as other than "less than best effort", then the actual priority (or, more precisely, the per-hop-behavior) of the packets may be changed from what is intended. It is common for LE to be treated the same as DF, so applications and browsers using LE cannot assume that LE will be treated differently than DF [RFC7657]. During development of this document, the CS1 DSCP was recommended for "very low" application
priority traffic; implementations that followed that recommendation SHOULD be updated to use the LE DSCP instead of the CS1 DSCP."

13.  IANA Considerations

This document assigns the Differentiated Services Field Codepoint (DSCP) ‘000001’ from the Differentiated Services Field Codepoints (DSCP) registry (https://www.iana.org/assignments/dscp-registry/dscp-registry.xhtml) (Pool 3, Codepoint Space xxxx01, Standards Action) to the LE PHB. This document suggests to use a DSCP from Pool 3 in order to avoid problems for other PHB marked flows to become accidentally remarked as LE PHB, e.g., due to partial DSCP bleaching. See [RFC8436] for re-classifying Pool 3 for Standards Action.

IANA is requested to update the registry as follows:

- Name: LE
- Value (Binary): 000001
- Value (Decimal): 1
- Reference: [RFC number of this memo]

14.  Security Considerations

There are no specific security exposures for this PHB. Since it defines a new class of low forwarding priority, remarking other traffic as LE traffic may lead to quality-of-service degradation of such traffic. Thus, any attacker that is able to modify the DSCP of a packet to LE may carry out a downgrade attack. See the general security considerations in [RFC2474] and [RFC2475].

With respect to privacy, an attacker could use the information from the DSCP to infer that the transferred (probably even encrypted) content is considered of low priority or low urgency by a user, in case the DSCP was set on the user’s request. On the one hand, this disclosed information is useful only if correlation with metadata (such as the user’s IP address) and/or other flows reveal user identity. On the other hand, it might help an observer (e.g., a state level actor) who is interested in learning about the user’s behavior from observed traffic: LE marked background traffic (such as software downloads, operating system updates, or telemetry data) may be less interesting for surveillance than general web traffic. Therefore, the LE marking may help the observer to focus on potentially more interesting traffic (however, the user may exploit this particular assumption and deliberately hide interesting traffic in the LE aggregate). Apart from such considerations, the impact of
disclosed information by the LE DSCP is likely negligible in most cases given the numerous traffic analysis possibilities and general privacy threats (e.g., see [RFC6973]).

15. References

15.1. Normative References


15.2. Informative References


Appendix A. History of the LE PHB

A first version of this PHB was suggested by Roland Bless and Klaus Wehrle in September 1999 [draft-bless-diffserv-lbe-phb-00], named "A Lower Than Best-Effort Per-Hop Behavior". After some discussion in
the Diffserv Working Group Brian Carpenter and Kathie Nichols proposed a "bulk handling" per-domain behavior and believed a PHB was not necessary. Eventually, "Lower Effort" was specified as per-domain behavior and finally became [RFC3662]. More detailed information about its history can be found in Section 10 of [RFC3662].

There are several other names in use for this type of PHB or associated service classes. Well-known is the QBone Scavenger Service (QBSS) that was proposed in March 2001 within the Internet2 QoS Working Group. Alternative names are "Lower-than-best-effort" [carlberg-lbe-2001] or "Less-than-best-effort" [chown-lbe-2003].

Appendix B. Acknowledgments

Since text is partially borrowed from earlier Internet-Drafts and RFCs the co-authors of previous specifications are acknowledged here: Kathie Nichols and Klaus Wehrle. David Black, Olivier Bonaventure, Spencer Dawkins, Toerless Eckert, Gorry Fairhurst, Ruediger Geib, and Kyle Rose provided helpful comments and (partially also text) suggestions.

Appendix C. Change History

This section briefly lists changes between Internet-Draft versions for convenience.

Changes in Version 10: (incorporated comments from IESG discussion as follows)

- Appended "for Differentiated Services" to the title as suggested by Alexey.
- Addressed Deborah Brungard’s discuss: changed phrase to "However, non-LE traffic (e.g., BE traffic) SHOULD NOT be remarked to LE." with additional explanation as suggested by Gorry.
- Fixed the sentence "An LE PHB SHOULD NOT be used for a customer’s "normal Internet" traffic nor should packets be "downgraded" to the LE PHB instead of being dropped, particularly when the packets are unauthorized traffic." according to Alice’s and Mirja’s comments.
- Made reference to RFC8174 normative.
- Added hint for the RFC editor to apply changes from section Section 12 and to delete it afterwards.
Incorporated Mirja’s and Benjamin’s suggestions.

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

Changes in Version 09:

Incorporated comments from IETF Last Call:

* from Olivier Bonaventure: added a bit of text for session resumption and congestion control aspects as well as ECN usage.

* from Kyle Rose: Revised privacy considerations text in Security Considerations Section

Changes in Version 08:

revised two sentences as suggested by Spencer Dawkins

Changes in Version 07:

revised some text for clarification according to comments from Spencer Dawkins

Changes in Version 06:

added Multicast Considerations section with input from Toerless Eckert

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

Changes in Version 05:

added scavenger service class into abstract

added some more history

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

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

revised text on congestion control in case of remarking to BE

added reference to DSCP measurement talk @IETF99

small typo fixes
Changes in Version 04:

- Several editorial changes according to review from Gorry Fairhurst
- Changed the section structure a bit (moved subsections 1.1 and 1.2 into own sections 3 and 7 respectively)
- Updated section 2 on requirements language
- Added updates to RFC 8325
- Tried to be more explicit what changes are required to RFCs 4594 and 8325

Changes in Version 03:

- Changed recommended codepoint to 000001
- Added text to explain the reasons for the DSCP choice
- Removed LE-min, LE-strict discussion
- Added one more potential use case: reporting errors or telemetry data from OSs
- Added privacy considerations to the security section (not worth an own section I think)
- Changed IANA considerations section

Changes in Version 02:

- Applied many editorial suggestions from David Black
- Added Multicast traffic use case
- Clarified what is required for deployment in section 1.2 (Deployment Considerations)
- Added text about implementations using AQMs and ECN usage
- Updated IANA section according to David Black’s suggestions
- Revised text in the security section
- Changed copyright Notice to pre5378Trust200902

Changes in Version 01:
Now obsoletes RFC 3662.

Tried to be more precise in section 1.1 (Applicability) according to R. Geib’s suggestions, so rephrased several paragraphs. Added text about congestion control.

Change section 2 (PHB Description) according to R. Geib’s suggestions.

Added RFC 2119 language to several sentences.

Detailed the description of remarking implications and recommendations in Section 8.

Added Section 10 to explicitly list changes with respect to RFC 4594, because this document will update it.

Appendix D. Note to RFC Editor

This section lists actions for the RFC editor during final formatting.

Apply the suggested changes of section Section 12 and add a normative reference in draft-ietf-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
Stream Control Transmission Protocol (SCTP) Network Address Translation Support
draft-ietf-tsvwg-natsupp-12.txt

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 NATs 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 3, 2019.
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 End Points and NATs ......................... 17
  6.1. Overview ....................................................... 17
  6.2. Association Setup Considerations ............................. 18
  6.3. Handling of Internal Port Number and Verification Tag
       Collisions ..................................................... 18
  6.4. Handling of Internal Port Number Collisions ............... 19
  6.5. Handling of Missing State ................................... 20
  6.6. Handling of Fragmented SCTP Packets ....................... 22
  6.7. Multi-Point Traversal Considerations ....................... 22
7. Various Examples of NAT Traversals ................................ 23
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 NATs 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 NATs is to use UDP encapsulation as specified in [RFC6951].

This document describes an SCTP specific variant NAT and specific packets and procedures to help NATs provide similar features of NAPT in the single-point and multi-point traversal scenario. An SCTP implementation supporting this extension will follow these procedures to 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 NATs 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</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>Not Support</td>
<td>No Support</td>
<td>Limited</td>
</tr>
<tr>
<td>Support</td>
<td>Not 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>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 we can see that when a NAT does not support the extension no communication can occur. This is because for the most part of the current situation i.e. SCTP packets sent externally from behind a NAT are discarded by the NAT. In some cases, where the NAT supports the feature but one of the two external 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 box which 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:

A variation of this case is shown below, i.e., multiple NATs in a single path:
Serial NATs scenario

In this single point traversal scenario, we must acknowledge that while one of the main benefits of SCTP multi-homing is redundant paths, 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 (or NATs) in this case sees all the packets of the SCTP association.

4.1.2. Multi Point Traversal

This case involves multiple NATs and each NAT only sees some of the packets in the SCTP association. An example is shown below:

Parallel NATs 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 NATs 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 we assume that we have multiple SCTP capable hosts behind a NAT which has one Public-Address. Furthermore we are focusing in this section on the single point traversal scenario.

The modification of SCTP packets sent to the public Internet is easy. 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 box to handle incoming packets, which is discussed later.

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.

For the SCTP NAT processing the NAT box 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.

The entries in this 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.

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

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.

The processing of other incoming SCTP packets 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 NATs and received by SCTP end points. Section 5.3 describes parameters sent by SCTP end points and used by NATs and SCTP end points.

5.1. Modified Chunks

This section presents existing chunks defined in [RFC4960] that are modified by this document.

5.1.1. Extended ABORT Chunk

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 6    | Reserved  |M|T|           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \        
/                   zero or more Error Causes                   /        
\                                                               \        
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
The ABORT chunk is extended to add the new ‘M-bit’. The M-bit indicates to the receiver of the ABORT chunk that the chunk was not generated by the peer SCTP endpoint, but instead by a middle box.

[NOTE:

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:

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

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:

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

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:

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

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:

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

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:

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

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:
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 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>
</tr>
<tr>
<td>+---------------------------------------------+</td>
<td>Cause Code = 0x00B1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+---------------------------------------------+</td>
<td>Cause Length = Variable</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\ 
| Incomplete Packet |
| +---------------------------------------------+ |

Cause Code: 2 bytes (unsigned integer)
This field holds the IANA defined cause code for the 'Missing State' Error Cause. The suggested value of this field for IANA is 0x00B1.

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

Incoming Packet: variable length
The Cause-Specific Information is filled with the IPv4 or IPv6 packet that caused this error. The IPv4 or IPv6 header MUST be included. Note that if the packet will not fit in the ERROR chunk or ABORT chunk being sent then the bytes that do not fit are truncated.
5.2.3. Port Number Collision Error Cause

<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>
</tbody>
</table>

Cause Code: 2 bytes (unsigned integer)
This field holds the IANA defined cause code for the ‘Port Number Collision’ Error Cause. The suggested value of this field for IANA is 0x00B2.

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. The suggested value of this field for IANA is 0xC007.

Parameter Length: 2 bytes (unsigned integer)
This field holds the length in bytes of the parameter. The value MUST be 4.

[NOTE: 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 Length = 16 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| ASCONF-Request Correlation ID |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Internal Verification Tag |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| External Verification Tag |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
Parameter Type: 2 bytes (unsigned integer)
This field holds the IANA defined parameter type for the VTags Parameter. The suggested value of this field for IANA is 0xC008.

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: 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 End Points and NATs

6.1. Overview

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 6.2.
o More than one host behind a NAT may pick 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 6.3.

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 6.4.

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 6.5.

o NAT boxes need to deal with SCTP packets being fragmented at the IP layer. This is discussed in Section 6.6.

o An SCTP endpoint may be behind two NATs providing redundancy. The method to set up this scenario is discussed in Section 6.7.

Each of these mechanisms requires additional chunks and parameters, defined in this document, and possibly modified handling procedures from those specified in [RFC4960].

6.2. Association Setup Considerations

The association setup procedure defined in [RFC4960] allows multi-homed SCTP end points to exchange its IP-addresses by using IPv4 or IPv6 address parameters in the INIT and INIT-ACK chunks. However, this can't be used when NATs 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 6.7 MUST be used.

The INIT and INIT-ACK chunk SHOULD contain the Disable Restart parameter defined in Section 5.3.1.

6.3. 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 box 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 box 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.

However, in this unlikely event the NAT box 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 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.

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).

6.4. 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.

But 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 MUST do the following:

- Include a Disable Restart parameter in the INIT-ACK to inform the client that it will support the feature.
- 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.

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 MUST do the following when processing an INIT:

- If the INIT is destined to an external address and port for which the NAT has no outbound connection, allow the INIT creating an internal mapping table.
- If the INIT matches the external address and port of an already existing connection, validate that the external server supports the Disable Restart feature, if it does allow the INIT to be forwarded.
- If the external server does not support the Disable Restart extension the NAT 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.

6.5. Handling of Missing State

If the NAT box 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).

Upon reception of this ERROR chunk by an SCTP endpoint the receiver SHOULD take the following actions:

- Validate that the verification tag is reflected by looking at the VTag that would have been included in the outgoing packet.
- Validate that the peer of the SCTP association supports the dynamic address extension, if it does not discard the incoming ERROR chunk.
- 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 can recover the appropriate state. The procedures for generating an ASCONF chunk can be found in [RFC5061].

If the NAT box receives a packet for which it has no NAT table entry and the packet contains an ASCONF chunk with the VTags parameter, the NAT box MUST update its NAT table according to the verification tags in the VTags parameter and the optional Disable Restart parameter.

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 MUST do the following:

- Validate that the verification tag is reflected by looking at the VTag that would have been included in the outgoing packet.

- Validate that the peer of the SCTP association supports the dynamic address extension, if it does not discard the incoming ERROR chunk.

- If the association is attempting to add an address (i.e. following the procedures in Section 6.7) 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 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, then the endpoint MUST abort the association. This would occur only if the local NAT 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 looses its state).

6.6. Handling of Fragmented SCTP Packets

A NAT box 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 box and the IP header forbids fragmentation a corresponding ICMP packet SHOULD be sent.

6.7. Multi-Point Traversal Considerations

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 NATs. The address to add is the wildcard address and the lookup address SHOULD also contain the VTags parameter and optionally the Disable Restart parameter as illustrated above.
7. Various Examples of NAT Traversals

Please note that this section is informational only.

The addresses being used in the following examples are IPv4 addresses for private-use networks and for documentation as specified in [RFC6890]. However, the method described here is not limited to this NAT44 case.

7.1. Single-homed Client to Single-homed Server

The internal client starts the association with the external server via a four-way-handshake. Host A starts by sending an INIT chunk.

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

NAT     | Int    | Int      | Ext     | Ext     | Priv      |
| VTag   | Port   | VTag     | Port   | Addr     |
+---------+--------+----------+--------+-----------+
  | 1234   |    1   |     0   |    2   | 10.0.0.1 |
+---------+--------+----------+--------+-----------+
```

INIT[Initiate-Tag = 1234]
10.0.0.1:1 -----> 203.0.113.1:2
Ext-VTag = 0

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

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

NAT     | Int    | Int      | Ext     | Ext     | Priv      |
| VTag   | Port   | VTag     | Port   | Addr     |
+---------+--------+----------+--------+-----------+
  | 1234   |    1   |     0   |    2   | 10.0.0.1 |
+---------+--------+----------+--------+-----------+
```

INIT[Initiate-Tag = 1234]
192.0.2.1:1 ------------------> 203.0.113.1:2
Ext-VTag = 0
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:

+---------+--------+----------+--------+-----------+
| NAT     |  Int    |  Int    |   Ext   |   Ext     |    Priv   |
+---------+--------+----------+--------+-----------+
|  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.
7.2. 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:

NAT | Int VTag | Int Port | Ext VTag | Ext Port | Priv Addr |
--- | -------- | -------- | -------- | -------- | ---------|
    | 1234     | 1        | 0        | 2        | 10.0.0.1  |

INIT[Initiate-Tag = 1234]
192.0.2.1:1 --------------------------> 203.0.113.1:2
Ext-VTag = 0

The server (Host B) includes its two addresses in the INIT-ACK chunk, which results in two NAT entries.
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:

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>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 includes its second address in the INIT-ACK, which results in two NAT entries in NAT 1.
NAT 1 does not need to update the table for second address:

<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:
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.

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

The NAT box cannot find entry for the association. It sends ERROR message with the M-Bit set and the cause "NAT state missing".
ERROR [M-Bit, NAT state missing]
10.0.0.1:1 <---------- 203.0.113.1:2
Ext-VTag = 5678

On reception of the ERROR message, Host A sends an ASCONF chunk indicating that the former information has to be deleted and the source address of the actual packet added.

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

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

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

Host B adds the new source address and deletes all former entries.
7.5. Peer-to-Peer Communication

If two hosts are behind NATs, 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 boxes 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:

NAT A
+----------+--------+----------+--------+-----------+
| Int VTag | Int Port| Ext VTag | Ext Port| Priv Addr |
+----------+--------+----------+--------+-----------+
| 1234     | 1      | 0        | 2      | 10.0.0.1  |
+----------+--------+----------+--------+-----------+

INIT[Initiate-Tag = 1234]
192.0.2.1:1 ----------------> 203.0.113.1:2
Ext-VTag = 0

NAT B processes INIT, but cannot find an entry. The SCTP packet is
silently discarded and leaves the NAT table of NAT B unchanged.
Now Host B sends INIT, which is processed by NAT B. Its parameters are used to create an entry.

<table>
<thead>
<tr>
<th>Internal</th>
<th>External</th>
<th>External</th>
<th>Internal</th>
</tr>
</thead>
</table>
| +--------+        | /--\---\ 
| Host A  | <-----| NAT A  | <----| Internet | <----| NAT B  | <-----| Host B |
| +--------+        | \--/---/ 

INIT[Initiate-Tag = 5678]
192.0.2.1:1 <-- 10.1.0.1:2
Ext-VTag = 0

<table>
<thead>
<tr>
<th>NAT B</th>
<th>Int VTag</th>
<th>Int Port</th>
<th>Priv Addr</th>
<th>Ext VTag</th>
<th>Ext Port</th>
</tr>
</thead>
</table>
| +-----+---------+---------+-----------+---------+---------+
| 5678 | 2       | 10.1.0.1 | 0         | 1       |

INIT[Initiate-Tag = 5678]
192.0.2.1:1 <---------- 203.0.113.1:2
Ext-VTag = 0

NAT A processes INIT. As the outgoing INIT of Host A has already created an entry, the entry is found and updated:
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:

<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>5678</td>
<td>2</td>
<td>1234</td>
<td>1</td>
<td>10.1.0.1</td>
</tr>
</tbody>
</table>

INIT-ACK[Initiate-Tag = 1234]
192.0.2.1:1 --> 10.1.0.1:2
Ext-VTag = 5678

The lookup for COOKIE-ECHO and COOKIE-ACK is successful.
8. Socket API Considerations

This section describes how the socket API defined in [RFC6458] is extended to provide a way for the application to control NAT friendliness.

Please note that this section is informational only.

A socket API implementation based on [RFC6458] is extended by supporting one new read/write socket option.
8.1. Get or Set the NAT Friendliness (SCTP_NAT_FRIENDLY)

This socket option uses the option_level IPPROTO_SCTP and the
option_name SCTP_NAT_FRIENDLY. It can be used to enable/disable the
NAT friendliness for future associations and retrieve the value for
future and specific ones.

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 suggested 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 suggested 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 suggested 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 suggested 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 suggested 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 suggested 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 end points, 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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,

[RFC4895] Tuexen, M., Stewart, R., Lei, P., and E. Rescorla,
"Authenticated Chunks for the Stream Control Transmission
Protocol (SCTP)", RFC 4895, DOI 10.17487/RFC4895, August

RFC 4960, DOI 10.17487/RFC4960, September 2007,

[RFC5061] Stewart, R., Xie, Q., Tuexen, M., Maruyama, S., and M.
Kozuka, "Stream Control Transmission Protocol (SCTP)
Dynamic Address Reconfiguration", RFC 5061,
DOI 10.17487/RFC5061, September 2007,

Protocol (SCTP) Chunk Flags Registration", RFC 6096,
DOI 10.17487/RFC6096, January 2011,
<https://www.rfc-editor.org/info/rfc6096>.

12.2. Informative References

address translation for SCTP", ACM SIGCOMM Computer
Communication Review Vol. 39, pp. 23,

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7,
RFC 793, DOI 10.17487/RFC0793, September 1981,
<https://www.rfc-editor.org/info/rfc793>.


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 RFC 4960 in September 2007 based on experience with implementing, testing, and using SCTP along with the suggested fixes. This document provides deltas to RFC 4960 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
Old text: (Section 9.2)

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

New text: (Section 9.2)

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

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

3.2.3. Solution Description

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

3.3. Registration of New Chunk Types

3.3.1. Description of the Problem

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

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

3.3.2. Text Changes to the Document
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>Optional 9 Reserved for ECN Capable (Note 2)</td>
<td>Optional</td>
<td>32768 (0x8000)</td>
<td></td>
</tr>
<tr>
<td>Host Name Address (Note 3)</td>
<td>Optional</td>
<td>11</td>
<td>Supported Address Types (Note 4)</td>
</tr>
<tr>
<td>Optional 12</td>
<td>Optional</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

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

### 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></td>
</tr>
<tr>
<td>IPv6 Address (Note 1)</td>
<td>Optional</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Cookie Preservative</td>
<td>Optional</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Reserved for ECN Capable (Note 2)</td>
<td>Optional</td>
<td>32768 (0x8000)</td>
<td></td>
</tr>
<tr>
<td>Host Name Address (Note 3)</td>
<td>Optional</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>Supported Address Types (Note 4)</td>
<td>Optional</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

3.4.3. Solution Description

Fix the formatting of the table.

3.5. CRC32c Sample Code on 64-bit Platforms

3.5.1. Description of the Problem

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

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

3.5.2. Text Changes to the Document
Old text: (Appendix C)

unsigned long
generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    unsigned long crc32 = ~0L;

New text: (Appendix C)

unsigned long
generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    unsigned long crc32 = 0xffffffffL;

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

3.5.3. Solution Description

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

3.6. Endpoint Failure Detection

3.6.1. Description of the Problem

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

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

3.6.2. Text Changes to the Document
Old text: (Section 8.1)

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

New text: (Section 8.1)

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

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

3.6.3. Solution Description

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

3.7. Data Transmission Rules

3.7.1. Description of the Problem

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

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

3.7.2. Text Changes to the Document
Old text: (Section 6.1 A))

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

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

New text: (Section 6.1 A))

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

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

3.7.3. Solution Description

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

3.8. T1-Cookie Timer

3.8.1. Description of the Problem

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

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

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

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

/---- COOKIE-ACK

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

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

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

/---- COOKIE-ACK

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

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

3.8.3. Solution Description

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

3.9. Miscellaneous Typos

3.9.1. Description of the Problem

While processing [RFC4960] some typos were not caught.

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

Old text: (Section 1.6)

Transmission Sequence Numbers wrap around when they reach 2**32 − 1. That is, the next TSN a DATA chunk MUST use after transmitting TSN = 2*32 − 1 is TSN = 0.

New text: (Section 1.6)

Transmission Sequence Numbers wrap around when they reach 2**32 − 1. That is, the next TSN a DATA chunk MUST use after transmitting TSN = 2**32 − 1 is TSN = 0.

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

Old text: (Section 3.3.10.9)

No User Data: This error cause is returned to the originator of a DATA chunk if a received DATA chunk has no user data.

New text: (Section 3.3.10.9)

No User Data: This error cause is returned to the originator of a DATA chunk if a received DATA chunk has no user data.

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

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

New text: (Section 6.7, Figure 9)

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

This text is in final form, and is not further updated in this document.
Old text: (Section 6.10)

An endpoint bundles chunks by simply including multiple chunks in one outbound SCTP packet. The total size of the resultant IP datagram, including the SCTP packet and IP headers, MUST be less that or equal to the current Path MTU.

New text: (Section 6.10)

An endpoint bundles chunks by simply including multiple chunks in one outbound SCTP packet. The total size of the resultant IP datagram, including the SCTP packet and IP headers, MUST be less than or equal to the current PMTU.

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

Old text: (Section 10.1 O))

o Receive Unacknowledged Message

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

New text: (Section 10.1 O))

O) Receive Unacknowledged Message

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

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

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 PMTU discovery.

----

New text: (Appendix C)

ICMP7) If the ICMP message is either a v6 "Packet Too Big" or a v4 "Fragmentation Needed", an implementation MAY process this information as defined for PMTU discovery.

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

----

Old text: (Section 5.4)

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

New text: (Section 5.4)

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

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

---

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

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

Old text: (Section 5.2.5)

---

5.2.5. Handle Duplicate COOKIE-ACK.

---

New text: (Section 5.2.5)

---

5.2.5. Handle Duplicate COOKIE ACK.

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

By default, an SCTP endpoint SHOULD monitor the reachability of the idle destination transport address(es) of its peer by sending a HEARTBEAT chunk periodically to the destination transport address(es). HEARTBEAT sending MAY begin upon reaching the ESTABLISHED state and is discontinued after sending either SHUTDOWN or SHUTDOWN-ACK. A receiver of a HEARTBEAT MUST respond to a HEARTBEAT with a HEARTBEAT-ACK after entering the COOKIE-ECHOED state (INIT sender) or the ESTABLISHED state (INIT receiver), up until reaching the SHUTDOWN-SENT state (SHUTDOWN sender) or the SHUTDOWN-ACK-SENT state (SHUTDOWN receiver).

New text: (Section 8.3)

By default, an SCTP endpoint SHOULD monitor the reachability of the idle destination transport address(es) of its peer by sending a HEARTBEAT chunk periodically to the destination transport address(es). HEARTBEAT sending MAY begin upon reaching the ESTABLISHED state and is discontinued after sending either SHUTDOWN or SHUTDOWN ACK. A receiver of a HEARTBEAT MUST respond to a HEARTBEAT with a HEARTBEAT ACK after entering the COOKIE-ECHOED state (INIT sender) or the ESTABLISHED state (INIT receiver), up until reaching the SHUTDOWN-SENT state (SHUTDOWN sender) or the SHUTDOWN-ACK-SENT state (SHUTDOWN receiver).

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

3.9.3. Solution Description

Typos fixed.

3.10. CRC32c Sample Code

3.10.1. Description of the Problem

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

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

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

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

3.10.3. Solution Description

Text moved to the appropriate location.

3.11. partial_bytes_acked after T3-rtx Expiration

3.11.1. Description of the Problem

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

3.11.2. Text Changes to the Document

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

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

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

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

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

\[
\text{ssthresh} = \text{max}(\text{cwnd}/2, 4\times\text{MTU})
\]
\[
\text{cwnd} = 1\times\text{MTU}
\]
\[
\text{partial_bytes_acked} = 0
\]
3.11.3. Solution Description

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

3.12. Order of Adjustments of partial_bytes_acked and cwnd

3.12.1. Description of the Problem

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

3.12.2. Text Changes to the Document

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

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

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

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

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

3.12.3. Solution Description

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

3.13.1. Description of the Problem

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

3.13.2. Text Changes to the Document

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

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

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

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

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

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

3.14.3. Solution Description

The new text clarifies where to send fast retransmissions.

3.15. Transmittal in Fast Recovery

3.15.1. Description of the Problem

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

3.15.2. Text Changes to the Document
Old text: (Section 7.2.4)

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

New text: (Section 7.2.4)

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

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

3.15.3. Solution Description

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

3.16. Initial Value of ssthresh

3.16.1. Description of the Problem

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

Old text: (Section 7.2.1)

- 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
3.17.3. Solution Description

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

3.18. Only One Packet after Retransmission Timeout

3.18.1. Description of the Problem

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

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

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

New text: (Section 7.2.1)

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

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

3.18.3. Solution Description

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

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

3.19.1. Description of the Problem

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

3.19.2. Text Changes to the Document
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.

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))

--------

o Stream Sequence Number - this value is returned indicating the Stream Sequence Number that was associated with the message.

o partial flag - if this returned flag is set to 1, then this message is a partial delivery of the whole message. When this flag is set, the stream id and Stream Sequence Number MUST accompany this receive. When this flag is set to 0, it indicates that no more deliveries will be received for this Stream Sequence Number.

--------

New text: (Section 10.1 N))

--------

o stream sequence number - this value is returned indicating the Stream Sequence Number that was associated with the message.

o partial flag - if this returned flag is set to 1, then this message is a partial delivery of the whole message. When this flag is set, the stream id and stream sequence number must accompany this primitive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.

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

---

- 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_acked although they confirm that the DATA chunks were successfully received by the peer.

3.22.2. Text Changes to the Document

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

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

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

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

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

3.22.3. Solution Description

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

3.23. Inconsistency in Notifications Handling

3.23.1. Description of the Problem

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

3.23.2. Text Changes to the Document

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

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

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

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

This text has been modified by multiple errata. It includes modifications from Section 3.6. It is in final form, and is not further updated in this document.
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).

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).

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 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.

- 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 \((\text{partial_bytes}_\text{acked} - \text{cwnd})\).

New text: (Section 7.2.2)

- 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.

- 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 \((\text{partial_bytes}_\text{acked} - \text{cwnd})\). Next, cwnd is increased by \(1^*\text{MTU}\).

This text has been modified by multiple errata. It includes modifications from Section 3.12 and Section 3.22. It is in final form, and is not further updated in this document.

3.26.3. Solution Description

The basic guidelines for incrementing cwnd during the congestion avoidance phase are added into Section 7.2.2. The guidelines include the normative language and are aligned with [RFC5681].

The algorithm from Section 7.2.2 is improved to not allow increasing cwnd by more than 1*MTU per RTT.

3.27. Refresh of cwnd and ssthresh after Idle Period

3.27.1. Description of the Problem

[RFC4960] prescribes to adjust cwnd per RTO if the endpoint does not transmit data on a given transport address. In addition to that, it prescribes to set cwnd to the initial value after a sufficiently long idle period. The latter is excessive. Moreover, it is unclear what is a sufficiently long idle period.

[RFC4960] doesn’t specify the handling of ssthresh in the idle case. If ssthresh is reduced due to a packet loss, ssthresh is never recovered. So traffic can end up in Congestion Avoidance all the time, resulting in a low sending rate and bad performance. The problem is even more serious for SCTP because in a multi-homed SCTP association traffic that switches back to the previously failed primary path will also lead to the situation where traffic ends up in Congestion Avoidance.

3.27.2. Text Changes to the Document

--------
Old text: (Section 7.2.1)
--------

o The initial cwnd before DATA transmission or after a sufficiently long idle period MUST be set to min(4*MTU, max (2*MTU, 4380 bytes)).

--------
New text: (Section 7.2.1)
--------

o The initial cwnd before DATA transmission MUST be set to min(4*MTU, max (2*MTU, 4380 bytes)).
Old text: (Section 7.2.1)

- When the endpoint does not transmit data on a given transport address, the cwnd of the transport address should be adjusted to \( \text{max}(\text{cwnd}/2, 4\times \text{MTU}) \) 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 \( \text{max}(\text{cwnd}/2, 4\times \text{MTU}) \) once per RTO. Before the first cwnd adjustment, the ssthresh of the transport address SHOULD be set to the cwnd.

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

3.27.3. Solution Description

A rule about cwnd adjustment after a sufficiently long idle period is removed.

The text is updated to describe the ssthresh handling. When the idle period is detected, the cwnd value is stored to the ssthresh value.

3.28. Window Updates After Receiver Window Opens Up

3.28.1. Description of the Problem

The sending of SACK chunks for window updates is only indirectly referenced in [RFC4960], Section 6.2, where it is stated that an SCTP receiver must not generate more than one SACK for every incoming packet, other than to update the offered window.

However, the sending of window updates when the receiver window opens up is necessary to avoid performance problems.

3.28.2. Text Changes to the Document
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 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.
Old text: (Section 6.1)
---------
C) When the time comes for the sender to transmit, before sending new DATA chunks, the sender MUST first transmit any outstanding DATA chunks that are marked for retransmission (limited by the current cwnd).

New text: (Section 6.1)
---------
C) When the time comes for the sender to transmit, before sending new DATA chunks, the sender MUST first transmit any DATA chunks that are marked for retransmission (limited by the current cwnd).

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

3.30.3. Solution Description

Now Section 1.3, Key Terms, includes explanations of outstanding data, data in flight and flightsize key terms. Section 6.1 is corrected to properly use the outstanding data term.

3.31. CWND Degradation due to Max.Burst

3.31.1. Description of the Problem

Some implementations were experiencing a degradation of cwnd because of the Max.Burst limit. This was due to misinterpretation of the suggestion in [RFC4960], Section 6.1, on how to use the Max.Burst parameter when calculating the number of packets to transmit.

3.31.2. Text Changes to the Document
D) When the time comes for the sender to transmit new DATA chunks, the protocol parameter Max.Burst SHOULD be used to limit the number of packets sent. The limit MAY be applied by adjusting cwnd as follows:

if((flightsize + Max.Burst*MTU) < cwnd) cwnd = flightsize + Max.Burst*MTU

Or it MAY be applied by strictly limiting the number of packets emitted by the output routine.

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.
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.

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.

Stewart, et al. Expires April 25, 2019 [Page 64]
Old text: (Appendix A)

[RFC3168] details negotiation of ECN during the SYN and SYN-ACK stages of a TCP connection.

New text: (Appendix A)


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

Old text: (Appendix A)

[RFC3168] details a specific bit for a receiver to send back in its TCP acknowledgements to notify the sender of the Congestion Experienced (CE) bit having arrived from the network.

New text: (Appendix A)

[RFC3168] updated by [RFC8311] details a specific bit for a receiver to send back in its TCP acknowledgements to notify the sender of the Congestion Experienced (CE) bit having arrived from the network.

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

[RFC3168] details a specific bit for a sender to send in the header of its next outbound TCP segment to indicate to its peer that it has reduced its congestion window.

New text: (Appendix A)

[RFC3168] updated by [RFC8311] details a specific bit for a sender to send in the header of its next outbound TCP segment to indicate to its peer that it has reduced its congestion window.

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

3.40.3. Solution Description

References to [RFC8311] have been added. While there, some wordsmithing has been performed.

3.41. Host Name Address Parameter Deprecated

3.41.1. Description of the Problem

[RFC4960] defines three types of address parameters to be used with INIT and INIT ACK chunks:

1. IPv4 Address parameters.
2. IPv6 Address parameters.
3. Host Name Address parameters.

The first two are supported by the SCTP kernel implementations of FreeBSD, Linux and Solaris, but the third one is not. In addition, the first two where 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.

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.

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
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.

3.41.3. Solution Description

The usage of the Host Name Address parameter has been deprecated.

3.42. Conflicting Text Regarding the Supported Address Types Parameter

3.42.1. Description of the Problem

When receiving an SCTP packet containing an INIT chunk sent from an address for which the corresponding address type is not listed in the Supported Address Types, there is conflicting text in Section 5.1.2 of [RFC4960]. It is stated that the association MUST be aborted and also that the association SHOULD be established and there SHOULD NOT be any error indication.
3.42.2. Text Changes to the Document

---------
Old text: (Section 5.1.2)
---------

The sender of INIT may include a ’Supported Address Types’ parameter in the INIT to indicate what types of address are acceptable. When this parameter is present, the receiver of INIT (initiate) MUST either use one of the address types indicated in the Supported Address Types parameter when responding to the INIT, or abort the association with an "Unresolvable Address" error cause if it is unwilling or incapable of using any of the address types indicated by its peer.

---------
New text: (Section 5.1.2)
---------

The sender of INIT chunks MAY include a ’Supported Address Types’ parameter in the INIT to indicate what types of addresses are acceptable.

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

3.42.3. Solution Description

The conflicting text has been removed.

3.43. Integration of RFC 6096

3.43.1. Description of the Problem

[RFC6096] updates [RFC4960] by adding a Chunk Flags Registry. This should be integrated into the base specification.

3.43.2. Text Changes to the Document

---------
Old text: (Section 14.1)
---------

14.1. IETF-Defined Chunk Extension

The assignment of new chunk parameter type codes is done through an IETF Consensus action, as defined in [RFC2434]. Documentation of the chunk parameter MUST contain the following information:
a) A long and short name for the new chunk type.

b) A detailed description of the structure of the chunk, which MUST conform to the basic structure defined in Section 3.2.

c) A detailed definition and description of the intended use of each field within the chunk, including the chunk flags if any.

d) A detailed procedural description of the use of the new chunk type within the operation of the protocol.

The last chunk type (255) is reserved for future extension if necessary.

--------
New text: (Section 14.1)
--------

14.1. IETF-Defined Chunk Extension

The assignment of new chunk type codes is done through an IETF Review action, as defined in [RFC8126]. Documentation of a new chunk MUST contain the following information:

a) A long and short name for the new chunk type;

b) A detailed description of the structure of the chunk, which MUST conform to the basic structure defined in Section 3.2 of [RFC4960];

c) A detailed definition and description of the intended use of each field within the chunk, including the chunk flags if any. Defined chunk flags will be used as initial entries in the chunk flags table for the new chunk type;

d) A detailed procedural description of the use of the new chunk type within the operation of the protocol.

The last chunk type (255) is reserved for future extension if necessary.

For each new chunk type, IANA creates a registration table for the chunk flags of that type. The procedure for registering particular chunk flags is described in the following Section 14.2.

This text has been modified by multiple errata. It includes modifications from Section 3.3. It is in final form, and is not further updated in this document.
New text: (Section 14.2)

14.2. New IETF Chunk Flags Registration

The assignment of new chunk flags is done through an RFC required action, as defined in [RFC8126]. Documentation of the chunk flags MUST contain the following information:

a) A name for the new chunk flag;

b) A detailed procedural description of the use of the new chunk flag within the operation of the protocol. It MUST be considered that implementations not supporting the flag will send ‘0’ on transmit and just ignore it on receipt.

IANA selects a chunk flags value. This MUST be one of 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, or 0x80, which MUST be unique within the chunk flag values for the specific chunk type.

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

Please note that Sections 14.2, 14.3, 14.4, and 14.5 need to be renumbered.

3.43.3. Solution Description

[RFC6096] was integrated and the reference updated to [RFC8126].

3.44. Integration of RFC 6335

3.44.1. Description of the Problem

[RFC6335] updates [RFC4960] by updating Procedures for the Port Numbers Registry. This should be integrated into the base specification. While there, update the reference to the RFC giving guidelines for writing IANA sections to [RFC8126].

3.44.2. Text Changes to the Document

Old text: (Section 14.5)

SCUP 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 |
| 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
New text: (Append to Section 6.1)

Whenever the sender of a DATA chunk can benefit from the corresponding SACK chunk being sent back without delay, the sender MAY set the I bit in the DATA chunk header. Please note that why the sender has set the I bit is irrelevant to the receiver.

Reasons for setting the I bit include, but are not limited to (see Section 4 of [RFC7053] for the benefits):

- 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.

New text: (Section 6.2)

Note: The SHUTDOWN chunk does not contain Gap Ack Block fields. Therefore, the endpoint SHOULD use a SACK instead of the SHUTDOWN chunk to acknowledge DATA chunks received out of order.

New text: (Section 6.2)

Note: The SHUTDOWN chunk does not contain Gap Ack Block fields. Therefore, the endpoint SHOULD use a SACK instead of the SHUTDOWN chunk to acknowledge DATA chunks received out of order.

Upon receipt of an SCTP packet containing a DATA chunk with the I bit set, the receiver SHOULD NOT delay the sending of the corresponding SACK chunk, i.e., the receiver SHOULD immediately respond with the corresponding SACK chunk.
Please note that this change is only about adding a paragraph.

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

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

E) Send

Format: SEND(association id, buffer address, byte count [,context]
        [,stream id] [,life time] [,destination transport address]
-> result

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

E) Send

Format: SEND(association id, buffer address, byte count [,context]
        [,stream id] [,life time] [,destination transport address]
        [,unordered flag] [,no-bundle flag] [,payload protocol-id]
        [,sack immediately] )
-> result

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

--------
New text: (Append optional parameter in Subsection E of Section 10.1)
--------

o sack immediately - set the I bit on the last DATA chunk used for sending buffer.

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

3.45.3. Solution Description

[RFC7053] was integrated.
3.46. CRC32c Code Improvements

3.46.1. Description of the Problem

The code given for the CRC32c computations uses types like long which may have different length on different operating systems or processors. Therefore, the code is changed to use specific types like uint32_t.

While there, fix also some syntax errors and a comment.

3.46.2. Text Changes to the Document

---------
Old text: (Appendix C)
---------
/*****************************/
/* Note Definition for Ross Williams table generator would */
/* be: TB_WIDTH=4, TB_POLLY=0x1EDC6F41, TB_REVER=TRUE */
/* For Mr. Williams direct calculation code use the settings */
/* cm_width=32, cm_poly=0x1EDC6F41, cm_init=0xFFFFFFFF, */
/* cm_refin=TRUE, cm_refot=TRUE, cm_xorort=0x00000000 */
/*****************************/
/
/* Example of the crc table file */
 ifndef __crc32cr_table_h__
 #define __crc32cr_table_h__

 #define CRC32C_POLY 0x1EDC6F41
 #define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])

 unsigned long  crc_c[256] =
{
  0x00000000L, 0xF26B8303L, 0xE13B70F7L, 0x1350F3F4L,
  0xC79A971FL, 0x35F1141CL, 0x26A1E7E8L, 0xD4CA64EBL,
  0x8AD958CFL, 0x78B2DBCCL, 0x6BE22838L, 0x9989AB3BL,
  0x4D43CFD0L, 0xBF284CD3L, 0xAC78BF27L, 0x5E133C24L,
  0x105EC76FL, 0xE235446CL, 0xF165B798L, 0x030E349BL,
  0xD7C45070L, 0x25AFD373L, 0x36FF2087L, 0xC494A384L,
  0x9A879FA0L, 0x68EC1CA3L, 0x7BBCEF57L, 0x89D76C54L,
  0x5D1D08FBL, 0xAF768BBCL, 0xBC267848L, 0x4E4DBF4BL,
  0x20BD8EDEL, 0xD2D60DDD1, 0xC186FE29L, 0x33ED7D2AL,
  0xE72719C1L, 0x154C9AC2L, 0x061C6936L, 0xF477EA35L,
  0xAA64D611L, 0x580F5512L, 0x4B5FA6E6L, 0xB93425E5L,
  0x6DFE410EL, 0x9F95C20DL, 0x8CC531F9L, 0x7EAB2FAL,
  0x30E349B1L, 0xC288CAB2L, 0xD1DB83946L, 0x23B3BA45L,
# endif

---------

New text: (Appendix B)
---------

<CODE BEGINS>

/*************************************************************/
/* Note Definition for Ross Williams table generator would */
/* be: TB_WIDTH=4, TB_POLLY=0x1EDC6F41, TB_REVER=TRUE */
/* For Mr. Williams direct calculation code use the settings */
/* cm_width=32, cm_poly=0x1EDC6F41, cm_init=0xFFFFFFFF */
/* cm_refin=TRUE, cm_refot=TRUE, cm_xorort=0x00000000 */
/*************************************************************/

/* Example of the crc table file */
#endif __crc32cr_h__
#define __crc32cr_h__
#define CRC32C_POLY 0x1EDC6F41UL
#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])

uint32_t crc_c[256] =
{
    0x00000000UL, 0xF26B8303UL, 0xE13B70F7UL, 0x1350F3F4UL,
    0xC79A971FUL, 0x35F1141CUL, 0x26A1E7E8UL, 0xD4CA64EBUL,
    0x8AD958CFUL, 0x78B2DBCCUL, 0x6BE22838UL, 0x9989AB3BUL,
    0x4D43CFD0UL, 0xBF284CD3UL, 0xAC78BF27UL, 0x5E133C24UL,
    0x105EC76FUL, 0xE235446CUL, 0xF165B798UL, 0x030E349BUL,
    0xD7C45070UL, 0x25AFD373UL, 0x36FF2087UL, 0xC494A384UL,
    0x9A879FA0UL, 0x68E1CA30UL, 0x89D76C54UL, 0xEF000000UL,
    0x5D1D08BFUL, 0xAF768BBCUL, 0xBC267848UL, 0x04E4DF4BUL,
    0x20BBDEDEUL, 0xBD6D0DDDUL, 0xC186FE29UL, 0x33ED7D2AUL,
    0xE72719C1UL, 0x5149C92CUL, 0x061C6936UL, 0xF477EA35UL,
    0xAA64D611UL, 0x580F5512UL, 0x4B5FA6E6UL, 0xB93425E5UL,
    0x7D9E410EUL, 0x9F95C2D9UL, 0x7CC351F9UL, 0x7EAEB2FAUL,
    0x30E349B1UL, 0xC288CB2UL, 0xD1D83946UL, 0x23B3BA45UL,
    0xF779DEAEUL, 0x05125DADUL, 0x1642AE95UL, 0xE4292D5AUL,
    0xB5A3A17EUL, 0x4851927DUL, 0x5B016189UL, 0xA96AE28AUL,
    0x7DA08661UL, 0x8FCEB056UL, 0x9C9BF696UL, 0x5EF07595UL,
    0x417B1D8CUL, 0xB3109EBFUL, 0xA0406D48UL, 0x522BE48UL,
    0x86B18A33UL, 0x748A094AUL, 0x67DAFA54UL, 0x95B17957UL,
    0xCBA24573UL, 0x39C9670UL, 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,
0xE3DB33C7UL, 0x2D80B0C4UL, 0x3ED04330UL, 0xC38D26C4UL,
0x0417B1DBUL, 0xF67C32D8UL, 0xE52CC12CUL, 0x1747422FUL,
0x28926E69UL, 0x0CAF9EE6AUL, 0xC9A99D9EUL, 0x38CD9F22UL,
0x6D522B88UL, 0x97BA1BAUL, 0x84E5A24EUL, 0x7681D14DUL,
0x2B1572C9UL, 0x144976B4UL, 0x532E023EUL, 0x1A45813DUL,
0x1A744D88UL, 0x0EA1A2DBUL, 0xF5944D2FUL, 0x0B21572CUL,
0x0B644D5AUL, 0x3ED0D4330UL, 0xCCB0C033UL, 0x1E6DCDEEUL,
0x144976B4UL, 0x8E3DB33C7UL, 0x0417B1DBUL, 0xF67C32D8UL,
0x28926E69UL, 0x0CAF9EE6AUL, 0xC9A99D9EUL, 0x38CD9F22UL,
0x6D522B88UL, 0x97BA1BAUL, 0x84E5A24EUL, 0x7681D14DUL,
0x2B1572C9UL, 0x144976B4UL, 0x532E023EUL, 0x1A45813DUL,
0x1A744D88UL, 0x0EA1A2DBUL, 0xF5944D2FUL, 0x0B21572CUL,
0x0B644D5AUL, 0x3ED0D4330UL, 0xCCB0C033UL, 0x1E6DCDEEUL,
0x144976B4UL, 0x8E3DB33C7UL, 0x0417B1DBUL, 0xF67C32D8UL,
0x28926E69UL, 0x0CAF9EE6AUL, 0xC9A99D9EUL, 0x38CD9F22UL,
0x6D522B88UL, 0x97BA1BAUL, 0x84E5A24EUL, 0x7681D14DUL,
0x2B1572C9UL, 0x144976B4UL, 0x532E023EUL, 0x1A45813DUL,
0x1A744D88UL, 0x0EA1A2DBUL, 0xF5944D2FUL, 0x0B21572CUL,
0x0B644D5AUL, 0x3ED0D4330UL, 0xCCB0C033UL, 0x1E6DCDEEUL,
0x144976B4UL, 0x8E3DB33C7UL, 0x0417B1DBUL, 0xF67C32D8UL,
0x28926E69UL, 0x0CAF9EE6AUL, 0xC9A99D9EUL, 0x38CD9F22UL,
0x6D522B88UL, 0x97BA1BAUL, 0x84E5A24EUL, 0x7681D14DUL,
0x2B1572C9UL, 0x144976B4UL, 0x532E023EUL, 0x1A45813DUL,
0x1A744D88UL, 0x0EA1A2DBUL, 0xF5944D2FUL, 0x0B21572CUL,
0x0B644D5AUL, 0x3ED0D4330UL, 0xCCB0C033UL, 0x1E6DCDEEUL,
0x144976B4UL, 0x8E3DB33C7UL, 0x0417B1DBUL, 0xF67C32D8UL,
0x28926E69UL, 0x0CAF9EE6AUL, 0xC9A99D9EUL, 0x38CD9F22UL,
0x6D522B88UL, 0x97BA1BAUL, 0x84E5A24EUL, 0x7681D14DUL,
0x2B1572C9UL, 0x144976B4UL, 0x532E023EUL, 0x1A45813DUL,
0x1A744D88UL, 0x0EA1A2DBUL, 0xF5944D2FUL, 0x0B21572CUL,
0x0B644D5AUL, 0x3ED0D4330UL, 0xCCB0C033UL, 0x1E6DCDEEUL,
0x144976B4UL, 0x8E3DB33C7UL, 0x0417B1DBUL, 0xF67C32D8UL,
This text has been modified by multiple errata. It includes modifications from Section 3.10. It is in final form, and is not further updated in this document.

---------
Old text: (Appendix C)
---------

/* Example of table build routine */

#include <stdio.h>
#include <stdlib.h>

#define OUTPUT_FILE   "crc32cr.h"
#define CRC32C_POLY    0x1EDC6F41L
FILE *tf;
unsigned long
reflect_32 (unsigned long b)
{
    int i;
    unsigned long rw = 0L;

    for (i = 0; i < 32; i++){
        if (b & 1)
            rw |= 1 << (31 - i);
        b >>= 1;
    }

    return (rw);
}

unsigned long
build_crc_table (int index)
{
    int i;
    unsigned long rb;

    rb = reflect_32 (index);

    for (i = 0; i < 8; i++){
        if (rb & 0x80000000L)
            rb = (rb << 1) ^ CRC32C_POLY;
        else
            rb <<= 1;
    }

    return (reflect_32 (rb));
}
```c
main ()
{
    int i;

    printf ("Generating CRC-32c table file <\%s>\n", OUTPUT_FILE);
    if ((tf = fopen (OUTPUT_FILE, "w")) == NULL){
        printf ("Unable to open %s\n", OUTPUT_FILE);
        exit (1);
    }
    fprintf (tf, "#ifndef __crc32cr_table_h__\n");
    fprintf (tf, "#define __crc32cr_table_h__\n\n");
    fprintf (tf, "#define CRC32C_POLY 0x%08lX\n", CRC32C_POLY);
    fprintf (tf, "#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])\n");
    fprintf (tf, "\nunsigned long  crc_c[256] =\n{\n    for (i = 0; i < 256; i++){
        fprintf (tf, "0x%08lXL, \n", build_crc_table (i));
        if ((i & 3) == 3)
            fprintf (tf, "\n\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);
}

---------
New text: (Appendix B)
---------
/* Example of table build routine */
#include <stdio.h>
#include <stdlib.h>
#define OUTPUT_FILE   "crc32cr.h"
#define CRC32C_POLY    0x1EDC6F41UL

static FILE *tf;

static uint32_t
reflect_32(uint32_t b)
{
    return b; /* Example of table build routine */
```

```
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("Generating CRC-32c table file <%s>\n",
            OUTPUT_FILE);
    if ((tf = fopen(OUTPUT_FILE, "w")) == NULL) {
        printf("Unable to open %s\n", OUTPUT_FILE);
        exit (1);
    }

    fprintf(tf, "#ifndef __crc32cr_h__\n");    
    fprintf(tf, "#define __crc32cr_h__\n");    
    fprintf(tf, "#define CRC32C_POLY 0x%08XUL\n",    
              (uint32_t)CRC32C_POLY);    
    fprintf(tf, "#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])\n");    
    fprintf(tf, "#define crc_c[256] \n");    
    for (i = 0; i < 256; i++) {
        fprintf(tf, "0x%08XUL,"),    
                build_crc_table (i));    
}
if ((i & 3) == 3)
    fprintf(tf, "\n")
else
    fprintf(tf, " ");
fprintf(tf, "};\n\n#endif\n*/

if (fclose (tf) != 0)
    printf("Unable to close <%s>.", OUTPUT_FILE);
else
    printf("\nThe CRC-32c table has been written to <%s>\n", OUTPUT_FILE);
}

This text has been modified by multiple errata. It includes modifications from Section 3.10. It is in final form, and is not further updated in this document.

---------
Old text: (Appendix C)
---------

/* Example of crc insertion */

#include "crc32cr.h"

unsigned long
generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    unsigned long crc32 = ~0L;
    unsigned long result;
    unsigned char byte0,byte1,byte2,byte3;

    for (i = 0; i < length; i++){
        CRC32C(crc32, buffer[i]);
    }

    result = ~crc32;

    /* result now holds the negated polynomial remainder;
     * since the table and algorithm is "reflected" [williams95].
     * That is, result has the same value as if we mapped the message
     * to a polynomial, computed the host-bit-order polynomial
     * remainder, performed final negation, then did an end-for-end
     * bit-reversal. */

* Note that a 32-bit bit-reversal is identical to four inplace
* 8-bit reversals followed by an end-for-end byteswap.
* In other words, the bytes of each bit are in the right order,
* but the bytes have been byteswapped. So we now do an explicit
* byteswap. On a little-endian machine, this byteswap and
* the final ntohl cancel out and could be elided.
*/

type = result & 0xff;
byte1 = (result>>8) & 0xff;
byte2 = (result>>16) & 0xff;
byte3 = (result>>24) & 0xff;
crc32 = ((byte0 << 24) |
(byte1 << 16) |
(byte2 << 8) |
byte3);
return ( crc32 );

int insert_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    unsigned long crc32;
    message = (SCTP_message *) buffer;
    message->common_header.checksum = 0L;
    crc32 = generate_crc32c(buffer,length);
    /* and insert it into the message */
    message->common_header.checksum = htonl(crc32);
    return 1;
}

int validate_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    unsigned int i;
    unsigned long original_crc32;
    unsigned long crc32 = ~0L;

    /* save and zero checksum */
    message = (SCTP_message *) buffer;
    original_crc32 = ntohl(message->common_header.checksum);
    message->common_header.checksum = 0L;
    crc32 = generate_crc32c(buffer,length);
    return ((original_crc32 == crc32) ? 1 : -1);
}
New text: (Appendix B)

/* Example of crc insertion */

#include "crc32cr.h"

uint32_t
generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    uint32_t crc32 = 0xffffffffUL;
    uint32_t result;
    uint8_t byte0, byte1, byte2, byte3;
    for (i = 0; i < length; i++) {
        CRC32C(crc32, buffer[i]);
    }
    result = ~crc32;
    /* result now holds the negated polynomial remainder;
     * since the table and algorithm is "reflected" [williams95].
     * That is, result has the same value as if we mapped the message
     * to a polynomial, computed the host-bit-order polynomial
     * remainder, performed final negation, then did an end-for-end
     * bit-reversal.
     * Note that a 32-bit bit-reversal is identical to four inplace
     * 8-bit reversals followed by an end-for-end byteswap.
     * In other words, the bits of each byte are in the right order,
     * but the bytes have been byteswapped. So we now do an explicit
     * byteswap. On a little-endian machine, this byteswap and
     * the final ntohl cancel out and could be elided.
     */
    byte0 = result & 0xff;
    byte1 = (result>>8) & 0xff;
    byte2 = (result>>16) & 0xff;
    byte3 = (result>>24) & 0xff;
    crc32 = ((byte0 << 24) |
              (byte1 << 16) |
              (byte2 << 8) |
              byte3);
    return (crc32);
}

int
insert_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    uint32_t crc32;
    message = (SCTP_message *) buffer;
    message->common_header.checksum = 0UL;
    crc32 = generate_crc32c(buffer, length);
    /* and insert it into the message */
    message->common_header.checksum = htonl(crc32);
    return 1;
}

int validate_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    unsigned int i;
    uint32_t original_crc32;
    uint32_t crc32;

    /* save and zero checksum */
    message = (SCTP_message *)buffer;
    original_crc32 = ntohl(message->common_header.checksum);
    message->common_header.checksum = 0L;
    crc32 = generate_crc32c(buffer, length);
    return ((original_crc32 == crc32) ? 1 : -1);
}

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 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 Aroun ds

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
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].
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 Morriss, Karen E. E. Nielsen, Tom Petch, Kacheong Poon, Julien Pourtet, Irene Ruengeler, Michael Welzl, and Qiaobing Xie for their invaluable comments.
7. References

7.1. Normative References


7.2. Informative References


Authors' Addresses

Randall R. Stewart
Netflix, Inc.
Chapin, SC  29036
United States

Email: randall@lakerest.net

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

Table of Contents

1. Introduction .................................................. 2
2. Terminology ................................................... 3
3. Scope of RFC 6040 .............................................. 3
   3.1. Feasibility of ECN Propagation between Tunnel Headers 4
   3.2. Desirability of ECN Propagation between Tunnel Headers 5
4. Making a non-ECN Tunnel Ingress Safe by Configuration ...... 5
5. IP-in-IP Tunnels with Tightly Coupled Shim Headers .......... 7
   5.1. Specific Updates to Protocols under IETF Change Control 9
      5.1.1. L2TP (v2 and v3) ECN Extension ................. 9
      5.1.2. GRE ............................................ 12
      5.1.3. Teredo ........................................ 13
      5.1.4. AMT ........................................... 14
6. IANA Considerations ........................................... 16
7. Security Considerations ....................................... 16
8. Comments Solicited ........................................... 16
9. Acknowledgements ............................................. 16
10. References .................................................. 17
    10.1. Normative References ................................ 17
    10.2. Informative References .............................. 18
Author’s Address ................................................ 21

1. Introduction

RFC 6040 on "Tunnelling of Explicit Congestion Notification" [RFC6040] made the rules for propagation of Explicit Congestion Notification (ECN [RFC3168]) consistent for all forms of IP in IP tunnel.

A common pattern for many tunnelling protocols is to encapsulate an inner IP header (v4 or v6) with shim header(s) then an outer IP header (v4 or v6). Some of these shim headers are designed as generic encapsulations, so they do not necessarily directly encapsulate an inner IP header. Instead they can encapsulate headers such as link-layer (L2) protocols that in turn often encapsulate IP.
To clear up confusion, this specification clarifies that the scope of RFC 6040 includes any IP-in-IP tunnel, including those with shim header(s) and other encapsulations between the IP headers. Where necessary, it updates the specifications of the relevant encapsulation protocols with the specific text necessary to comply with RFC 6040.

This specification also updates RFC 6040 to state how operators ought to configure a legacy tunnel ingress to avoid unsafe system configurations.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119] when, and only when, they appear in all capitals, as shown here.

This specification uses the terminology defined in RFC 6040 [RFC6040].

3. Scope of RFC 6040

In section 1.1 of RFC 6040, its scope is defined as:

"...ECN field processing at encapsulation and decapsulation for any IP-in-IP tunnelling, whether IPsec or non-IPsec tunnels. It applies irrespective of whether IPv4 or IPv6 is used for either the inner or outer headers. ..."

This was intended to include cases where shim header(s) sit between the IP headers. Many tunnelling implementers have interpreted the scope of RFC 6040 as it was intended, but it is ambiguous. Therefore, this specification updates RFC 6040 by adding the following scoping text after the sentences quoted above:

"It applies in cases where an outer IP header encapsulates an inner IP header either directly or indirectly by encapsulating other headers that in turn encapsulate (or might encapsulate) an inner IP header."

There is another problem with the scope of RFC 6040. Like many IETF specifications, RFC 6040 is written as a specification that implementations can choose to claim compliance with. This means it does not cover two important cases:
1. those cases where it is infeasible for an implementation to access an inner IP header when adding or removing an outer IP header;

2. those implementations that choose not to propagate ECN between IP headers.

However, the ECN field is a non-optional part of the IP header (v4 and v6). So any implementation that creates an outer IP header has to give the ECN field some value. There is only one safe value a tunnel ingress can use if it does not know whether the egress supports propagation of the ECN field; it has to clear the ECN field in any outer IP header to 0b00.

However, an RFC has no jurisdiction over implementations that choose not to comply with it or cannot comply with it, including all those implementations that pre-dated the RFC. Therefore it would have been unreasonable to add such a requirement to RFC 6040. Nonetheless, to ensure safe propagation of the ECN field over tunnels, it is reasonable to add requirements on operators, to ensure they configure their tunnels safely (where possible). Before stating these configuration requirements in Section 4, the factors that determine whether propagating ECN is feasible or desirable will be briefly introduced.

3.1. Feasibility of ECN Propagation between Tunnel Headers

In many cases shim header(s) and an outer IP header are always added to (or removed from) an inner IP packet as part of the same procedure. We call this a tightly coupled shim header. Processing the shim and outer together is often necessary because the shim(s) are not sufficient for packet forwarding in their own right; not unless complemented by an outer header. In these cases it will often be feasible for an implementation to propagate the ECN field between the IP headers.

In some cases a tunnel adds an outer IP header and a tightly coupled shim header to an inner header that is not an IP header, but that in turn encapsulates an IP header (or might encapsulate an IP header). For instance an inner Ethernet (or other link layer) header might encapsulate an inner IP header as its payload. We call this a tightly coupled shim over an encapsulating header.

Digging to arbitrary depths to find an inner IP header within an encapsulation is strictly a layering violation so it cannot be a required behaviour. Nonetheless, some tunnel endpoints already look within a L2 header for an IP header, for instance to map the Diffserv codepoint between an encapsulated IP header and an outer IP header.
In such cases at least, it should be feasible to also (independently) propagate the ECN field between the same IP headers. Thus, access to the ECN field within an encapsulating header can be a useful and benign optimization. The guidelines in section 5 of [I-D.ietf-tsvwg-ecn-encap-guidelines] give the conditions for this layering violation to be benign.

3.2. Desirability of ECN Propagation between Tunnel Headers

Developers and network operators are encouraged to implement and deploy tunnel endpoints compliant with RFC 6040 (as updated by the present specification) in order to provide the benefits of wider ECN deployment [RFC8087]. Nonetheless, propagation of ECN between IP headers, whether separated by shim headers or not, has to be optional to implement and to use, because:

- Legacy implementations of tunnels without any ECN support already exist
- A network might be designed so that there is usually no bottleneck within the tunnel
- If the tunnel endpoints would have to search within an L2 header to find an encapsulated IP header, it might not be worth the potential performance hit

4. Making a non-ECN Tunnel Ingress Safe by Configuration

Even when no specific attempt has been made to implement propagation of the ECN field at a tunnel ingress, it ought to be possible for the operator to render a tunnel ingress safe by configuration. The main safety concern is to disable (clear to zero) the ECN capability in the outer IP header at the ingress if the egress of the tunnel does not implement ECN logic to propagate any ECN markings into the packet forwarded beyond the tunnel. Otherwise the non-ECN egress could discard any ECN marking introduced within the tunnel, which would break all the ECN-based control loops that regulate the traffic load over the tunnel.

Therefore this specification updates RFC 6040 by inserting the following text at the end of section 4.3:

```
Whether or not an ingress implementation claims compliance with RFC 6040, RFC 4301 or RFC3168, when the outer tunnel header is IP (v4 or v6), if possible, the operator MUST configure the ingress to zero the outer ECN field in any of the following cases:
```
* if it is known that the tunnel egress does not support 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 make propagation of the ECN field between inner and outer IP headers independent of any configuration of Diffserv codepoint propagation;

* it SHOULD be able to be configured to zero the outer ECN field.

There might be concern that the above "MUST" makes compliant equipment non-compliant at a stroke. However, any equipment that is still treating the former ToS octet (IPv4) or the former Traffic Class octet (IPv6) as a single 8-bit field is already non-compliant, and has been since 1998 when the upper 6 bits were separated off for the Diffserv field [RFC2474], [RFC3260]. For instance, 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.

Permanently zeroing the outer ECN field is safe, but it is not sufficient to claim compliance with RFC 6040 because it does not meet the aim of introducing ECN support to tunnels (see Section 4.3 of [RFC6040]).
5. 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 5.1.2 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 the UDP usage guidelines [RFC8085] already explains that a tunnel that encapsulates an IP header directly 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 so, by reference, they automatically update 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 legacy 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.

5.1. Specific Updates to Protocols under IETF Change Control

5.1.1. 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 5.1.1.2 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.

5.1.1.1. 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 5.1.1.2 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].

5.1.1.2. 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 5.1.1.2.1 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.

5.1.1.2.1. LCCE Capability AVP for ECN Capability Negotiation

The LCCE Capability Attribute-Value Pair (AVP) defined here has Attribute Type ZZ. The Attribute Value field for this AVP is a bit-mask with the following 16-bit format:

```
0                   1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
|X X X X X X X X X X X X X X X E|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

Figure 1: Value Field for the LCCE Capability Attribute

This AVP MAY be present in the following message types: SCCRQ and SCCR (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 (SCCR) 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 SCCR.
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 5.1.1.1 above, when it acts as an ingress LCCE.

5.1.2. 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 [RFC5996] 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 5.1.2.1 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.

5.1.2.1. 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)

5.1.3. 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..."
5.1.4. 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 tunneled 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 5.1.4.2 below.

5.1.4.1. 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)
5.1.4.2.  ECN Capability Declaration of an AMT Gateway

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 5.1.4.1 above.

6. 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>

[TO BE REMOVED: This registration should take place at the following location: https://www.iana.org/assignments/l2tp-parameters/l2tp-parameters.xhtml]

7. 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.

8. 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.

9. 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.

10. References

10.1. Normative References

[I-D.ietf-tsvwg-ecn-encap-guidelines]
Briscoe, B., Kalppallimalil, J., and P. Thaler,
"Guidelines for Adding Congestion Notification to
Protocols that Encapsulate IP", draft-ietf-tsvwg-ecn-
encap-guidelines-11 (work in progress), November 2018.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,

[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,

[RFC2661] Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn,
G., and B. Palter, "Layer Two Tunneling Protocol "L2TP"",
RFC 2661, DOI 10.17487/RFC2661, August 1999,

Traina, "Generic Routing Encapsulation (GRE)", RFC 2784,
DOI 10.17487/RFC2784, March 2000,

of Explicit Congestion Notification (ECN) to IP",
RFC 3168, DOI 10.17487/RFC3168, September 2001,

"Layer Two Tunneling Protocol - Version 3 (L2TPv3)",
RFC 3931, DOI 10.17487/RFC3931, March 2005,

Internet Protocol", RFC 4301, DOI 10.17487/RFC4301,
10.2. Informative References


Briscoe  Expires September 30, 2019  [Page 20]

Author’s Address

Bob Briscoe
Independent
UK

EMail: ietf@bobbriscoe.net
URI: http://bobbriscoe.net/
Abstract

This document describes two fully-specified Forward Erasure Correction (FEC) Schemes for Sliding Window Random Linear Codes (RLC), one for RLC over the Galois Field (A.K.A. Finite Field) GF(2), a second one for RLC over the Galois Field GF(2^8), each time with the possibility of controlling the code density. They can protect arbitrary media streams along the lines defined by FECFRAME extended to sliding window FEC codes. These sliding window FEC codes rely on an encoding window that slides over the source symbols, generating new repair symbols whenever needed. Compared to block FEC codes, these sliding window FEC codes offer key advantages with real-time flows in terms of reduced FEC-related latency while often providing improved packet erasure recovery capabilities.

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.
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
1. Introduction

Application-Level Forward Erasure Correction (AL-FEC) codes, or simply FEC codes, are a key element of communication systems. They are used to recover from packet losses (or erasures) during content delivery sessions to a potentially large number of receivers (multicast/broadcast transmissions). This is the case with the FLUTE/ALC protocol [RFC6726] when used for reliable file transfers over lossy networks, and the FECFRAME protocol [RFC6363] when used for reliable continuous media transfers over lossy networks.

The present document only focuses on the FECFRAME protocol, used in multicast/broadcast delivery mode, in particular for contents that feature stringent real-time constraints: each source packet has a maximum validity period after which it will not be considered by the destination application.
1.1. Limits of Block Codes with Real-Time Flows

With FECFRAME, there is a single FEC encoding point (either an end-host/server (source) or a middlebox) and a single FEC decoding point per receiver (either an end-host (receiver) or middlebox). In this context, currently standardized AL-FEC codes for FECFRAME like Reed-Solomon [RFC6865], LDPC-Staircase [RFC6816], or Raptor/RaptorQ [RFC6681], are all linear block codes: they require the data flow to be segmented into blocks of a predefined maximum size.

To define this block size, it is required to find an appropriate balance between robustness and decoding latency: the larger the block size, the higher the robustness (e.g., in case of long packet erasure bursts), but also the higher the maximum decoding latency (i.e., the maximum time required to recover a lost (erased) packet thanks to FEC protection). Therefore, with a multicast/broadcast session where different receivers experience different packet loss rates, the block size should be chosen by considering the worst communication conditions one wants to support, but without exceeding the desired maximum decoding latency. This choice then impacts the FEC-related latency of all receivers, even those experiencing a good communication quality, since no FEC encoding can happen until all the source data of the block is available at the sender, which directly depends on the block size.

1.2. Lower Latency and Better Protection of Real-Time Flows with the Sliding Window RLC Codes

This document introduces two fully-specified FEC Schemes that do not follow the block code approach: the Sliding Window Random Linear Codes (RLC) over either Galois Fields (A.K.A. Finite Fields) GF(2) (the "binary case") or GF(2^8), each time with the possibility of controlling the code density. These FEC Schemes are used to protect arbitrary media streams along the lines defined by FECFRAME extended to sliding window FEC codes [fecframe-ext]. These FEC Schemes, and more generally Sliding Window FEC codes, are recommended for instance, with media that feature real-time constraints sent within a multicast/broadcast session [Roca17].

The RLC codes belong to the broad class of sliding-window AL-FEC codes (A.K.A. convolutional codes) [RFC8406]. The encoding process is based on an encoding window that slides over the set of source packets (in fact source symbols as we will see in Section 3.2), this window being either of fixed size or variable size (A.K.A. an elastic window). Repair symbols are generated on-the-fly, by computing a random linear combination of the source symbols present in the current encoding window, and passed to the transport layer.
At the receiver, a linear system is managed from the set of received source and repair packets. New variables (representing source symbols) and equations (representing the linear combination carried by each repair symbol received) are added upon receiving new packets. Variables and the equations they are involved in are removed when they are too old with respect to their validity period (real-time constraints). Lost source symbols are then recovered thanks to this linear system whenever its rank permits to solve it (at least partially).

The protection of any multicast/broadcast session needs to be dimensioned by considering the worst communication conditions one wants to support. This is also true with RLC (more generally any sliding window) code. However, the receivers experiencing a good to medium communication quality will observe a reduced FEC-related latency compared to block codes [Roca17] since an isolated lost source packet is quickly recovered with the following repair packet. On the opposite, with a block code, recovering an isolated lost source packet always requires waiting for the first repair packet to arrive after the end of the block. Additionally, under certain situations (e.g., with a limited FEC-related latency budget and with constant bitrate transmissions after FECFRAME encoding), sliding window codes can more efficiently achieve a target transmission quality (e.g., measured by the residual loss after FEC decoding) by sending fewer repair packets (i.e., higher code rate) than block codes.

1.3. Small Transmission Overheads with the Sliding Window RLC FEC Scheme

The Sliding Window RLC FEC Scheme is designed to limit the packet header overhead. The main requirement is that each repair packet header must enable a receiver to reconstruct the set of source symbols plus the associated coefficients used during the encoding process. In order to minimize packet overhead, the set of source symbols in the encoding window as well as the set of coefficients over GF(2^m) (where m is 1 or 8, depending on the FEC Scheme) used in the linear combination are not individually listed in the repair packet header. Instead, each FEC Repair Packet header contains:

- the Encoding Symbol Identifier (ESI) of the first source symbol in the encoding window as well as the number of symbols (since this number may vary with a variable size, elastic window). These two pieces of information enable each receiver to reconstruct the set of source symbols considered during encoding, the only constraint being that there cannot be any gap;
- the seed and density threshold parameters used by a coding coefficients generation function (Section 3.6). These two pieces
of information enable each receiver to generate the same set of coding coefficients over GF(2^m) as the sender;

Therefore, no matter the number of source symbols present in the encoding window, each FEC Repair Packet features a fixed 64-bit long header, called Repair FEC Payload ID (Figure 8). Similarly, each FEC Source Packet features a fixed 32-bit long trailer, called Explicit Source FEC Payload ID (Figure 6), that contains the ESI of the first source symbol (Section 3.2).

1.4. Document Organization

This fully-specified FEC Scheme follows the structure required by [RFC6363], section 5.6. "FEC Scheme Requirements", namely:

3. Procedures: This section describes procedures specific to this FEC Scheme, namely: RLC parameters derivation, ADUI and source symbols mapping, pseudo-random number generator, and coding coefficients generation function;

4. Formats and Codes: This section defines the Source FEC Payload ID and Repair FEC Payload ID formats, carrying the signaling information associated to each source or repair symbol. It also defines the FEC Framework Configuration Information (FFCI) carrying signaling information for the session;

5. FEC Code Specification: Finally this section provides the code specification.

2. Definitions and Abbreviations

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

This document uses the following definitions and abbreviations:

\[ a^{^b} \] a to the power of b

GF(q) denotes a finite field (also known as the Galois Field) with q elements. We assume that q = 2^m in this document.

m defines the length of the elements in the finite field, in bits. In this document, m is equal to 1 or 8.

ADU: Application Data Unit

ADUI: Application Data Unit Information (includes the F, L and padding fields in addition to the ADU)

E: size of an encoding symbol (i.e., source or repair symbol), assumed fixed (in bytes)
br_in: transmission bitrate at the input of the FECFRAME sender, assumed fixed (in bits/s)
br_out: transmission bitrate at the output of the FECFRAME sender, assumed fixed (in bits/s)
max_lat: maximum FEC-related latency within FECFRAME (a decimal number expressed in seconds)
cr: RLC coding rate, ratio between the total number of source symbols and the total number of source plus repair symbols
ew_size: encoding window current size at a sender (in symbols)
ew_max_size: encoding window maximum size at a sender (in symbols)
dw_max_size: decoding window maximum size at a receiver (in symbols)
ls_max_size: linear system maximum size (or width) at a receiver (in symbols)
WSR: window size ratio parameter used to derive ew_max_size (encoder) and ls_max_size (decoder).
PRNG: pseudo-random number generator
TinyMT32: PRNG used in this specification.
DT: coding coefficients density threshold, an integer between 0 and 15 (inclusive) the controls the fraction of coefficients that are non zero

3. Common Procedures

This section introduces the procedures that are used by these FEC Schemes.

3.1. Codec Parameters

A codec implementing the Sliding Window RLC FEC Scheme relies on several parameters:

Maximum FEC-related latency budget, max_lat (a decimal number expressed in seconds) with real-time flows:

a source ADU flow can have real-time constraints, and therefore any FECFRAME related operation should take place within the validity period of each ADU (Appendix D describes an exception to this rule). When there are multiple flows with different real-time constraints, we consider the most stringent constraints (see [RFC6363], Section 10.2, item 6, for recommendations when several flows are globally protected). The maximum FEC-related latency budget, max_lat, accounts for all sources of latency added by FEC encoding (at a sender) and FEC decoding (at a receiver). Other sources of latency (e.g., added by network communications) are out of scope and must be considered separately (said differently, they have already been deducted from max_lat). max_lat can be regarded as the latency budget permitted for all FEC-related operations.

This is an input parameter that enables a FECFRAME sender to derive other internal parameters (see Appendix C);
Encoding window current (resp. maximum) size, \( ew_{\text{size}} \) (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 se (it’s not required to process a repair symbol at a receiver). This code rate parameter can be static. However, in specific use-cases (e.g., with unicast transmissions in presence of a feedback mechanism that estimates the communication quality, out of scope of FECFRAME), the code rate may be adjusted dynamically.

Appendix C proposes non normative techniques to derive those parameters, depending on the use-case specificities.
3.2. ADU, ADUI and Source Symbols Mappings

At a sender, an ADU coming from the application is not directly mapped to source symbols. When multiple source flows (e.g., media streams) are mapped onto the same FECFRAME instance, each flow is assigned its own Flow ID value (see below). This Flow ID is then prepended to each ADU before FEC encoding. This way, FEC decoding at a receiver also recovers this Flow ID and the recovered ADU can be assigned to the right source flow (note that the 5-tuple used to identify the right source flow of a received ADU is absent with a recovered ADU since it is not FEC protected).

Additionally, since ADUs are of variable size, padding is needed so that each ADU (with its flow identifier) contribute to an integral number of source symbols. This requires adding the original ADU length to each ADU before doing FEC encoding. Because of these requirements, an intermediate format, the ADUI, or ADU Information, is considered [RFC6363].

For each incoming ADU, an ADUI MUST created as follows. First of all, 3 bytes are prepended (Figure 1):

Flow ID (F) (8-bit field): this unsigned byte contains the integer identifier associated to the source ADU flow to which this ADU belongs. It is assumed that a single byte is sufficient, which implies that no more than 256 flows will be protected by a single FECFRAME session instance.

Length (L) (16-bit field): this unsigned integer contains the length of this ADU, in network byte order (i.e., big endian). This length is for the ADU itself and does not include the F, L, or Pad fields.

Then, zero padding is added to the ADU if needed:

Padding (Pad) (variable size field): this field contains zero padding to align the F, L, ADU and padding up to a size that is multiple of E bytes (i.e., the source and repair symbol length).

The data unit resulting from the ADU and the F, L, and Pad fields is called ADUI. Since ADUs can have different sizes, this is also the case for ADUIs. However, an ADUI always contributes to an integral number of source symbols.
Figure 1: ADUI Creation example (here 3 source symbols are created for this ADUI).

Note that neither the initial 3 bytes nor the optional padding are sent over the network. However, they are considered during FEC encoding, and a receiver who lost a certain FEC Source Packet (e.g., the UDP datagram containing this FEC Source Packet when UDP is used as the transport protocol) will be able to recover the ADUI if FEC decoding succeeds. Thanks to the initial 3 bytes, this receiver will get rid of the padding (if any) and identify the corresponding ADU flow.

3.3. Encoding Window Management

Source symbols and the corresponding ADUs are removed from the encoding window:

- when the sliding encoding window has reached its maximum size, `ew_max_size`. In that case the oldest symbol MUST be removed before adding a new symbol, so that the current encoding window size always remains inferior or equal to the maximum size: `ew_size <= ew_max_size`;
- when an ADU has reached its maximum validity duration in case of a real-time flow. When this happens, all source symbols corresponding to the ADUI that expired SHOULD be removed from the encoding window;

Source symbols are added to the sliding encoding window each time a new ADU arrives, once the ADU-to-source symbols mapping has been performed (Section 3.2). The current size of the encoding window, `ew_size`, is updated after adding new source symbols. This process may require to remove old source symbols so that: `ew_size <= 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^{28}), 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^{28}) 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].

```c
#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
 * (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^{28})), otherwise
 *                    a fraction of them will be 0.
 * (in) m             Finite Field GF(2^{m}) parameter. In this
 *                    document only values 1 and 8 are considered.
 * (out)              returns 0 in case of success, an error code
 *                    different than 0 otherwise.
 */
```

Roca & Teibi Expires December 20, 2019 [Page 13]
int generate_coding_coefficients (uint16_t  repair_key,
    uint8_t*  cc_tab,
    uint16_t  cc_nb,
    uint8_t   dt,
    uint8_t   m)
{
    uint32_t      i;
    tinymt32_t    s;    /* PRNG internal state */

    if (dt > 15) {
        return -1; /* error, bad dt parameter */
    }

    switch (m) {
    case 1:
        if (dt == 15) {
            /* all coefficients are 1 */
            memset(cc_tab, 1, cc_nb);
        } else {
            /* here coefficients are either 0 or 1 */
            tinymt32_init(&s, repair_key);
            for (i = 0 ; i < cc_nb ; i++) {
                cc_tab[i] = (tinymt32_rand16(&s) <= dt) ? 1 : 0;
            }
        }
        break;
    case 8:
        tinymt32_init(&s, repair_key);
        if (dt == 15) {
            /* coefficient 0 is avoided here in order to include
                * all the source symbols */
            for (i = 0 ; i < cc_nb ; i++) {
                do {
                    cc_tab[i] = (uint8_t) tinymt32_rand256(&s);
                } while (cc_tab[i] == 0);
            }
        } else {
            /* here a certain number of coefficients should be 0 */
            for (i = 0 ; i < cc_nb ; i++) {
                if (tinymt32_rand16(&s) <= dt) {
                    do {
                        cc_tab[i] = (uint8_t) tinymt32_rand256(&s);
                    } while (cc_tab[i] == 0);
                } else {
                    cc_tab[i] = 0;
                }
            }
        }
    }
3.7. Finite Fields Operations

3.7.1. Finite Field Definitions

The two RLC FEC Schemes specified in this document reuse the Finite Fields defined in [RFC5510], section 8.1. More specifically, the elements of the field GF(2\(^m\)) are represented by polynomials with binary coefficients (i.e., over GF(2)) and degree lower or equal to m-1. The addition between two elements is defined as the addition of binary polynomials in GF(2), which is equivalent to a bitwise XOR operation on the binary representation of these elements.

With GF(2\(^8\)), multiplication between two elements is the multiplication modulo a given irreducible polynomial of degree 8. The following irreducible polynomial is used for GF(2\(^8\)):

\[ x^8 + x^4 + x^3 + x^2 + 1 \]

With GF(2), multiplication corresponds to a logical AND operation.

3.7.2. Linear Combination of Source Symbols Computation

The two RLC FEC Schemes require the computation of a linear combination of source symbols, using the coding coefficients produced by the generate_coding_coefficients() function and stored in the cc_tab[] array.

With the RLC over GF(2\(^8\)) FEC Scheme, a linear combination of the ew_size source symbol present in the encoding window, say src_0 to src_ew_size_1, in order to generate a repair symbol, is computed as follows. For each byte of position i in each source and the repair symbol, where i belongs to [0; E-1], compute:

\[
\text{repair}[i] = \text{cc_tab}[0] \times \text{src}_0[i] \text{ XOR } \text{cc_tab}[1] \times \text{src}_1[i] \text{ XOR } \ldots \text{ XOR } \text{cc_tab}[\text{ew_size} - 1] \times \text{src_ew_size}_1[i]
\]
where * is the multiplication over GF(2\(^8\)). In practice various optimizations need to be used in order to make this computation efficient (see in particular [PGM13]).

With the RLC over GF(2) FEC Scheme (binary case), a linear combination is computed as follows. The repair symbol is the XOR sum of all the source symbols corresponding to a coding coefficient \(cc_{tab}[j]\) equal to 1 (i.e., the source symbols corresponding to zero coding coefficients are ignored). The XOR sum of the byte of position \(i\) in each source is computed and stored in the corresponding byte of the repair symbol, where \(i\) belongs to \([0; E-1]\). In practice, the XOR sums will be computed several bytes at a time (e.g., on 64 bit words, or on arrays of 16 or more bytes when using SIMD CPU extensions).

With both FEC Schemes, the details of how to optimize the computation of these linear combinations are of high practical importance but out of scope of this document.

4. Sliding Window RLC FEC Scheme over GF(2\(^8\)) for Arbitrary Packet Flows

This fully-specified FEC Scheme defines the Sliding Window Random Linear Codes (RLC) over GF(2\(^8\)).

4.1. Formats and Codes

4.1.1. FEC Framework Configuration Information

Following the guidelines of [RFC6363], section 5.6, this section provides the FEC Framework Configuration Information (or 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.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Encoding Symbol Length (E)   |     WSR     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: FSSI Encoding Format
4.1.2. Explicit Source FEC Payload ID

A FEC Source Packet MUST contain an Explicit Source FEC Payload ID that is appended to the end of the packet as illustrated in Figure 5.

```
+--------------------------------+  
|           IP Header            |  
+--------------------------------+  
|        Transport Header        |  
+--------------------------------+  
|              ADU               |  
+--------------------------------+  
| Explicit Source FEC Payload ID |  
+--------------------------------+  
```

Figure 5: Structure of an FEC Source Packet with the Explicit Source FEC Payload ID

More precisely, the Explicit Source FEC Payload ID is composed of the following field, carried in "big-endian" or "network order" format, that is, most significant byte first (Figure 6):

Encoding Symbol ID (ESI) (32-bit field): this unsigned integer identifies the first source symbol of the ADUI corresponding to this FEC Source Packet. The ESI is incremented for each new source symbol, and after reaching the maximum value (2^32-1), wrapping to zero occurs.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Encoding Symbol ID (ESI)                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6: Source FEC Payload ID Encoding Format

4.1.3. Repair FEC Payload ID

A FEC Repair Packet MAY contain one or more repair symbols. When there are several repair symbols, all of them MUST have been generated from the same encoding window, using Repair Key values that are managed as explained below. A receiver can easily deduce the number of repair symbols within a FEC Repair Packet by comparing the received FEC Repair Packet size (equal to the UDP payload size when UDP is the underlying transport protocol) and the symbol size, E, communicated in the FFCI.
A FEC Repair Packet MUST contain a Repair FEC Payload ID that is prepended to the repair symbol as illustrated in Figure 7.

```
+--------------------------------+   |           IP Header            |
|                               |   +--------------------------------+   |        Transport Header         |
|                               |   +--------------------------------+   |     Repair FEC Payload ID      |
|                               |   +--------------------------------+   |         Repair Symbol          |
|  Figure 7: Structure of an FEC Repair Packet with the Repair FEC Payload ID |
```

More precisely, the Repair FEC Payload ID is composed of the following fields where all integer fields are carried in "big-endian" or "network order" format, that is, most significant byte first (Figure 8):

- **Repair Key (16-bit field):** this unsigned integer is used as a seed by the coefficient generation function (Section 3.6) in order to generate the desired number of coding coefficients. This repair key may be a monotonically increasing integer value that loops back to 0 after reaching 65535 (see Section 6.1). When a FEC Repair Packet contains several repair symbols, this repair key value is that of the first repair symbol. The remaining repair keys can be deduced by incrementing by 1 this value, up to a maximum value of 65535 after which it loops back to 0.
- **Density Threshold for the coding coefficients, DT (4-bit field):** this unsigned integer carries the Density Threshold (DT) used by the coding coefficient generation function Section 3.6. More precisely, it controls the probability of having a non zero coding coefficient, which equals \((DT+1) / 16\). When a FEC Repair Packet contains several repair symbols, the DT value applies to all of them;
- **Number of Source Symbols in the encoding window, NSS (12-bit field):** this unsigned integer indicates the number of source symbols in the encoding window when this repair symbol was generated. When a FEC Repair Packet contains several repair symbols, this NSS value applies to all of them;
- **ESI of First Source Symbol in the encoding window, FSS_ESI (32-bit field):** this unsigned integer indicates the ESI of the first source symbol in the encoding window when this repair symbol was generated.
When a FEC Repair Packet contains several repair symbols, this FSS_ESI value applies to all of them;

```
| 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.

Encoding symbol length (E): setting this E parameter to a different value will confuse a receiver. If the size of a received FEC Repair Packet is no longer multiple of the modified E value, a receiver quickly detects a problem and SHOULD reject the packet. If the new E value is a sub-multiple of the original E value (e.g., half the original value), then receivers may not detect the problem immediately. For instance, a receiver may think that a received FEC Repair Packet contains more repair symbols (e.g., twice as many if E is reduced by half), leading to malfunctions whose nature depends on implementation details. Here also, the attack always results in a form of DoS or corrupted content.

It is therefore RECOMMENDED that security measures be taken to guarantee the FFCI integrity, as specified in [RFC6363]. How to achieve this depends on the way the FFCI is communicated from the sender to the receiver, which is not specified in this document.

Similarly, attacks are possible against the Explicit Source FEC Payload ID and Repair FEC Payload ID. More specifically, in case of a FEC Source Packet, the following value can be modified by an attacker who targets receivers:

- 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:

o YYYY refers to the Sliding Window Random Linear Codes (RLC) over GF(2) FEC Scheme for Arbitrary Packet Flows, as defined in Section 5 of this document.

o XXXX refers to the Sliding Window Random Linear Codes (RLC) over GF(2**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


Internet-Draft               RLC FEC Scheme                    June 2019


[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/>.


Roca & Teibi            Expires December 20, 2019              [Page 29]
Appendix A. TinyMT32 Validation Criteria (Normative)

PRNG determinism, for a given seed, is a requirement. Consequently, in order to validate an implementation of the TinyMT32 PRNG, the following criteria MUST be met.

The first criterion focusses on the `tinymt32_rand256()`, where the 32-bit integer of the core TinyMT32 PRNG is scaled down to an 8-bit integer. Using a seed value of 1, the first 50 values returned by `tinymt32_rand256()` as 8-bit unsigned integers MUST be equal to values provided in Figure 9, to be read line by line.

| 37 | 225 | 177 | 176 | 21 |
| 246 | 54 | 139 | 168 | 237 |
| 211 | 187 | 62 | 190 | 104 |
| 135 | 210 | 99 | 176 | 11 |
| 207 | 35 | 40 | 113 | 179 |
| 214 | 254 | 101 | 212 | 211 |
| 226 | 41 | 234 | 232 | 203 |
| 29 | 194 | 211 | 112 | 107 |
| 217 | 104 | 197 | 135 | 23 |
| 89 | 210 | 252 | 109 | 166 |

Figure 9: First 50 decimal values (to be read per line) returned by `tinymt32_rand256()` as 8-bit unsigned integers, with a seed value of 1.

The second criterion focusses on the `tinymt32_rand16()`, where the 32-bit integer of the core TinyMT32 PRNG is scaled down to a 4-bit integer. Using a seed value of 1, the first 50 values returned by `tinymt32_rand16()` as 4-bit unsigned integers MUST be equal to values provided in Figure 10, to be read line by line.

| 5 | 1 | 1 | 0 | 5 |
| 6 | 6 | 11 | 8 | 13 |
| 3 | 11 | 14 | 14 | 8 |
| 7 | 2 | 3 | 0 | 11 |
| 15 | 3 | 8 | 1 | 3 |
| 6 | 14 | 5 | 4 | 3 |
| 2 | 9 | 10 | 8 | 11 |
| 13 | 2 | 3 | 0 | 11 |
| 9 | 8 | 5 | 7 | 7 |
| 9 | 2 | 12 | 13 | 6 |

Figure 10: First 50 decimal values (to be read per line) returned by `tinymt32_rand16()` as 4-bit unsigned integers, with a seed value of 1.
Appendix B. Assessing the PRNG Adequacy (Informational)

This annex discusses the adequacy of the TinyMT32 PRNG and the
\texttt{tinymt32\_rand16()} and \texttt{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 \texttt{tinymt32\_rand16()} or \texttt{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 \texttt{tinymt32\_rand16()}: We first generate a huge number
(1,000,000,000) of small sequences (20 pseudo-random numbers per
sequence), increasing the seed value for each sequence, and perform
statistics on the number of occurrences of each of the 16 possible
values across all sequences. In this first test we consider 32-bit
seed values in order to assess the PRNG quality after output
truncation to 4 bits.
value    occurrences       percentage (%) (total of 20000000000)
0        1250036799        6.2502
1        1249995831        6.2500
2        1250038674        6.2502
3        1250000881        6.2500
4        1250023929        6.2501
5        1249986320        6.2499
6        1249995587        6.2500
7        1250020363        6.2501
8        1249995276        6.2500
9        1249982856        6.2499
10       1249984111        6.2499
11       1250009551        6.2500
12       1249955768        6.2498
13       1249994654        6.2500
14       1250000569        6.2500
15       1249978831        6.2499

Figure 11: tinymt32_rand16(): occurrence statistics across a huge number (1,000,000,000) of small sequences (20 pseudo-random numbers per sequence), with 0 as the first PRNG seed.

The results (Figure 11) show that all possible values are almost equally represented, or said differently, that the tinymt32_rand16() output converges to a uniform distribution where each of the 16 possible values would appear exactly 1 / 16 * 100 = 6.25% of times.

Since the RLC FEC Schemes use of this PRNG will be limited to 16-bit seed values, we carried out the same test for the first $2^{16}$ seed values only. The distribution (not shown) is of course less uniform, with value occurrences ranging between 6.2121% (i.e., 81,423 occurrences out of a total of 65536*20=1,310,720) and 6.2948% (i.e., 82,507 occurrences). However, we do not believe it significantly impacts the RLC FEC Scheme behavior.

Other types of biases may exist that may be visible with smaller tests, for instance to evaluate the convergence speed to a uniform distribution. We therefore perform 200 tests, each of them consisting in producing 200 sequences, keeping only the first value of each sequence. We use non overlapping repair keys for each sequence, starting with value 0 and increasing it after each use.
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:

\[ dw_{\text{max \_size}} = \frac{\text{max \_lat} \times \text{br\_in}}{8 \times E} \]

assuming that the encoding and decoding times are negligible with respect to the target max_lat. This is a reasonable assumption in many situations (e.g., see Section 9.1 in case of small window sizes). Otherwise the max_lat parameter should be adjusted in order to avoid the problem. In any case, interoperability will never be compromised by choosing a too large value.

In a second configuration, the FECFRAME sender generates a fixed bitrate flow, equal to the CBR communication path bitrate equal to br_out (in bits/s), and this value is known by the FECFRAME sender,
as in [Roca17]. The maximum source flow bitrate needs to be such that, with the added repair flow overhead, the total transmission bitrate remains inferior or equal to br_out. We have:

\[
dw_{\text{max}} = \frac{(\text{max}
lat \times \text{br}_\text{out} \times \text{cr})}{(8 \times E)}
\]

assuming here also that the encoding and decoding times are negligible with respect to the target max_lat.

For decoding to be possible within the latency budget, it is required that the encoding window maximum size be smaller than or at most equal to the decoding window maximum size. The ew_max_size is the main parameter at a FECFRAME sender, but its exact value has no impact on the the FEC-related latency budget. The ew_max_size parameter is computed as follows:

\[
\text{ew}_{\text{max}} = \text{dw}_{\text{max}} \times \text{WSR} / 255
\]

In line with [Roca17], WSR = 191 is considered as a reasonable value (the resulting encoding to decoding window size ratio is then close to 0.75), but other values between 1 and 255 inclusive are possible, depending on the use-case.

The dw_max_size is computed by a FECFRAME sender but not explicitly communicated to a FECFRAME receiver. However, a FECFRAME receiver can easily evaluate the ew_max_size by observing the maximum Number of Source Symbols (NSS) value contained in the Repair FEC Payload ID of received FEC Repair Packets (Section 4.1.3). A receiver can then easily compute dw_max_size:

\[
dw_{\text{max}} = \text{max}_\text{ NSS}_\text{observed} \times 255 / \text{WSR}
\]

A receiver can then chose an appropriate linear system maximum size:

\[
\text{ls}_{\text{max}} \geq \text{dw}_{\text{max}}
\]

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:

\[
\text{max\_lat\_for\_encoding} = \text{max\_lat} * \frac{\text{WSR}}{255}
\]

It follows that any source symbols associated to an ADU that has timed-out with respect to max_lat_for_encoding SHOULD be removed from the encoding window. With this approach there is no pre-determined ew_size value: this value fluctuates over the time according to the instantaneous source ADU flow bitrate. For practical reasons, a FECFRAME sender may still require that ew_size does not increase beyond a maximum value (Appendix C.3).
With both approaches, and no matter the choice of the FECFRAME sender, a FECFRAME receiver can still easily evaluate the ew_max_size by observing the maximum Number of Source Symbols (NSS) value contained in the Repair FEC Payload ID of received FEC Repair Packets. A receiver can then compute dw_max_size and derive an appropriate ls_max_size as explained in Appendix C.1.

When the observed NSS fluctuates significantly, a FECFRAME receiver may want to adapt its ls_max_size accordingly. In particular when the NSS is significantly reduced, a FECFRAME receiver may want to reduce the ls_max_size too in order to limit computation complexity. A balance must be found between using an ls_max_size "too large" (which increases computation complexity and memory requirements) and the opposite (which reduces recovery performance).

C.3. Case of a Non Real-Time Flow

Finally there are configurations where a source ADU flow has no real-time constraints. FECFRAME and the FEC Schemes defined in this document can still be used. The choice of appropriate parameter values can be directed by practical considerations. For instance, it can derive from an estimation of the maximum memory amount that could be dedicated to the linear system at a FECFRAME receiver, or the maximum computation complexity at a FECFRAME receiver, both of them depending on the ls_max_size parameter. The same considerations also apply to the FECFRAME sender, where the maximum memory amount and computation complexity depend on the ew_max_size parameter.

Here also, the NSS value contained in FEC Repair Packets is used by a FECFRAME receiver to determine the current coding window size and ew_max_size by observing its maximum value over the time.

Appendix D. Decoding Beyond Maximum Latency Optimization (Informational)

This annex introduces non normative considerations. It is provided as suggestions, without any impact on interoperability. For more information see [Roca16].

With a real-time source ADU flow, it is possible to improve the decoding performance of sliding window codes without impacting maximum latency, at the cost of extra memory and CPU overhead. The optimization consists, for a FECFRAME receiver, to extend the linear system beyond the decoding window maximum size, by keeping a certain number of old source symbols whereas their associated ADUs timed-out:

\[ \text{ls}_\text{max}_\text{size} > \text{dw}_\text{max}_\text{size} \]
Usually the following choice is a good trade-off between decoding performance and extra CPU overhead:

ls_max_size = 2 * dw_max_size

When the dw_max_size is very small, it may be preferable to keep a minimum ls_max_size value (e.g., LS_MIN_SIZE_DEFAULT = 40 symbols). Going below this threshold will not save a significant amount of memory nor CPU cycles. Therefore:

ls_max_size = max(2 * dw_max_size, LS_MIN_SIZE_DEFAULT)

Finally, it is worth noting that a receiver that benefits from an FEC protection significantly higher than what is required to recover from packet losses, can choose to reduce the ls_max_size. In that case lost ADUs will be recovered without relying on this optimization.

ls_max_size

/---------------------------------^-------------------------------\

late source symbols
(pot. decoded but not delivered) dw_max_size

/-------------^-----------------\ /-------------^-----------------\ src0 src1 src2 src3 src4 src5 src6 src7 src8 src9 src10 src11 src12

Figure 13: Relationship between parameters to decode beyond maximum latency.

It means that source symbols, and therefore ADUs, may be decoded even if the added latency exceeds the maximum value permitted by the application (the "late source symbols" of Figure 13). It follows that the corresponding ADUs will not be useful to the application. However, decoding these "late symbols" significantly improves the global robustness in bad reception conditions and is therefore recommended for receivers experiencing bad communication conditions [Roca16]. In any case whether or not to use this optimization and what exact value to use for the ls_max_size parameter are local decisions made by each receiver independently, without any impact on the other receivers nor on the source.

Authors’ Addresses

Vincent Roca
INRIA
Univ. Grenoble Alpes
France

EMail: vincent.roca@inria.fr
The Impact of Transport Header Confidentiality on Network Operation and Evolution of the Internet
draft-ietf-tsvwg-transport-encrypt-06

Abstract

This document describes implications of applying end-to-end encryption at the transport layer. It identifies in-network uses of transport layer header information. It then reviews the implications of developing end-to-end transport protocols that use authentication to protect the integrity of transport information or encryption to provide confidentiality of the transport protocol header and expected implications of transport protocol design and network operation. Since transport measurement and analysis of the impact of network characteristics have been important to the design of current transport protocols, it also considers the impact on transport and application evolution.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 2, 2019.

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, new protocols that employ end-to-end encryption at the transport layer, including the transport layer headers. An example of such a transport is the QUIC transport protocol [I-D.ietf-quic-transport], currently being standardised in the IETF. Encryption of transport layer headers and payload data has many benefits in terms of protecting user privacy. These benefits have been widely discussed [RFC7258], [RFC7624], and this document strongly supports the increased use of encryption in transport protocols. 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.

This document discusses some consequences of applying end-to-end encryption at the transport layer. It reviews the implications of developing end-to-end transport protocols that use encryption to provide confidentiality of the transport protocol header, and considers the effect of such changes on transport protocol design and network operations. It also considers anticipated implications on transport and application evolution.

The remainder of this document discusses some consequences of applying end-to-end encryption at the transport layer. It reviews the implications of developing end-to-end transport protocols that use encryption to provide confidentiality of the transport protocol header, and considers the effect of such changes on transport protocol design and network operations. It also considers anticipated implications on transport and application evolution.

Transports are increasingly encrypting and authenticating the payload (i.e., the application data carried within the transport connection) end-to-end. Such protection is encouraged, and the implications of protecting the payload are not further discussed in this memo.

2. Context and Rationale

The transport layer provides end-to-end interactions between endpoints (processes) using an Internet path. Transport protocols layer directly over the network-layer service and are sent in the payload of network-layer packets. They support end-to-end communication between applications, supported by higher-layer protocols, running on the end systems (or transport endpoints). This simple architectural view hides one of the core functions of the transport, however, to discover and adapt to the properties of the Internet path that is currently being used. The design of Internet transport protocols is as much about trying to avoid the unwanted side effects of congestion on a flow and other capacity-sharing flows, avoiding congestion collapse, adapting to changes in the path characteristics, etc., as it is about end-to-end feature negotiation, flow control and optimising for performance of a specific application.

To achieve stable Internet operations the IETF transport community has to date relied heavily on measurement and 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 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.

There are many motivations for deploying encrypted transports [RFC7624] (i.e., transport protocols that use encryption to provide confidentiality of some or all of the transport-layer header information), and encryption of transport payloads (i.e., Confidentiality of the payload data). The increasing public concerns about interference with Internet traffic have led to a rapidly expanding deployment of encryption to protect end-user privacy, e.g., QUIC [I-D.ietf-quic-transport]. Encryption is also expected to form a basis for future transport protocol designs.

Some network operators and access providers have come to rely on the in-network measurement of transport properties and the functionality provided by middleboxes to both support network operations and enhance performance. There can therefore be implications when working with encrypted transport protocols that hide transport header information from the network. These present architectural challenges and considerations in the way transport protocols are designed, and ability to characterise and compare different transport solutions [Measure]. Implementations of network devices are encouraged to avoid side-effects when protocols are updated. Introducing cryptographic integrity checks to header fields can also prevent undetected manipulation of the field by network devices, or undetected addition of information to a packet. However, this does not prevent inspection of the information by a device on path, and it is possible that such devices could develop mechanisms that rely on the presence of such a field or a known value in the field.

Reliance on the presence and semantics of specific header information leads to ossification. 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 this is not beneficial (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 could prevent the device from forwarding packets using a different version of a protocol that introduces a new feature that changes the value present in this field, preventing the evolution of the protocol). Experience developing Transport Layer
Security [RFC8446], required a design that recognised that deployed middleboxes relied on the exposed information in TLS 1.2

Examples of the impact of ossification on transport protocol design and ease of deployment can be seen in the case of Multipath TCP (MPTCP) and the TCP Fast Open option. The design of MPTCP had to be revised to account for middleboxes, so called "TCP Normalizers", that monitor the evolution of the window advertised in the TCP headers and that reset connections if the window does not grow as expected. Similarly, TCP Fast Open has had issues with middleboxes that remove unknown TCP options, that drop segments with unknown TCP options, that drop segments that contain data and have the SYN bit set, that drop packets with SYN/ACK that acknowledge data, or that disrupt connections that send data before the three-way handshake completes. In both cases, the issue was caused by middleboxes that had a hard-coded understanding of transport behaviour, and that interacted poorly with transports that tried to change that behaviour. Other examples have included middleboxes that rewrite TCP sequence and acknowledgement numbers but are unaware of the (newer) SACK option and don’t correctly rewrite selective acknowledgements to match the changes made to the fixed TCP header.

A protocol design that uses header encryption can provide confidentiality of some or all of the protocol header information. Encryption with secure key distribution prevents an on-path device from observing the header field. It, therefore, prevents mechanisms being built that directly rely on the information or seek to infer semantics of an exposed header field. Using encryption to provide confidentiality of the transport layer brings some well-known privacy and security benefits and can therefore help reduce ossification of the transport layer. In particular, it is important that protocols either do not expose information where the usage could change in future protocols or that methods that utilise the information are robust to potential changes as protocols evolve over time. To avoid unwanted inspection, a protocol could also intentionally vary the format and/or value of header fields (sometimes known as Greasing [I-D.thomson-quic-grease]). However, while encryption hides the protocol header information, it does not prevent ossification of the network service. People seeking to understand network traffic could come to rely on pattern inferences and other heuristics as the basis for network decision and to derive measurement data, creating new dependencies on the transport protocol.

Specification of non-encrypted transport header fields explicitly allows protocol designers to make specific header information observable in the network. This supports other uses of this information by on-path devices, and at the same time this can be expected to lead to ossification of the transport header, because
network forwarding could evolve to depend on the presence and/or value of these fields. 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. For example, a design that provides confidentiality of protocol header information can impact the following activities that rely on measurement and analysis of traffic flows:

Network Operations and Research: Observable transport headers enable both operators and the research community to explicitly measure and analyse protocol performance, network anomalies, and failure pathologies.

This information can help inform capacity planning and assist in determining the need for equipment and/or configuration changes by network operators.

The data can also inform Internet engineering research, and help in the development of new protocols, methodologies, and procedures. Concealing the transport protocol header information makes the stream performance unavailable to passive observers along the path, and likely leads to the development of alternative methods to collect or infer that data (for example heuristics based on the analysis of traffic patterns).

Providing confidentiality of the transport payload, but leaving some, or all, of the transport headers unencrypted, possibly with authentication, can provide many of the privacy and security benefits while supporting operations and research, but at the cost of ossifying the transport headers.

Protection from Denial of Service: Observable transport headers currently provide useful input to classify traffic and detect anomalous events (e.g., changes in application behaviour, distributed denial of service attacks). To be effective, this protection needs to be able to uniquely disambiguate unwanted traffic. An inability to separate this traffic using packet header information could result in less-efficient identification of unwanted traffic or development of different methods (e.g. rate-limiting of uncharacterised traffic).

Network Troubleshooting and Diagnostics: Encrypting transport header information eliminates the incentive for operators to troubleshoot since they cannot interpret the data. A flow experiencing packet loss or jitter looks like an unaffected flow when only observing network layer headers (if transport sequence numbers and flow identifiers are obscured). This limits
understanding of the impact of packet loss or latency on the flows, or even localizing the network segment causing the packet loss or latency. Encrypted traffic could imply "don’t touch" to some, and could limit a trouble-shooting response to "can’t help, no trouble found". Additional mechanisms will need to be introduced to help reconstruct or replace transport-level metrics to support troubleshooting and diagnostics, but these add complexity and operational costs (e.g., in deploying additional functions in equipment or adding traffic overhead).

Network Traffic Analysis: Hiding transport protocol header information can make it harder to determine which transport protocols and features are being used across a network segment and to measure trends in the pattern of usage. This could impact the ability for an operator to anticipate the need for network upgrades and roll-out. It can also impact the on-going traffic engineering activities performed by operators (such as determining which parts of the path contribute delay, jitter or loss). While this impact could, in many cases, be small, there are scenarios where operators directly support particular services (e.g., to troubleshoot issues relating to Quality of Service, QoS; the ability to perform fast re-routing of critical traffic, or support to mitigate the characteristics of specific radio links). The more complex the underlying infrastructure the more important this impact.

Open and Verifiable Network Data: Hiding transport protocol header information can reduce the range of actors that can capture useful measurement data. This limits the information sources available to the Internet community to understand the operation of new transport protocols, so preventing access to the information necessary 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. The ability of other stakeholders to review transport header traces can help develop deeper insight into performance and traffic contribution of specific variants of a protocol. In the heterogeneous Internet, this helps extend the range of topologies, vendor equipment, and traffic patterns that are evaluated.

Independently captured data is important to help ensure the health of the research and development communities. It can provide input and test scenarios to support the development of new transport
protocol mechanisms, especially when this analysis can be based on the behaviour experienced in a diversity of deployed networks.

Independently verifiable performance metrics might also be utilised to demonstrate regulatory compliance in some jurisdictions, and to provide a basis for informing design decisions.

The last point leads us to consider the impact of hiding transport headers in the specification and development of protocols and standards. This has a potential impact on:

- **Understanding Feature Interactions:** An appropriate vantage point, coupled with timing information about traffic flows, provides a valuable tool for benchmarking equipment, functions, and/or configurations, and to 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.

- **Supporting Common Specifications:** Transmission Control Protocol (TCP) is currently the predominant transport protocol used over Internet paths. Its many variants have broadly consistent approaches to avoiding congestion collapse, and to ensuring the stability of the Internet. Increased use of transport layer encryption can overcome ossification, allowing deployment of new transports and different types of congestion control. This flexibility can be beneficial, but it can come at the cost of fragmenting the ecosystem. There is little doubt that developers will try to produce high quality transports for their intended target uses, but it is not clear there are sufficient incentives to ensure good practice that benefits the wide diversity of requirements for the Internet community as a whole. 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 specifications.

- **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. If this information is only carried in encrypted transport headers, operators will not be able to use this information directly. If operators still wish to use these practices, they may turn to more
ambitious ways of discovering this information. For example, if an operator wants to know that traffic is audio traffic, and no longer has access to Session Description Protocol (SDP) session descriptions that would explicitly say a flow "is audio", the operator might use heuristics to guess that short UDP packets with regular spacing are carrying audio traffic. Operational practices aimed at guessing transport parameters are out of scope for this document, and are only mentioned here to recognize that encryption may not prevent operators from attempting to apply the same practices they used with unencrypted transport headers.

- Compliance: Published transport specifications allow operators and regulators to check compliance. 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 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.

- Restricting research and development: Hiding 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).

In summary, there are trade-offs. On the one hand, transport protocol designers have often ignored the implications of whether the information in transport header fields can or will be used by in-network devices, and the implications this places on protocol evolution. This motivates a design that provides confidentiality of the header information. On the other hand, it can be expected that a lack of visibility of transport header information can impact the ways that protocols are deployed, standardised, and their operational support.
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 choice of whether future transport protocols encrypt their protocol headers therefore 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. An appropriate balance will emerge over time as real instances of this tension are considered." This balance between information exposed and information concealed ought to be carefully considered when specifying new transport protocols.

3. Current uses of Transport Headers within the Network

Despite transport headers having end-to-end meaning, some of these transport headers have come to be used in various ways within the Internet. In response to pervasive monitoring [RFC7624] revelations and the IETF consensus that "Pervasive Monitoring is an Attack" [RFC7258], efforts are underway to increase encryption of Internet traffic. Applying confidentiality to transport header fields would affect how protocol information is used [RFC8404]. To understand these implications, it is first necessary to understand how transport layer headers are currently observed and/or modified by middleboxes within the network.

Transport protocols can be designed to encrypt or authenticate transport header fields. Authentication at the transport layer can be used to detect any changes to an immutable header field that were made by a network device along a path. The intentional modification of transport headers by middleboxes (such as Network Address Translation, NAT, or Firewalls) is not considered. Common issues concerning IP address sharing are described in [RFC6269].

3.1. Observing Transport Information in the Network

If in-network observation of transport protocol headers is needed, this requires knowledge of the format of the transport header:

- Flows need to be identified at the level required to perform the observation;
o The protocol and version of the header need to be visible, e.g.,
by defining the wire image [I-D.trammell-wire-image]. As
protocols evolve over time and there could be a need to introduce
new transport headers. This could require interpretation of
protocol version information or connection setup information;

o 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

Transport protocol header information (together with information in
the network header), has been used to identify a flow and the
connection state of the flow, together with the protocol options
being used. In some usages, a low-numbered (well-known) transport
port number has been used to identify a protocol (although port
information alone is not sufficient to guarantee identification of a
protocol, since applications can use arbitrary ports, multiple
sessions can be multiplexed on a single port, and ports can be re-
used by subsequent sessions).

Transport protocols, such as TCP and the Stream Control Transport
Protocol (SCTP) specify a standard base header that includes sequence
number information and other data, with the possibility to negotiate
additional headers at connection setup, identified by an option
number in the transport header. UDP-based protocols can use, but
sometimes do not use, well-known port numbers. Some flows can
instead be identified by observing signalling protocol data (e.g.,
[RFC3261], [I-D.ietf-rtcweb-overview]) or through the use of magic
numbers placed in the first byte(s) of the datagram payload
[RFC7983].

Flow identification is a common function. For example, performed by
measurement activities, QoS classification, firewalls, Denial of
Service, DOS, prevention. It becomes more complex and less easily
achieved when multiplexing is used at or above the transport layer.

3.1.2. Metrics derived from Transport Layer Headers

Some actors manage their portion of the Internet by characterizing
the performance of link/network segments. Passive monitoring can
observe traffic that does not encrypt the transport header
information to make inferences from transport headers to derive these
performance metrics. A variety of open source and commercial tools
have been deployed that utilise this information. The following metrics can be derived from transport header information:

Traffic Rate and Volume: Header information (e.g., sequence number and packet size) allows derivation of volume measures per-application, to characterise the traffic that uses a network segment or the pattern of network usage. This can be measured per endpoint or for an aggregate of endpoints (e.g., by an operator 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 rate requires either maintaining per flow packet counters or by observing sequence numbers in transport headers. Loss can be monitored at the interface level by devices in the network. It is often valuable to understand the conditions under which packet loss occurs. This usually requires relating loss to the traffic flowing on the network node/segment at the time of loss.

Observation of transport feedback information (e.g., RTP Control Protocol (RTCP) reception reports [RFC3550], TCP SACK blocks) can increase understanding of the impact of loss and help identify cases where loss could have been wrongly identified, or the transport did not require the lost packet. It is sometimes more helpful to understand the pattern of loss, than the loss rate, because losses can often occur as bursts, rather than randomly-timed events.

Throughput and Goodput: The throughput achieved by a flow can be determined even when a flow is encrypted, providing the individual flow can be identified. Goodput [RFC7928] is a measure of useful data exchanged (the ratio of useful/total volume of traffic sent
by a flow). This requires ability to differentiate loss and retransmission of packets (e.g., by observing packet sequence numbers in the TCP or the Real-time Transport Protocol, RTP, headers [RFC3550]).

Latency: Latency is a key performance metric that impacts application response time and user-perceived response time. It often indirectly impacts throughput and flow completion time. Latency determines the reaction time of the transport protocol itself, impacting flow setup, congestion control, loss recovery, and other transport mechanisms. The observed latency can have many components [Latency]. Of these, unnecessary/unwanted queuing in network buffers has often been observed as a significant factor [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 can measure the experienced round trip time (RTT) using packet sequence numbers, and acknowledgements, or by observing header timestamp information. Such information allows an observation point in the network to determine not only the path RTT, but also to measure the upstream and downstream contribution to the RTT. This could be used to locate a source of latency, e.g., by observing cases where the median RTT is much greater than the minimum RTT for a part of a path.

The service offered by network operators can benefit from latency information to understand the impact of deployment and tune deployed services. Latency metrics are key to evaluating and deploying AQM [RFC7567], DiffServ [RFC2474], and Explicit Congestion Notification (ECN) [RFC3168] [RFC8087]. Measurements could identify excessively large buffers, indicating where to deploy or configure AQM. An AQM method is often deployed in combination with other techniques, such as scheduling [RFC7567] [RFC8290] and although parameter-less methods are desired [RFC7567], current methods [RFC8290] [RFC8289] [RFC8033] often cannot scale across all possible deployment scenarios.

Variation in delay: Some network applications are sensitive to small changes in packet timing (jitter). Short and long-term delay variation can impact on the latency of a flow and hence the perceived quality of applications using the network (e.g., jitter metrics are often cited when characterising paths supporting real-time traffic). To assess the performance of such applications, it can be necessary to measure the variation in delay observed along a portion of the path [RFC3393] [RFC5481]. The requirements resemble those for the measurement of latency.
Flow Reordering: Significant packet reordering within a flow can impact time-critical applications and can be interpreted as loss by reliable transports. Many transport protocol techniques are impacted by reordering (e.g., triggering TCP retransmission or re-buffering of real-time applications). Packet reordering can occur for many reasons, from equipment design to misconfiguration of forwarding rules. Since this impacts transport performance, network tools are needed to detect and measure unwanted/excessive reordering.

There have been initiatives in the IETF transport area to reduce the impact of reordering within a transport flow, possibly leading to a reduction in the requirements for preserving ordering. These have potential to simplify network equipment design as well as the potential to improve robustness of the transport service. Measurements of reordering can help understand the present level of reordering within deployed infrastructure, and inform decisions about how to progress such mechanisms. Key performance indicators are retransmission rate, packet drop rate, sector utilisation level, a measure of reordering, peak rate, the ECN congestion experienced (CE) marking rate, etc.

Metrics have been defined that evaluate whether a network has maintained packet order on a packet-by-packet basis [RFC4737] and [RFC5236].

Techniques for measuring reordering typically observe packet sequence numbers. Some protocols provide in-built monitoring and reporting functions. Transport fields in the RTP header [RFC3550] [RFC4585] can be observed to derive traffic volume measurements and provide information on the progress and quality of a session using RTP. As with other measurement, metadata is often needed to understand the context under which the data was collected, including the time, observation point, 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.

The above passively monitor transport protocol headers to derive metrics about network layer performance useful for operation and management of a network.

3.1.3. Transport use of Network Layer Header Fields

Information from the transport protocol can be used by a multi-field classifier as a part of policy framework. Policies are commonly used for management of the QoS or Quality of Experience (QoE) in resource-
constrained networks and by firewalls that use the information to implement access rules (see also section 2.2.2 of [RFC8404]). Network-layer classification methods that rely on a multi-field classifier (e.g. Inferring QoS from the 5-tuple or choice of application protocol) are incompatible with transport protocols that encrypt the transport information. Traffic that cannot be classified, will typically receive a default treatment.

Transport information can also be explicitly set in network-layer header fields that are not encrypted. This can provide information to enable a different forwarding treatment by the network, even when a transport employs encryption to protect other header information.

On the one hand, the user of a transport that multiplexes multiple sub-flows could wish to hide the presence and characteristics of these sub-flows. On the other hand, an encrypted transport could set the network-layer information to indicate the presence of sub-flows and to reflect the network needs of individual sub-flows. There are several ways this could be done:

IP Address: Applications expose the addresses used by endpoints, and this is used in the forwarding decisions in network devices. Address and other protocol information can be used by a Multi-Field (MF) classifier to determine how traffic is treated [RFC2475], and hence the quality of experience for a flow.

Using the IPv6 Network-Layer Flow Label: A number of Standards Track and Best Current Practice RFCs (e.g., [RFC8085], [RFC6437], [RFC6438]) encourage endpoints to set the IPv6 Flow label field of the network-layer header. IPv6 "source nodes SHOULD assign each unrelated transport connection and application data stream to a new flow" [RFC6437]. A multiplexing transport could choose to use multiple Flow labels to allow the network to independently forward subflows. RFC6437 provides further guidance on choosing a flow label value, stating these "should be chosen such that their bits exhibit a high degree of variability", and chosen so that "third parties should be unlikely to be able to guess the next value that a source of flow labels will choose". To promote privacy, the Flow Label assignment needs to avoid introducing linkability that a network device may observe. Once set, a label can provide information that can help inform network-layer queuing and forwarding [RFC6438](e.g. for Equal Cost Multi-Path, ECMP, routing, and Link Aggregation, LAG) [RFC6294]. [RFC6438] describes considerations when used with IPsec.

Using the Network-Layer Differentiated Services Code Point: Applications can expose their delivery expectations to the network by setting the Differentiated Services Code Point (DSCP) field of
IPv4 and IPv6 packets [RFC2474]. For example, WebRTC applications identify different forwarding treatments for individual sub-flows (audio vs. video) based on the value of the DSCP field [I-D.ietf-tsvwg-rtcweb-qos]). This provides explicit information to inform network-layer queuing and forwarding, rather than an operator inferring traffic requirements from transport and application headers via a multi-field classifier.

Since the DSCP value can impact the quality of experience for a flow, observations of service performance need to consider this 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 Congestion Experienced (CE) marking of IP packets [RFC8087], since it can react to congestion without also having to recover from lost packets.

ECN exposes the presence of congestion. The reception of CE-marked packets can be used to estimate the level of incipient congestion on the upstream portion of the path from the point of observation (Section 2.5 of [RFC8087]). Interpreting the marking behaviour (i.e., assessing congestion and diagnosing faults) requires context from the transport layer (such as path RTT).

AQM and ECN offer a range of algorithms and configuration options. Tools therefore need to be available to network operators and researchers to understand the implication of configuration choices and transport behaviour as the use of ECN increases and new methods emerge [RFC7567].

Careful use of the network layer features can therefore help address some of the reasons why the network inspects transport protocol headers.

3.2. Transport Measurement

The common language between network operators and application/content providers/users is packet transfer performance at a layer that all can view and analyse. For most packets, this has been the transport layer, until the emergence of QUIC, with the obvious exception of Virtual Private Networks (VPNs) and IPSec.
When encryption conceals more layers in each packet, people seeking understanding of the network operation rely more on pattern inferences and other heuristics reliance on pattern inferences and accuracy suffers. For example, the traffic patterns between server and browser are dependent on browser supplier and version, even when the sessions use the same server application (e.g., web e-mail access). It remains to be seen whether more complex inferences can be mastered to produce the same monitoring accuracy (see section 2.1.1 of [RFC8404]).

When measurement datasets are made available by servers or client endpoints, additional metadata, such as the state of the network, is often required to interpret this data to answer questions about network performance or understand a pathology. Collecting and coordinating such metadata is more difficult when the observation point is at a different location to the bottleneck/device under evaluation.

Packet sampling techniques are used to scale the processing involved in observing packets on high rate links. This exports only the packet header information of (randomly) selected packets. The utility of these measurements depends on the type of bearer and number of mechanisms used by network devices. Simple routers are relatively easy to manage, a device with more complexity demands understanding of the choice of many system parameters. This level of complexity exists when several network methods are combined.

This section discusses topics concerning observation of transport flows, with a focus on transport measurement.

### 3.2.1. Point of Observation

On-path measurements are particularly useful for locating the source of problems, or to assess the performance of a network segment or a particular device configuration. Often issues can only be understood in the context of the other flows that share a particular path, common network device, interface port, etc. A simple example is monitoring of a network device that uses a scheduler or active queue management technique [RFC7567], where it could be desirable to understand whether the algorithms are correctly controlling latency, or if overload protection is working. This understanding implies knowledge of how traffic is assigned to any sub-queues used for flow scheduling, but can also require information about how the traffic dynamics impact active queue management, starvation prevention mechanisms, and circuit-breakers.

Sometimes multiple on-path observation points are needed. By correlating observations of headers at multiple points along the path
(e.g., at the ingress and egress of a network segment), an observer can determine the contribution of a portion of the path to an observed metric, to locate a source of delay, jitter, loss, reordering, congestion marking, etc.

3.2.2. Use by Operators to Plan and Provision Networks

Traffic measurements (e.g., traffic volume, loss, latency) is used by operators to help plan deployment of new equipment and configuration in their networks. Data is also valuable to equipment vendors who want to understand traffic trends and patterns of usage as inputs to decisions about planning products and provisioning for new deployments. This measurement information can also be correlated with billing information when this is also collected by an operator.

A network operator supporting traffic that uses transport header encryption might not have access to per-flow measurement data. Trends in aggregate traffic can be observed and can be related to the endpoint addresses being used, but it may be impossible to correlate patterns in measurements with changes in transport protocols (e.g., the impact of changes in introducing a new transport protocol mechanism). This increases the dependency on other indirect sources of information to inform planning and provisioning.

3.2.3. Service Performance Measurement

Traffic measurements (e.g., traffic volume, loss, latency) can be used by various actors to help analyse the performance offered to the users of a network segment, and to inform operational practice.

While active measurements may be used within a network, passive measurements can have advantages in terms of eliminating unproductive test traffic, reducing the influence of test traffic on the overall traffic mix, and the ability to choose the point of observation (see Section 3.2.1). However, passive measurements can rely on observing transport headers which is not possible if those headers are encrypted.

3.2.4. Measuring Transport to Support Network Operations

Information provided by tools observing transport headers can help determine whether mechanisms are needed in the network to prevent flows from acquiring excessive network capacity. Operators can implement operational practices to manage traffic flows (e.g., to prevent flows from acquiring excessive network capacity under severe congestion) by deploying rate-limiters, traffic shaping or network transport circuit breakers [RFC8084].
Congestion Control Compliance of Traffic: Congestion control is a key transport function [RFC2914]. Many network operators implicitly accept that TCP traffic complies with a behaviour that is acceptable for use in the shared Internet. TCP algorithms have been continuously improved over decades and they have reached a level of efficiency and correctness that custom application-layer mechanisms will struggle to easily duplicate [RFC8085].

A standards-compliant TCP stack provides congestion control that may therefore be judged safe for use across the Internet. Applications developed on top of well-designed transports can be expected to appropriately control their network usage, reacting when the network experiences congestion, by back-off and reduce the load placed on the network. This is the normal expected behaviour for IETF-specified transport (e.g., TCP and SCTP).

However, when anomalies are detected, tools can interpret the transport protocol header information to help understand the impact of specific transport protocols (or protocol mechanisms) on the other traffic that shares a network. An observation in the network can gain an understanding of the dynamics of a flow and its congestion control behaviour. Analysing observed flows can help to build confidence that an application flow backs-off its share of the network load in the face of persistent congestion, and hence to understand whether the behaviour is appropriate for sharing limited network capacity. For example, it is common to visualise plots of TCP sequence numbers versus time for a flow to understand how a flow shares available capacity, deduce its dynamics in response to congestion, etc. The ability to identify sources that contribute to persistent congestion is important to safe operation of network infrastructure, and mechanisms can inform configuration of network devices to complement the endpoint congestion avoidance mechanisms [RFC7567] [RFC8084] to avoid a portion of the network being driven into congestion collapse [RFC2914].

Congestion Control Compliance for UDP traffic: UDP provides a minimal message-passing datagram transport that has no inherent congestion control mechanisms. Because congestion control is critical to the stable operation of the Internet, applications and other protocols that choose to use UDP as a transport are required to employ mechanisms to prevent congestion collapse, avoid unacceptable contributions to jitter/latency, and to establish an acceptable share of capacity with concurrent traffic [RFC8085].

A network operator needs tools to understand if datagram flows comply with congestion control expectations and therefore whether there is a need to deploy methods such as rate-limiters, transport
circuit breakers or other methods to enforce acceptable usage for the offered service.

UDP flows that expose a well-known header by specifying the format of header fields can allow information to be observed to gain understanding of the dynamics of a flow and its congestion control behaviour. For example, tools exist to monitor various aspects of the RTP and RTCP header information of real-time flows (see Section 3.1.2, and the Secure RTP extensions [RFC3711] were explicitly designed to expose header information to enable such observation.

3.3. Use for Network Diagnostics and Troubleshooting

Transport header information can be useful for a variety of operational tasks [RFC8404]: to diagnose network problems, assess network provider performance, evaluate equipment/protocol performance, capacity planning, management of security threats (including denial of service), and responding to user performance questions. Sections 3.1.2 and 5 of [RFC8404] provide further examples. These tasks seldom involve the need to determine the contents of the transport payload, or other application details.

A network operator supporting traffic that uses transport header encryption can see only encrypted transport headers. This prevents deployment of performance measurement tools that rely on transport protocol information. Choosing to encrypt all the information reduces the ability of an operator to observe transport performance and could limit the ability of network operators to trace problems, make appropriate QoS decisions, or response to other queries about the network service. For some this will be blessing, for others it may be a curse. For example, operational performance data about encrypted flows needs to be determined by traffic pattern analysis, rather than relying on traditional tools. This can impact the ability of the operator to respond to faults, it could require reliance on endpoint diagnostic tools or user involvement in diagnosing and troubleshooting unusual use cases or non-trivial problems. A key need here is for tools to provide useful information during network anomalies (e.g., significant reordering, high or intermittent loss).

Measurements can be used to monitor the health of a portion of the Internet, to provide early warning of the need to take action. They can assist in 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 (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.

3.4. Header Compression

Header compression saves link bandwidth by compressing network and transport protocol headers on a per-hop basis. It was widely used with low bandwidth dial-up access links, and still finds application on wireless links that are subject to capacity constraints. Header compression has been specified for use with TCP/IP and RTP/UDP/IP flows [RFC2507], [RFC2508], [RFC4995].

While it is possible to compress only the network layer headers, significant bandwidth savings can be made if both the network and transport layer headers are compressed together as a single unit. The Secure RTP extensions [RFC3711] were explicitly designed to leave the transport protocol headers unencrypted, but authenticated, since support for header compression was considered important. Encrypting the transport protocol headers does not break such header compression, but does cause it to fall back to compressing only the
4. Encryption and Authentication of Transport Headers

End-to-end encryption can be applied at various protocol layers. It can be applied above the transport to encrypt the transport payload. Encryption methods can hide information from an eavesdropper in the network. Encryption can also help protect the privacy of a user, by hiding data relating to user/device identity or location. Neither an integrity check nor encryption methods prevent traffic analysis, and usage needs to reflect that profiling of users, identification of location and fingerprinting of behaviour can take place even on encrypted traffic flows. Any header information that has a clear definition in the protocol’s message format(s), or is implied by that definition, and is not cryptographically confidentiality-protected can be unambiguously interpreted by on-path observers [I-D.trammell-wire-image].

There are several motivations:

- One motive to use encryption is a response to perceptions that the network has become ossified by over-reliance on middleboxes that prevent new protocols and mechanisms from being deployed. This has lead to a perception that there is too much "manipulation" of protocol headers within the network, and that designing to deploy in such networks is preventing transport evolution. In the light of this, a method that authenticates transport headers may help improve the pace of transport development, by eliminating the need to always consider deployed middleboxes [I-D.trammell-plus-abstract-mech], or potentially to only explicitly enable middlebox use for particular paths with particular middleboxes that are deliberately deployed to realise a useful function for the network and/or users[RFC3135].

- Another motivation stems from increased concerns about privacy and surveillance. Some Internet users have valued the ability to protect identity, user location, and defend against traffic analysis, and have used methods such as IPsec Encapsulated Security Payload (ESP), Virtual Private Networks (VPNs) and other encrypted tunnel technologies. Revelations about the use of pervasive surveillance [RFC7624] have, to some extent, eroded trust in the service offered by network operators, and following the Snowden revelation in the USA in 2013 has led to an increased desire for people to employ encryption to avoid unwanted "eavesdropping" on their communications. Concerns have also been voiced about the addition of information to packets by third parties to provide analytics, customization, advertising, cross-
site tracking of users, to bill the customer, or to selectively allow or block content. Whatever the reasons, there are now activities in the IETF to design new protocols that could include some form of transport header encryption (e.g., QUIC [I-D.ietf-quic-transport]).

Authentication methods (that provide integrity checks of protocols fields) have also been specified at the network layer, and this also protects transport header fields. The network layer itself carries protocol header fields that are increasingly used to help forwarding decisions reflect the need of transport protocols, such as the IPv6 Flow Label [RFC6437], DSCP, and ECN fields.

The use of transport layer authentication and encryption exposes a tussle between middlebox vendors, operators, applications developers and users.

- On the one hand, future Internet protocols that enable large-scale encryption assist in the restoration of the end-to-end nature of the Internet by returning complex processing to the endpoints, since middleboxes cannot modify what they cannot see.

- On the other hand, encryption of transport layer header information has implications for people who are responsible for operating networks and researchers and analysts seeking to understand the dynamics of protocols and traffic patterns.

Whatever the motives, a decision to use pervasive transport header encryption will have implications on the way in which design and evaluation is performed, and which can in turn impact the direction of evolution of the transport protocol stack. While the IETF can specify protocols, the success in actual deployment is often determined by many factors [RFC5218] that are not always clear at the time when protocols are being defined.

The following briefly reviews some security design options for transport protocols. A Survey of Transport Security Protocols [I-D.ietf-taps-transport-security] provides more details concerning commonly used encryption methods at the transport layer.

Authenticating the Transport Protocol Header: Transport layer header information can be authenticated. An integrity check that protects the immutable transport header fields, but can still expose the transport protocol header information in the clear, allowing in-network devices to observe these fields. An integrity check is not able to prevent in-network modification, but can prevent a receiving from accepting changes and avoid impact on the transport protocol operation.
An example transport authentication mechanism is TCP-Authentication (TCP-AO) [RFC5925]. This TCP option authenticates the IP pseudo header, TCP header, and TCP data. TCP-AO protects the transport layer, preventing attacks from disabling the TCP connection itself and provides replay protection. TCP-AO may interact with middleboxes, depending on their behaviour [RFC3234].

The IPsec Authentication Header (AH) [RFC4302] was designed to work at the network layer and authenticate the IP payload. This approach authenticates all transport headers, and verifies their integrity at the receiver, preventing in-network modification. Secure RTP [RFC3711] is another example of a transport protocol that allows header authentication.

Greasing: Transport layer header information that is observable can be observed in the network. Protocols often provide extensibility features, reserving fields or values for use by future versions of a specification. The specification of receivers has traditionally ignored unspecified values, however in-network devices have emerged that ossify to require a certain value in a field, or reuse a field for another purpose. When the specification is later updated, it is impossible to deploy the new use of the field, and forwarding of the protocol could even become conditional on a specific header field value.

A protocol can intentionally vary the value, format, and/or presence of observable transport header fields. This behaviour, known as GREASE (Generate Random Extensions And Sustain Extensibility), is designed to avoid a network device ossifying the use of a specific observable field. Greasing seeks to ease deployment of new methods. It can also prevent in-network devices utilising the information in a transport header, or can make an observation robust to a set of changing values, rather than a specific set of values.

Encrypting the Transport Payload: The transport layer payload can be encrypted to protect the content of transport segments. This leaves transport protocol header information in the clear. The integrity of immutable transport header fields could be protected by combining this with an integrity check.

Examples of encrypting the payload include Transport Layer Security (TLS) over TCP [RFC8446] [RFC7525], Datagram TLS (DTLS) over UDP [RFC6347] [RFC7525], Secure RTP [RFC3711], and TCPcrypt [I-D.ietf-tcpinc-tcpcrypt] which permits opportunistic encryption of the TCP transport payload.
Encrypting the Transport Headers and Payload: The network layer payload could be encrypted (including the entire transport header and the payload). This method provides confidentiality of the entire transport packet. It therefore does not expose any transport information to devices in the network, which also prevents modification along a network path.

One example of encryption at the network layer is use of IPsec Encapsulating Security Payload (ESP) [RFC4303] in tunnel mode. This encrypts and authenticates all transport headers, preventing visibility of the transport headers by in-network devices. Some Virtual Private Network (VPN) methods also encrypt these headers.

Selectively Encrypting Transport Headers and Payload: A transport protocol design can encrypt selected header fields, while also choosing to authenticate the entire transport header. This allows specific transport header fields to be made observable by network devices. End-to-end integrity checks can prevent an endpoint from undetected modification of the immutable transport headers.

Mutable fields in the transport header provide opportunities for middleboxes to modify the transport behaviour (e.g., the extended headers described in [I-D.trammell-plus-abstract-mech]). This considers only immutable fields in the transport headers, that is, fields that can be authenticated End-to-End across a path.

An example of a method that encrypts some, but not all, transport information is GRE-in-UDP [RFC8086] when used with GRE encryption.

Optional Encryption of Header Information: There are implications to the use of optional header encryption in the design of a transport protocol, where support of optional mechanisms can increase the complexity of the protocol and its implementation and in the management decisions that are required to use variable format fields. Instead, fields of a specific type ought to always be sent with the same level of confidentiality or integrity protection.

As seen, different transports use encryption to protect their header information to varying degrees. There is, however, a trend towards increased protection with newer transport protocols.

5. Addition of Transport Information to Network-Layer Protocol Headers

Some measurements can be made by adding additional protocol headers carrying operations, administration and management (OAM) information to packets at the ingress to a maintenance domain (e.g., an Ethernet protocol header with timestamps and sequence number information using...
a method such as 802.11ag or in-situ OAM [I-D.ietf-ippm-ioam-data])
and removing the additional header at the egress of the maintenance
domain. This approach enables some types of measurements, but does
not cover the entire range of measurements described in this
document. In some cases, it can be difficult to position measurement
tools at the required segments/nodes and there can be challenges in
correlating the downstream/upstream information when in-band OAM data
is inserted by an on-path device. This has the advantage that a
single header can support all transport protocols, but there could
also be less desirable implications of separating the operation of
the transport protocol from the measurement framework.

Another example of a network-layer approach is the IPv6 Performance
and Diagnostic Metrics (PDM) Destination Option [RFC8250]. This
allows a sender to optionally include a destination option that
caries header fields that can be used to observe timestamps and
packet sequence numbers. This information could be authenticated by
receiving transport endpoints when the information is added at the
sender and visible at the receiving endpoint, although methods to do
this have not currently been proposed. This method needs to be
explicitly enabled at the sender.

Current measurements suggest it can be undesirable to rely on methods
requiring the presence of network options or extension headers. IPv4
network options are often not supported (or are carried on a slower
processing path) and some IPv6 networks are also known to drop
packets that set an IPv6 header extension (e.g., [RFC7872]). Another
disadvantage is that protocols that separately expose header
information do not necessarily have an incentive to expose the
information that is utilised by the protocol itself, and could
manipulate the exposed header information to gain an advantage from
the network.

6. Implications of Protecting the Transport Headers

The choice of which fields to expose and which to encrypt is a design
choice for the transport protocol. Any selective encryption method
requires trading two conflicting goals for a transport protocol
designer to decide which header fields to encrypt. Security work
typically employs a design technique that seeks to expose only what
is needed. This approach provides incentives to not reveal any
information that is not necessary for the end-to-end communication.
However, there can be performance and operational benefits in
exposing selected information to network tools.

This section explores key implications of working with encrypted
transport protocols.
6.1. Independent Measurement

Independent observation by multiple actors is important for scientific analysis. Encrypting transport header encryption changes the ability for other actors to collect and independently analyse data. Internet transport protocols employ a set of mechanisms. Some of these need to work in cooperation with the network layer - loss detection and recovery, congestion detection and congestion control, some of these need to work only end-to-end (e.g., parameter negotiation, flow-control).

The majority of present Internet applications use two well-known transport protocols, TCP and UDP. Although TCP represents the majority of current traffic, some real-time applications use UDP, and much of this traffic utilises RTP format headers in the payload of the UDP datagram. Since these protocol headers have been fixed for decades, a range of tools and analysis methods have become common and well-understood.

Protocols that expose the state information used by the transport protocol in their header information (e.g., timestamps used to calculate the RTT, packet numbers used to assess congestion and requests for retransmission) provide an incentive for the sending endpoint to provide correct information, increasing confidence that the observer understands the transport interaction with the network. For example, when TCP is used over an unencrypted network path (i.e., one that does not use IPsec or other encryption below the transport), it implicitly exposes header information that can be used for measurement at any point along the path. This information is necessary for the protocol’s correct operation, therefore there is no incentive for a TCP implementation to put incorrect information in this transport header. A network device can have confidence that the well-known (and ossified) transport information represents the actual state of the endpoints.

When encryption is used to conceal some or all of the transport headers, the transport protocol choose what information to reveal to the network about its internal state, what information to leave encrypted, and what fields to grease to protect against future ossification. Such a transport could be designed, for example, to provide summary data regarding its performance, congestion control state, etc., or to make an explicit measurement signal available. For example, a QUIC endpoint could set the spin bit to reflect to explicitly reveal a session’s RTT ([I-D.ietf-quic-spin-exp]).

When providing or using such information, it becomes important to consider the privacy of the user and their incentive for providing accurate and detailed information. Protocols that selectively reveal
some transport state or measurement signals are choosing to establish a trust relationship with the network operators. There is no protocol mechanism that can guarantee that the information provided represents the actual transport state of the endpoints, since those endpoints can always send additional information in the encrypted part of the header, to update to replace whatever they reveal. This reduces the ability to independently measure and verify that a protocol is behaving as expected. Some operational uses need the information to contain sufficient detail to understand, and possibly reconstruct, the network traffic pattern for further testing; such operators must gain the trust of transport protocol implementers if they are to correctly reveal such information.

OAM data records [I-D.ietf-ippm-ioam-data] could be embedded into a variety of encapsulation methods at different layers to support the goals of a specific operational domain. OAM-related metadata can support functions such as performance evaluation, path-tracing, path verification information, classification and a diversity of other uses. When encryption is used to conceal some or all of the transport headers, analysis will require coordination between actors at different layers to successfully characterise flows and correlate the performance or behavior of a specific mechanism with the configuration and traffic using operational equipment (e.g. Combining transport and network measurements to explore congestion control dynamics or the implications of active queue management).

For some usage a standardised endpoint-based logging format (e.g., based onQuic-Trace [Quic-Trace]) could offer an alternative to in-network measurement. Such information will have a diversity of uses - examples include developers wishing to debug/understand the transport/applications protocols with which they work, to researchers seeking to spot trends, anomalies and to characterise variants of protocols. This use will need to establish the validity and provenance of the logging information (e.g., to establish how and when traces were captured).

However, endpoint logs do not provide equivalent information to in-network measurements. In particular, endpoint logs contain only a part of the information needed to understand the operation of network devices and identify issues such as link performance or capacity sharing between multiple flows. Additional information is needed to determine which equipment/links are used and the configuration of equipment along the network paths being measured.
6.2. Characterising "Unknown" Network Traffic

The patterns and types of traffic that share Internet capacity change over time as networked applications, usage patterns and protocols continue to evolve.

If "unknown" or "uncharacterised" traffic patterns form a small part of the traffic aggregate passing through a network device or segment of the network, the dynamics of the uncharacterised traffic may not have a significant collateral impact on the performance of other traffic that shares this network segment. Once the proportion of this traffic increases, the need to monitor the traffic and determine if appropriate safety measures need to be put in place.

Tracking the impact of new mechanisms and protocols requires traffic volume to be measured and new transport behaviours to be identified. This is especially true of protocols operating over a UDP substrate. The level and style of encryption needs to be considered in determining how this activity is performed. On a shorter timescale, information may also need to be collected to manage denial of service attacks against the infrastructure.

6.3. Accountability and Internet Transport Protocols

Information provided by tools observing transport headers can be used to classify traffic, and to limit the network capacity used by certain flows, as discussed in Section 3.2.4). Equally, operators could use analysis of transport headers and transport flow state to demonstrate that they are not providing differential treatment to certain flows. Obfuscating or hiding this information using encryption may lead operators and maintainers of middleboxes (firewalls, etc.) to seek other methods to classify, and potentially other mechanisms to condition, network traffic.

A lack of data that reduces the level of precision with which flows can be classified also reduces the design space for conditioning mechanisms (e.g., rate limiting, circuit breaker techniques [RFC8084], or blocking of uncharacterised traffic), and this needs to be considered when evaluating the impact of designs for transport encryption [RFC5218].

6.4. Impact on Operational Cost

Many network operators currently utilise observed transport information as a part of their operational practice, and have developed tools and operational practices based around currently deployed transports and their applications. Encryption of the transport information prevents tools from directly observing this
information. A variety of open source and commercial tools have been deployed that utilise this information for a variety of short and long term measurements.

The network will not break just because transport headers are encrypted, although alternative diagnostic and troubleshooting tools would need to be developed and deployed. Introducing a new protocol or application can require these tool chains and practice to be updated, and may in turn impact operational mechanisms, and policies. Each change can introduce associated costs, including the cost of collecting data, and the tooling needed to handle multiple formats (possibly as these co-exist in the network, when measurements need to span time periods during which changes are deployed, or to compare with historical data). These costs are incurred by an operator to manage the service and debug network issues.

At the time of writing, the additional operational cost of using encrypted transports is not yet well understood. Design trade-offs could mitigate these costs by explicitly choosing to expose selected information (e.g., header invariants and the spin-bit in QUIC[I-D.ietf-quic-transport]), the specification of common log formats and development of alternative approaches.

6.5. Impact on Research, Development and Deployment

Measurement has a critical role in the design of transport protocol mechanisms and their acceptance by the wider community (e.g., as a method to judge the safety for Internet deployment) and is increasingly being used to inform design decisions in networking research, during development of new mechanisms and protocols and in standardisation. Observation of pathologies are also important in 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.

Evolution and the ability to understand (measure) the impact need to proceed hand-in-hand. Attention needs to be paid to the expected scale of deployment of new protocols and protocol mechanisms. Whatever the mechanism, experience has shown that it is often difficult to correctly implement combination of mechanisms [RFC8085]. These mechanisms therefore typically evolve as a protocol matures, or in response to changes in network conditions, changes in network traffic or changes to application usage.

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.

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.

7. Conclusions

Confidentiality and strong integrity checks have properties that are being incorporated into new protocols and that have important benefits. The pace of development of transports using the WebRTC data channel and the rapid deployment of the QUIC transport protocol can both be attributed to using the combination of UDP as a substrate while providing confidentiality and authentication of the encapsulated transport headers and payload.

The traffic that can be observed by on-path network devices is a function of transport protocol design/options, network use, applications, and user characteristics. In general, when only a small proportion of the traffic has a specific (different) characteristic, such traffic seldom leads to operational concern, although the ability to measure and monitor it is less. The desire to understand the traffic and protocol interactions typically grows as the proportion of traffic increases in volume. The challenges increase when multiple instances of an evolving protocol contribute to the traffic that share network capacity.

An increased pace of evolution therefore needs to be accompanied by methods that can be successfully deployed and used across operational networks. This leads to a need for network operators (at various level (ISPs, enterprises, firewall maintainer, etc) to identify appropriate operational support functions and procedures.

Protocols that change their transport header format (wire format) or their behaviour (e.g., algorithms that are needed to classify and characterise the protocol), will require new tooling to be developed to catch-up with the change. If the currently deployed tools and
methods are no longer relevant then it may no longer be possible to correctly measure performance. This can increase the response-time after faults, and can impact the ability to manage the network resulting in traffic causing traffic to be treated inappropriately (e.g., rate limiting because of being incorrectly classified/monitored).

There are benefits in exposing consistent information to the network that avoids traffic being inappropriately classified and then receiving a default treatment by the network. The flow label and DSCP fields provide examples of how transport information can be made available for network-layer decisions. Extension headers could also be used to carry transport information that can inform network-layer decisions.

As part of a protocol’s design, the community 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 have appropriate tools to manage their networks and enable stable operation of the Internet as new protocols are deployed.

8. Security Considerations

This document is about design and deployment considerations for transport protocols. Issues relating to security are discussed in the various sections of the document.

Authentication, confidentiality protection, and integrity protection are identified as Transport Features by [RFC8095]. As currently deployed in the Internet, these features are generally provided by a protocol or layer on top of the transport protocol [I-D.ietf-taps-transport-security].

Confidentiality and strong integrity checks have properties that can also be incorporated into the design of a transport protocol. Integrity checks can protect an endpoint from undetected modification of protocol fields by network devices, whereas encryption and obfuscation or greasing can further prevent these headers being utilised by network devices. Hiding headers can therefore provide the opportunity for greater freedom to update the protocols and can ease experimentation with new techniques and their final deployment in endpoints. A protocol specification needs to weigh the benefits of ossifying common headers, versus the potential demerits of exposing specific information that could be observed along the network path to provide tools to manage new variants of protocols.
A protocol design that uses header encryption can provide confidentiality of some or all of the protocol header information. This prevents an on-path device from knowledge of the header field. It therefore prevents mechanisms being built that directly rely on the information or 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 can not be predicted (see section 5.1 of [RFC8085]). A receiver could also require additional information to be used as a part of a validation check before accepting packets at the transport layer (e.g., utilising a part of the sequence number space that is encrypted; or by verifying an encrypted token not visible to an attacker). This would also mitigate on-path attacks. An additional processing cost can be incurred when decryption needs to be attempted before a receiver is able to discard injected packets.
Open standards motivate a desire for this evaluation to include independent observation and evaluation of performance data, which in turn suggests control over where and when measurement samples are collected. This requires consideration of the appropriate balance between encrypting all and no transport information. Open data, and accessibility to tools that can help understand trends in application deployment, network traffic and usage patterns can all contribute to understanding security challenges.

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, and other members of the TSVWG for their comments and feedback.

This work has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 688421. The opinions expressed and arguments employed reflect only the authors’ view. The European Commission is not responsible for any use that may be made of that information.

This work has received funding from the UK Engineering and Physical Sciences Research Council under grant EP/R04144X/1.

11. Informative References


[I-D.ietf-quic-spin-exp]

[I-D.ietf-quic-transport]

[I-D.ietf-rtcweb-overview]

[I-D.ietf-taps-transport-security]

[I-D.ietf-tcpinc-tcpcrypt]

[I-D.ietf-tsvwg-rtcweb-qos]

[I-D.thomson-quic-grease]

[I-D.trammell-plus-abstract-mech]

[I-D.trammell-wire-image]


"https://QUIC trace utilities //github.com/google/quic-trace".


Fairhurst & Perkins     Expires December 2, 2019     [Page 37]


Appendix A. Revision information

-00 This is an individual draft for the IETF community.

-01 This draft was a result of walking away from the text for a few days and then reorganising the content.

-02 This draft fixes textual errors.

-03 This draft follows feedback from people reading this draft.

-04 This draft adds an additional contributor and includes significant reworking to ready this for review by the wider IETF community Colin Perkins joined the author list.

Comments from the community are welcome on the text and recommendations.

-05 Corrections received and helpful inputs from Mohamed Boucadair.

-06 Updated following comments from Stephen Farrell, and feedback via email. Added a draft conclusion section to sketch some strawman scenarios that could emerge.

-07 Updated following comments from Al Morton, Chris Seal, and other feedback via email.

-08 Updated to address comments sent to the TSVWG mailing list by Kathleen Moriarty (on 08/05/2018 and 17/05/2018), Joe Touch on 11/05/2018, and Spencer Dawkins.

-09 Updated security considerations.

-10 Updated references, split the Introduction, and added a paragraph giving some examples of why ossification has been an issue.

-01 This resolved some reference issues. Updated section on observation by devices on the path.

-02 Comments received from Kyle Rose, Spencer Dawkins and Tom Herbert. The network-layer information has also been re-organised after comments at IETF-103.

-03 Added a section on header compression and rewriting of sections referring to RTP transport. This version contains author editorial work and removed duplicate section.

-04 Revised following SecDir Review
o Added some text on TLS story (additional input sought on relevant considerations).

o Section 2, paragraph 8 - changed to be clearer, in particular, added "Encryption with secure key distribution prevents"

o Flow label description rewritten based on PS/BCP RFCs.

o Clarify requirements from RFCs concerning the IPv6 flow label and highlight ways it can be used with encryption. (section 3.1.3)

o Add text on the explicit spin-bit work in the QUIC DT. Added greasing of spin-bit. (Section 6.1)

o Updated section 6 and added more explanation of impact on operators.

o Other comments addressed.

-05 Editorial pass and minor corrections noted on TSVWG list.

-06 Updated conclusions and minor corrections. Responded to request to add OAM discussion to Section 6.1.

Authors’ Addresses

Godred Fairhurst
University of Aberdeen
Department of Engineering
Fraser Noble Building
Aberdeen AB24 3UE
Scotland

EMail: gorry@erg.abdn.ac.uk
URI: http://www.erg.abdn.ac.uk/

Colin Perkins
University of Glasgow
School of Computing Science
Glasgow G12 8QQ
Scotland

EMail: csp@csperkins.org
URI: https://csperkins.org//
Transport Options for UDP
draft-ietf-tsvwg-udp-options-07.txt

Status of this Memo

This Internet-Draft is submitted in full conformance with the
provisions of BCP 78 and BCP 79. This document may not be modified,
and derivative works of it may not be created, except to format it
for publication as an RFC or to translate it into languages other
than English.

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

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

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

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

This Internet-Draft will expire on September 8, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal
Provisions Relating to IETF Documents
(http://trustee.ietf.org/license-info) in effect on the date of
publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with
Transport protocols are extended through the use of transport header options. This document experimentally extends UDP by indicating the location, syntax, and semantics for UDP transport layer options.

Table of Contents

1. Introduction...................................................3
2. Conventions used in this document..............................3
3. Background.....................................................3
4. The UDP Option Area............................................4
5. UDP Options....................................................7
  5.1. End of Options List (EOL)................................8
  5.2. No Operation (NOP)........................................9
  5.3. Option Checksum (OCS)...................................9
  5.4. Alternate Checksum (ACS).................................11
  5.5. Lite (LITE)...............................................11
  5.6. Maximum Segment Size (MSS).............................13
  5.7. Fragmentation (FRAG)....................................14
  5.8. Coupling FRAG with LITE................................16
  5.9. Timestamps (TIME).......................................17
  5.10. Authentication and Encryption (AE).....................18
6. Echo request (REQ) and echo response (RES)....................19
  6.1. Experimental (EXP).....................................19
7. Rules for designing new options................................20
8. Option inclusion and processing...............................21
9. UDP API Extensions............................................23
10. Whose options are these?....................................23
11. UDP options LITE option vs. UDP-Lite.......................24
12. Interactions with Legacy Devices............................25
13. Options in a Stateless, Unreliable Transport Protocol........25
14. UDP Option State Caching....................................26
15. Updates to RFC 768...........................................26
16. Multicast Considerations....................................26
17. Security Considerations.....................................27
18. IANA Considerations.........................................27
19. References...................................................28
  19.1. Normative References..................................28
  19.2. Informative References................................28
20. Acknowledgments..............................................30
Appendix A. Implementation Information.............................31

1. Introduction

Transport protocols use options as a way to extend their capabilities. TCP [RFC793], SCTP [RFC4960], and DCCP [RFC4340] include space for these options but UDP [RFC768] currently does not. This document defines an experimental extension to UDP that provides space for transport options including their generic syntax and semantics for their use in UDP’s stateless, unreliable message protocol.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lowercase uses of these words are not to be interpreted as carrying significance described in RFC 2119.

In this document, the characters ">>" preceding an indented line(s) indicates a statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the portions of this RFC covered by these key words.

3. Background

Many protocols include a default header and an area for header options. These options enable the protocol to be extended for use in particular environments or in ways unforeseen by the original designers. Examples include TCP’s Maximum Segment Size, Window Scale, Timestamp, and Authentication Options [RFC793][RFC5925][RFC7323].

These options are used both in stateful (connection-oriented, e.g., TCP [RFC793], SCTP [RFC4960], DCCP [RFC4340]) and stateless (connectionless, e.g., IPv4 [RFC791], IPv6 [RFC8200]) protocols. In stateful protocols they can help extend the way in which state is managed. In stateless protocols their effect is often limited to individual packets, but they can have an aggregate effect on a sequence as well. One example of such uses is Substrate Protocol for User Datagrams (SPUD) [Tr16], and this document is intended to provide an out-of-band option area as an alternative to the in-band mechanism currently proposed [Hi15].
UDP is one of the most popular protocols that lacks space for options [RFC768]. The UDP header was intended to be a minimal addition to IP, providing only ports and a data checksum for protection. This document experimentally extends UDP to provide a trailer area for options located after the UDP data payload.

4. The UDP Option Area

The UDP transport header includes demultiplexing and service identification (port numbers), a checksum, and a field that indicates the UDP datagram length (including UDP header). The UDP Length field is typically redundant with the size of the maximum space available as a transport protocol payload (see also discussion in Section 12).

For IPv4, IP Total Length field indicates the total IP datagram length (including IP header), and the size of the IP options is indicated in the IP header (in 4-byte words) as the "Internet Header Length" (IHL), as shown in Figure 1 [RFC791]. As a result, the typical (and largest valid) value for UDP Length is:

\[
\text{UDP\_Length} = \text{IPv4\_Total\_Length} - \text{IPv4\_IHL} \times 4
\]

For IPv6, the IP Payload Length field indicates the datagram after the base IPv6 header, which includes the IPv6 extension headers and space available for the transport protocol, as shown in Figure 2 [RFC8200]. Note that the Next HDR field in IPv6 might not indicate UDP (i.e., 17), e.g., when intervening IP extension headers are present. For IPv6, the lengths of any additional IP extensions are indicated within each extension [RFC8200], so the typical (and largest valid) value for UDP Length is:

\[
\text{UDP\_Length} = \text{IPv6\_Payload\_Length} - \text{sum(extension header lengths)}
\]

In both cases, the space available for the UDP transport protocol data unit is indicated by IP, either completely in the base header (for IPv4) or adding information in the extensions (for IPv6). In either case, this document will refer to this available space as the "IP transport payload".
As a result of this redundancy, there is an opportunity to use the UDP Length field as a way to break up the IP transport payload into two areas - that intended as UDP user data and an additional "surplus area" (as shown in Figure 3).
In most cases, the IP transport payload and UDP Length point to the same location, indicating that there is no surplus area. It is important to note that this is not a requirement of UDP [RFC768] (discussed further in Section 12). UDP-Lite used the difference in these pointers to indicate the partial coverage of the UDP Checksum, such that the UDP user data, UDP header, and UDP pseudoheader (a subset of the IP header) are covered by the UDP checksum but additional user data in the surplus area is not covered [RFC3828]. This document uses the surplus area for UDP transport options.

The UDP option area is thus defined as the location between the end of the UDP payload and the end of the IP datagram as a trailing options area. This area can occur at any valid byte offset, i.e., it need not be 16-bit or 32-bit aligned. In effect, this document redefines the UDP "Length" field as a "trailer offset".

UDP options are defined using a TLV (type, length, and optional value) syntax similar to that of TCP [RFC793]. They are typically a minimum of two bytes in length as shown in Figure 4, excepting only the one byte options "No Operation" (NOP) and "End of Options List" (EOL) described below.

> Example of UDP option default format

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
</tr>
</thead>
</table>

>> UDP options MAY occur at any UDP length offset.

>> The UDP length MUST be at least as large as the UDP header (8) and no larger than the IP transport payload. Values outside this range MUST be silently discarded as invalid and logged where rate-limiting permits.

Others have considered using values of the UDP Length that is larger than the IP transport payload as an additional type of signal. Using
a value smaller than the IP transport payload is expected to be
backward compatible with existing UDP implementations, i.e., to
deliver the UDP Length of user data to the application and silently
ignore the additional surplus area data. Using a value larger than
the IP transport payload would either be considered malformed (and
be silently dropped) or could cause buffer overruns, and so is not
considered silently and safely backward compatible. Its use is thus
out of scope for the extension described in this document.

UDP options MUST be interpreted in the order in which they occur
in the UDP option area.

5. UDP Options

The following UDP options are currently defined:

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0*</td>
<td>-</td>
<td>End of Options List (EOL)</td>
</tr>
<tr>
<td>1*</td>
<td>-</td>
<td>No operation (NOP)</td>
</tr>
<tr>
<td>2*</td>
<td>2</td>
<td>Option checksum (OCS)</td>
</tr>
<tr>
<td>3*</td>
<td>4</td>
<td>Alternate checksum (ACS)</td>
</tr>
<tr>
<td>4*</td>
<td>4</td>
<td>Lite (LITE)</td>
</tr>
<tr>
<td>5*</td>
<td>4</td>
<td>Maximum segment size (MSS)</td>
</tr>
<tr>
<td>6*</td>
<td>8/10</td>
<td>Fragmentation (FRAG)</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>Timestamps (TIME)</td>
</tr>
<tr>
<td>8</td>
<td>(varies)</td>
<td>Authentication and Encryption (AE)</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>Request (REQ)</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>Response (RES)</td>
</tr>
<tr>
<td>11-126</td>
<td>(varies)</td>
<td>UNASSIGNED (assignable by IANA)</td>
</tr>
<tr>
<td>127-253</td>
<td></td>
<td>RESERVED</td>
</tr>
<tr>
<td>254</td>
<td>N(&gt;=4)</td>
<td>RFC 3692-style experiments (EXP)</td>
</tr>
<tr>
<td>255</td>
<td></td>
<td>RESERVED</td>
</tr>
</tbody>
</table>

These options are defined in the following subsections. Options 0
and 1 use the same values as for TCP.

An endpoint supporting UDP options MUST support those marked with
a "**" 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).

Required options MUST come before other options. Each required option MUST NOT occur more than once (if they are repeated in a received segment, all except the first MUST be silently ignored).

The requirement that required options come before others is intended to allow for endpoints to implement DOS protection, as discussed further in Section 17.

5.1. End of Options List (EOL)

The End of Options List (EOL) option indicates that there are no more options. It is used to indicate the end of the list of options without needing to pad the options to fill all available option space.

```
+--------+
| Kind=0  |
+--------+
```

Figure 5 UDP EOL option format

When the UDP options do not consume the entire option area, the last non-NOP option SHOULD be EOL (vs. filling the entire option area with NOP values).
>> All bytes after EOL MUST be ignored by UDP option processing. As a result, there can only ever be one EOL option (even if other bytes were zero, they are ignored).

5.2. No Operation (NOP)

The No Operation (NOP) option is a one byte placeholder, intended to be used as padding, e.g., to align multi-byte options along 16-bit or 32-bit boundaries.

```
+--------+
| Kind=1 |
+--------+
```

Figure 6 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) is conventional Internet checksum that covers all of the UDP options. The primary purpose of OCS is to detect non-standard (i.e., non-option) uses of the options area.

OCS is calculated by computing the Internet checksum options area [RFC1071]. OCS protects the option area from errors in a similar way that the UDP checksum protects the UDP user data (when not zero).

```
+--------+--------+
| Kind=2 |checksum|
+--------+--------+
```

Figure 7 UDP OCS option format
When present, the option checksum SHOULD occur as early as possible, preceded by only NOP options for alignment and the LITE option if present.

The OCS checksum MUST be half-word coordinated with the start of the UDP options area.

This coordination is accomplished by computing the Internet checksum over the UDP options area (including EOL, if present) and then adjusting the result before storing it into the OCS checksum field. If that field is aligned to the start of the options area, then the checksum is inserted as-is, otherwise the checksum bytes are swapped before inserting them into the field.

The adjustment above helps enable that OCS, together with the other options, result in an overall zero ones-complement sum. This feature is intended to potentially help the UDP options traverse devices that incorrectly attempt to checksum the surplus area (as originally proposed as the Checksum Compensation Option, i.e., CCO [Fa18]).

Note that this incorrect checksum traversal feature is defeated by the use of LITE, whether alone or with FRAG, because the LITE area is deliberately not covered by OCS. It also is defeated by the use of a zero UDP checksum (i.e., UDP checksum disabled).

OCS covers the UDP option area, including the Lite option (but not LITE data area) as formatted before swapping (or relocation) for transmission (or, equivalently, after the swap/relocation after reception), as the LITE option would occur at the beginning of the original (prior to rearrangement for transmission) or restored (after rearrangement upon reception) UDP option area.

If OCS fails, all options MUST be ignored and any trailing surplus data (and Lite data, if used) silently discarded.

UDP data that is validated by a correct UDP checksum MUST be delivered to the application layer, even if OCS fails, unless the endpoints have negotiated otherwise for this segment’s socket pair.

As a reminder, use of the UDP checksum is optional. When not used, i.e., when the field is zero, that checksum is assumed to be "correct" for the purpose of accepting UDP packets at a receiver.

OCS is intended to check for accidental errors, not for attacks.
5.4. Alternate Checksum (ACS)

The Alternate Checksum (ACS) provides a stronger alternative to the checksum in the UDP header, using a 16-bit CRC of the conventional UDP payload only (excluding the IP pseudoheader, UDP header, and UDP options, and not include the LITE area). Because it does not include the IP pseudoheader or UDP header, it need not be updated by NATs when IP addresses or UDP ports are rewritten. Its purpose is to detect errors that the UDP checksum, when used, might not detect.

CRC-CCITT (polynomial $x^{16} + x^{12} + x^{5} + 1$) has been chosen because of its ubiquity and use in other packet protocols, such as X.25, HDLC, and Bluetooth. The option contains FCS-16 as defined in Appendix C of [RFC1662], except that it is not inverted in the final step and that it is stored in the ACS option in network byte order.

```
+--------+--------+--------+--------+
| Kind=3 | Len=4  |     CRC16sum    |
+--------+--------+--------+--------+
```

Figure 8 UDP ACS option format

When present, the ACS always contains a valid CRC checksum. There are no reserved values, including the value of zero. If the CRC is zero, this must indicate a valid checksum (i.e., it does not indicate that the ACS is not used; instead, the option would simply not be included if that were the desired effect).

ACS does not protect the UDP pseudoheader; only the current UDP checksum provides that protection (when used). ACS cannot provide that protection because it would need to be updated whenever the UDP pseudoheader changed, e.g., during NAT address and port translation; because this is not the case, ACS does not cover the pseudoheader.

5.5. Lite (LITE)

The Lite option (LITE) is intended to provide equivalent capability to the UDP Lite transport protocol [RFC3828]. UDP Lite allows the UDP checksum to cover only a prefix of the UDP data payload, to protect critical information (e.g., application headers) but allow potentially erroneous data to be passed to the user. This feature helps protect application headers but allows for application data errors. Some applications are impacted more by a lack of data than errors in data, e.g., voice and video.

>> When LITE is active, it MUST come first in the UDP options list.
LITE is intended to support the same API as for UDP Lite to allow applications to send and receive data that has a marker indicating the portion protected by the UDP checksum and the portion not protected by the UDP checksum.

LITE includes a 2-byte offset that indicates the length of the portion of the UDP data that is not covered by the UDP checksum.

```
+--------+--------+--------+--------+
| Kind=4 | Len=4  |      Offset     |
+--------+--------+--------+--------+
```

Figure 9 UDP LITE option format

At the sender, the option is formed using the following steps:

1. Create a LITE option, ordered as the first UDP option (Figure 10).

2. Calculate the location of the start of the options as an absolute offset from the start of the UDP header and place that length in the last two bytes of the LITE option.

3. If the LITE data area is 4 bytes or longer, swap all four bytes of the LITE option with the first 4 bytes of the LITE data area (Figure 11). If the LITE data area is 0-3 bytes long, slide the LITE option to the front of the LITE data area (i.e., placing the 0-3 bytes of LITE data after the LITE option).

```
+---------+--------------+--------------+------------------+
| UDP Hdr |  user data   |  LITE data   |LITE| other opts  |
+---------+--------------+--------------+------------------+
<------------------>
```

---

Figure 10 LITE option formation - LITE goes first

```
+---------+--------------+--------------+------------------+
| UDP Hdr |  user data   |  LITE data   |LITE| other opts  |
+---------+--------------+--------------+------------------+
        |              |              |                 |
```

---

Figure 11 Before sending swap LITE option and front of LITE data
The resulting packet has the format shown in Figure 12. Note that the UDP length now points to the LITE option, and the LITE option points to the start of the option area.

```
+---------+--------------+----------------+------------------+
| UDP Hdr |  user data   |LITE| LITE data |Ldat| other opts |
+---------+--------------+----------------+------------------+
<---------------------->    |             ^
   UDP Length             +-----------+
```

Figure 12   Lite option as sent

A legacy endpoint receiving this packet will discard the LITE option and everything that follows, including the lite data and remainder of the UDP options. The UDP checksum will protect only the user data, not the LITE option or lite data.

Receiving endpoints capable of processing UDP options will do the following:

1. Process options as usual. This will start at the LITE option.

2. When the LITE option is encountered, record its location as the start of the LITE data area and (if the LITE offset indicates a LITE data length of at least 4 bytes) swap the four bytes there with the four bytes at the location indicated inside the LITE option, which indicates the start of all of the options, including the LITE one (one past the end of the lite data area). If the LITE offset indicates a LITE data area of 0-3 bytes, then slide the LITE option forward that amount and slide the corresponding bytes after the LITE option to where the LITE option originally began. In either case, this restores the format of the option as it was prior to being sent, as per Figure 10.

3. Continue processing the remainder of the options, which are now in the format shown in Figure 11.

The purpose of this swap (or slide) is to support the equivalent of UDP Lite operation together with other UDP options without requiring the entire LITE data area to be moved after the UDP option area.

5.6. Maximum Segment Size (MSS)

The Maximum Segment Size (MSS, Kind = 3) is a 16-bit indicator of the largest UDP segment that can be received. As with the TCP MSS option [RFC793], the size indicated is the IP layer MTU decreased by
the fixed IP and UDP headers only [RFC6691]. The space needed for IP and UDP options need to be adjusted by the sender when using the value indicated. The value transmitted is based on EMTU_R, the largest IP datagram that can be received (i.e., reassembled at the receiver) [RFC1122].

```
+--------+--------+--------+--------+
| Kind=5 | Len=4  | MSS size |
+--------+--------+--------+--------+
```

Figure 13  UDP MSS option format

The UDP MSS option MAY be used for path MTU discovery [RFC1191][RFC8201], but this may be difficult because of known issues with ICMP blocking [RFC2923] as well as UDP lacking automatic retransmission. It is more likely to be useful when coupled with IP source fragmentation to limit the largest reassembled UDP message, e.g., when EMTU_R is larger than the required minimums (576 for IPv4 [RFC791] and 1500 for IPv6 [RFC8200]).

5.7. Fragmentation (FRAG)

The Fragmentation option (FRAG) supports UDP fragmentation and 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 14  UDP non-terminal FRAG option format

The FRAG option also lacks a "more" bit, zeroed for the terminal fragment of a set. This is possible because the terminal FRAG option is indicated as a longer, 10-byte variant, which includes an Internet checksum over the entire reassembled UDP payload (omitting the IP pseudoheader and UDP header, as well as UDP options), as shown in Figure 15.
The reassembly checksum SHOULD be used, but MAY be unused in the same situations when the UDP checksum is unused (e.g., for transit tunnels or applications that have their own integrity checks [RFC8200]), and by the same mechanism (set the field to 0x0000).

+--------+--------+--------+--------+
| Kind=6 | Len=10 | Frag. Offset |
+--------+--------+--------+--------+
| Identification |
+----------------+
| Checksum |
+----------+

Figure 15  UDP terminal FRAG option format

During fragmentation, the UDP header checksum of each fragment needs to be recomputed based on each datagram’s pseudoheader.

After reassembly is complete and validated using the checksum of the terminal FRAG option, the UDP header checksum of the resulting datagram needs to be recomputed based on the datagram’s pseudoheader.

The Fragment Offset is 16 bits and indicates the location of the UDP payload fragment in bytes from the beginning of the original unfragmented payload. The Len field indicates whether there are more fragments (Len=8) or no more fragments (Len=12).

The Identification field is a 32-bit value that MUST be unique over the expected fragment reassembly timeout.

The Identification field SHOULD be generated in a manner similar to that of the IPv6 Fragment ID [RFC8200].

UDP fragments MUST NOT overlap.

FRAG needs to be used with extreme care because it will present incorrect datagram boundaries to a legacy receiver, unless encoded as LITE data (see Section 5.8).

A host SHOULD indicate FRAG support by transmitting an unfragmented datagram using the Fragmentation option (e.g., with Offset zero and length 12, i.e., including the checksum area), except when encoded as LITE.
A host MUST NOT transmit a UDP fragment before receiving recent confirmation from the remote host, except when FRAG is encoded as LITE.

UDP fragmentation relies on a fragment expiration timer, which can be preset or could use a value computed using the UDP Timestamp option.

The default UDP reassembly SHOULD be no more than 2 minutes. Implementers are advised to limit the space available for UDP reassembly.

UDP reassembly space SHOULD be limited to reduce the impact of DOS attacks on resource use.

UDP reassembly space limits SHOULD NOT be implemented as an aggregate, to avoid cross-socketpair DOS attacks.

Individual UDP fragments MUST NOT be forwarded to the user. The reassembled datagram is received only after complete reassembly, checksum validation, and continued processing of the remaining options. Any additional UDP options would follow the FRAG option in the final fragment, and would be included in the reassembled packet. Processing of those options would commence after reassembly.

UDP options MUST NOT follow the FRAG header in non-terminal fragments. Any data following the FRAG header in non-terminal fragments MUST be silently dropped. All other options that apply to a reassembled packet MUST follow the FRAG header in the terminal fragment.

5.8. Coupling FRAG with LITE

FRAG can be coupled with LITE to avoid impacting legacy receivers. Each fragment is sent as LITE un-checksummed data, where each UDP packet contains no legacy-compatible data. Legacy receivers interpret these as zero-length payload packets (i.e., UDP Length field is 8, the length of just the UDP header), which would not affect the receiver unless the presence of the packet itself were a signal. The header of such a packet would appear as shown in Figure 16 and Figure 17.
When a packet is reassembled, it appears as a complete LITE data region. The UDP header of the reassembled packet is adjusted accordingly, so that the reassembled region now appears as conventional UDP user data, and processing of the UDP options continues, as with the non-LITE FRAG variant.

5.9. Timestamps (TIME)

The UDP Timestamp option (TIME) exchanges two four-byte timestamp fields. It serves a similar purpose to TCP’s TS option [RFC7323], enabling UDP to estimate the round trip time (RTT) between hosts. For UDP, this RTT can be useful for establishing UDP fragment reassembly timeouts or transport-layer rate-limiting [RFC8085].

```
+--------+--------+------------------+------------------+
| Kind=7 | Len=10 |      TSval       |      TSecr       |
+--------+--------+------------------+------------------+
    1 byte  1 byte       4 bytes            4 bytes
```

Figure 18  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 option (AE) is intended to allow UDP to provide a similar type of authentication as the TCP Authentication Option (TCP-AO) [RFC5925]. It uses the same format as specified for TCP-AO, except that it uses a Kind of 8. AE supports NAT traversal in a similar manner as TCP-AO [RFC6978]. AE can also be extended to provide a similar encryption capability as TCP-AO-ENC, in a similar manner [To18ao].

```
+--------+--------+--------+--------+
| Kind=8 |  Len   |     Digest...   |
|          Digest (con’t)...        |
|--------+--------+--------+--------+
```

Figure 19   UDP AE option format

Like TCP-AO, AE is not negotiated in-band. Its use assumes both endpoints have populated Master Key Tuples (MKTs), used to exclude non-protected traffic.

TCP-AO generates unique traffic keys from a hash of TCP connection parameters. UDP lacks a three-way handshake to coordinate connection-specific values, such as TCP’s Initial Sequence Numbers (ISNs) [RFC793], thus AE’s Key Derivation Function (KDF) uses zeroes.
as the value for both ISNs. This means that the AE reuses keys when
socket pairs are reused, unlike TCP-AO.

AE can be configured to either include or exclude UDP options, the
same way as can TCP-AO. When UDP options are covered, the OCS option
area checksum and AE hash areas are zeroed before computing the AE
hash. It is important to consider that options not yet defined might
yield unpredictable results if not confirmed as supported, e.g., if
they were to contain other hashes or checksums that depend on the
option area contents. This is why such dependencies are not
permitted except as defined for OCS and UDP-AE.

Similar to TCP-AO-NAT, AE can be configured to support NAT
traversal, excluding one or both of the UDP ports [RFC6978].

6. Echo request (REQ) and echo response (RES)

The echo request (REQ, kind=9) and echo response (RES, kind=10)
options provide a means for UDP options to be used to provide
packet-level acknowledgements. Their use is described as part of the
UDP variant of packetization layer path MTU discovery (PLPMTUD)
[Fa19]. The options both have the format indicated in Figure 20.

+--------+--------+------------------+
|  Kind  | Len=6  |      nonce       |
+--------+--------+------------------+
    1 byte   1 byte       4 bytes

Figure 20   UDP REQ and RES options format

6.1. Experimental (EXP)

The Experimental option (EXP) is reserved for experiments [RFC3692].
It uses a Kind value of 254. Only one such value is reserved because
experiments are expected to use an Experimental ID (ExIDs) to
differentiate concurrent use for different purposes, using UDP ExIDs
registered with IANA according to the approach developed for TCP
experimental options [RFC6994].
Figure 21  UDP EXP option format

>> The length of the experimental option MUST be at least 4 to account for the Kind, Length, and the minimum 16-bit UDP ExID identifier (similar to TCP ExIDs [RFC6994]).

7. Rules for designing new options

The UDP option Kind space allows for the definition of new options, however the currently defined options do not allow for arbitrary new options. For example, LITE needs to come first if present; new options cannot declare that they need to precede it. The following is a summary of rules for new options and their rationales:

>> New options MUST NOT depend on option space content. Only OCS and AE depend on the content of the options themselves and their order is fixed (on transmission, AE is computed first using a zero-checksum OCS if present, and OCS is computed last before transmission, over the entire option area, including AE).

>> New options MUST NOT declare their order relative to other options, whether new or old.

>> At the sender, new options MUST NOT modify UDP packet content anywhere except within their option field; areas that need to remain unmodified include the IP header, IP options, the UDP body, the UDP option area (i.e., other options), and the post-option area.

>> Options MUST NOT be modified in transit. This includes those already defined as well as new options. New options MUST NOT require or intend optionally for modification of any UDP options, including their new areas, in transit.

Note that only certain of the initially defined options violate these rules:

- >> LITE MUST be first, if present, and MUST be processed when encountered (e.g., even before security options).
- >> LITE is the only option that modifies the UDP body or option areas.
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 of 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

Voelker, "Packetization Layer Path MTU Discovery for
Datagram Transports," draft-ietf-tsvwg-datagram-plpmtud,

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate

[RFC768] Postel, J., "User Datagram Protocol," RFC 768, August
1980.


[RFC1071] Braden, R., D. Borman, C. Partridge, "Computing the

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts --


19.2. Informative References

Options for UDP Options", draft-fairhurst-udp-options-cco,

Datagrams (SPUD) Prototype," draft-hildebrand-spud-
prototype-03, Mar. 2015.

[RFC793] Postel, J., "Transmission Control Protocol" RFC 793,
September 1981.


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></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>
IPv6 Packet Truncation

draft-leddy-6man-truncate-05

Abstract

This document defines IPv6 packet truncation procedures. These procedures make Path MTU Discovery (PMTUD) more reliable. Upper-layer protocols can leverage these procedures in order to take advantage of large MTUs.

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 15, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

1. Introduction

An Internet path connects a source node to a destination node. A path can contain links and routers.

Each link is constrained by the number of bytes that it can convey in a single IP packet. This constraint is called the link Maximum Transmission Unit (MTU). IPv6 [RFC8200] requires every link to have an MTU of 1280 bytes or greater. This value is called IPv6 minimum link MTU.

Likewise, each Internet path is constrained by the number of bytes that it can convey in a single IP packet. This constraint is called the Path MTU (PMTU). For any given path, the PMTU is equal to the smallest of its link MTUs.

IPv6 allows fragmentation at the source node only. If an IPv6 source node sends a packet whose length exceeds the PMTU, an intermediate node will discard the packet. In order to prevent this, IPv6 nodes can either:
Refrain from sending packets whose length exceeds the IPv6 minimum link MTU.

Maintain a running estimate of the PMTU and refrain from sending packets whose length exceeds that estimate.

In order to maintain a running estimate of the PMTU, IPv6 nodes can execute Path MTU Discovery (PMTUD) [RFC8201] procedures. In these procedures, the source node produces an initial PMTU estimate. This initial estimate equals the MTU of the first link along the path to the destination. It can be greater than the actual PMTU.

Having produced an initial PMTU estimate, the source node sends packets to the destination node. If one of these packets is larger than the actual PMTU, an intermediate node will not be able to forward the packet through the next link along the path. Therefore, the intermediate node discards the packet and sends an Internet Control Message Protocol (ICMP) [RFC4443] Packet Too Big (PTB) message to the source node. The ICMP PTB message indicates the MTU of the link through which the packet could not be forwarded. The source node uses this information to refine its PMTU estimate.

PMTUD relies on the network to deliver ICMP PTB messages from the intermediate node to the source node. If the network cannot deliver these messages, a persistent black hole can develop. In this scenario, the source node sends a packet whose length exceeds the PMTU. An intermediate node discards the packet and sends an ICMP PTB message to the source. However, the network cannot deliver the ICMP PTB message to the source. Therefore, the source node does not update its PMTU estimate and it continues to send packets whose length exceeds the PMTU. The intermediate node discards these packets and sends more ICMP PTB messages to the source. These ICMP PTB messages are lost, exactly as previous ICMP PTB messages were lost.

In some operational scenarios (Section 3), networks cannot deliver ICMP PTB messages from an intermediate node to the source node. Therefore, enhanced procedures are required.

This document defines IPv6 packet truncation procedures. When an IPv6 source node originates a packet, it executes the following procedure:

Mark the packet as being eligible for truncation.

Forward the packet towards its destination.
If an intermediate node cannot forward the packet because of an MTU issue, it executes the following procedure:

- Detect that the packet is eligible for truncation.
- Send an ICMP PTB message to the source node, with the MTU field indicating the MTU of the link through which the packet could not be forwarded.
- Truncate the packet.
- Mark the packet as being truncated.
- Update the packet’s upper-layer checksum (if possible).
- Forward the packet towards its destination.

When the destination node receives the packet, it executes the following procedure:

- Detect that the packet has been truncated.
- Send an ICMP PTB message to the source node, with the MTU field indicating the length of the truncated packet.
- Discard the packet.

Both ICMP PTB messages, mentioned above, contain MTU information that the source node can use to refine its PMTU estimate.

The procedures described herein prevent incomplete (i.e., truncated) data from being delivered to upper-layer protocols. While IPv6 packet truncation may facilitate new upper-layer procedures, upper-layer procedures are beyond the scope of this document.

The procedures described herein make PMTUD more reliable by increasing the probability that the source node will receive ICMP PTB feedback from a downstream device. Even when the network cannot deliver ICMP PTB messages from an intermediate router to a source node, it may be able to deliver an ICMP PTB messages from the destination node to the source node.

However, the procedures described herein do not make PMTUD one hundred per cent reliable. In some operational scenarios, the network cannot deliver any ICMP messages to the source node, regardless of their origin.
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. Operational Considerations

The packet truncation procedures described herein make PMTUD more resilient when:

- The network can deliver ICMP messages from the destination node to the source node.
- The network cannot deliver ICMP messages from an intermediate node to the source node.

The following are common operational scenarios in which packet truncation procedures can make PMTUD more resilient:

- The destination node has a viable route to the source node, but the intermediate node does not.
- The source node is protected by a firewall that administratively blocks all packets except for those from specified subnetworks. The destination node resides in one of the specified subnetworks, but the intermediate node does not.
- The source address of the original packet (i.e., the packet that elicited the ICMP message) was an anycast address. Therefore, the destination address of the ICMP message is the same anycast address. In this case, an ICMP message from the destination node is likely to be delivered to the correct anycast instance. By contrast, an ICMP message from an intermediate node is less likely to be delivered to the correct anycast instance.

Packet truncation procedures do not make PMTUD more resilient when the network cannot reliably deliver any ICMP messages to the source node. The following are operational scenarios where the network cannot reliably deliver any ICMP PTB messages to the source node:

- The source node is protected by a firewall that administratively blocks all ICMP messages.
The source node is an anycast instance served by a load-balancer as defined in [RFC7690]. The load-balancer does not implement the mitigations defined in [RFC7690].

4. IPv6 Destination Options

This document defines the following IPv6 Destination options:

4.1. The Truncation Eligible Option

The Truncation Eligible option indicates that the packet is eligible for truncation. It also indicates that the packet has not been truncated.

The Truncation Eligible option contains the following fields:

- Option Type – Truncation Eligible option. Value TBD by IANA. See Notes below.
- Opt Data Len – Length of Option Data, measured in bytes. MUST be equal to 0.

IPv6 packets that include the Fragment header MUST NOT include the Truncation Eligible option.

IPv6 packets whose length is less than the IPv6 minimum link MTU SHOULD NOT include the Truncation Eligible option.

The IPv6 Hop-by-hop Options header SHOULD NOT include the Truncation Eligible option.

The IPv6 Destination Options header:

- MAY include a single instance of the Truncation Eligible option.
- SHOULD NOT include multiple instances of the Truncation Eligible option.
- MUST NOT include both the Truncation Eligible option and the Truncated Packet option (Section 4.2).

NOTE 1: According to [RFC8200], the highest-order two bits of the Option Type (i.e., the "act" bits) specify the action taken by a processing node that does not recognize Option Type. The required action is skip over this option and continue processing the header. Therefore, IANA is requested to assign this Option Type with "act" bits "00".
NOTE 2: According to [RFC8200], the third-highest-order bit (i.e., the "chg" bit) of the Option Type specifies whether Option Data can change on route to the packet's destination. Because this option contains no Option Data, IANA can assign this Option Type without regard to the "chg" bit.

4.2. The Truncated Packet Option

The Truncated Packet option indicates that the packet has been truncated and is eligible for further truncation.

The Truncated Packet option contains the following fields:

- Option Type - Truncated Packet option. Value TBD by IANA. See Notes below.
- Opt Data Len - Length of Option Data, measured in bytes. MUST be equal to 0.

IPv6 packets that include the Fragment header MUST NOT include the Truncated Packet option.

IPv6 packets whose length is less than the IPv6 minimum link MTU MUST NOT include the Truncated Packet option.

The IPv6 Hop-by-hop Options header SHOULD NOT include the Truncated Packet option.

The IPv6 Destination Options:

- MAY include a single instance of the Truncated Packet option.
- SHOULD NOT include multiple instances of the Truncated Packet option.
- MUST NOT include both the Truncated Packet option and the Truncation Eligible option.

NOTE 1: According to [RFC8200], the highest-order two bits of the Option Type (i.e., the "act" bits) specify the action taken by a processing node that does not recognize Option Type. The required action is to discard the packet and send an ICMP Parameter Problem, Code 2, message to the packet's Source Address, pointing to the unrecognized Option Type. Therefore, IANA is requested to assign this Option Type with "act" bits "10".

NOTE 2: According to [RFC8200], the third-highest-order bit (i.e., the "chg" bit) of the Option Type specifies whether Option Data of
that option can change on route to the packet’s destination. Because this option contains no Option Data, IANA can assign this Option Type without regard to the "chg" bit.

5. Reference Topology

```
<p>| Upper | | | | | | Upper |
| Layer | | | | | | Layer |
| IP |&lt;--------&gt; | IP |&lt;--------&gt; | IP |&lt;--------&gt; | IP |</p>
<table>
<thead>
<tr>
<th>Layer</th>
<th>MTU</th>
<th>Layer</th>
<th>MTU</th>
<th>Layer</th>
<th>MTU</th>
<th>Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Router 1</td>
<td>Router 2</td>
<td>Destination</td>
<td>Node</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 1: Reference Topology

Figure 1 depicts a network that contains a Source Node, intermediate nodes (i.e., Router 1, Router 2), and a Destination Node. The link that connects the Source Node to Router 1 has an MTU of 9000 bytes. The link that connects Router 1 to Router 2 has an MTU of 4000 bytes, and the link that connects Router 2 to the Destination Node has an MTU of 1500 bytes. The PMTU between the Source Node and the Destination Node is 1500 bytes.

This topology is used in examples throughout the document.

6. Truncation Procedures

In the Reference Topology (Figure 1), the Source Node produces an initial estimate of the PMTU between itself and the Destination Node. This initial estimate equals the MTU of the first link on the path to the Destination Node (e.g., 9000 bytes).

The Source Node refrains from sending packets whose length exceeds the above-mentioned estimate. However, the above-mentioned estimate is significantly larger than the actual PMTU (1500 bytes). Therefore, the Source Node may send packets whose length exceeds the actual PMTU.

At some time in the future, an upper-layer protocol on the Source Node causes the IP layer to emit a packet. The packet contains a Destination Options header and the Destination Options header contains a Truncation Eligible option. The total packet length, including all headers and the payload, is 1350 bytes. Because the total packet length is less than the actual PMTU, this packet can be
delivered to the Destination Node without encountering any MTU issues.

The IP layer on the Source Node forwards the packet to the Router 1, Router 1 forwards the packet to Router 2, and the Router 2 forwards the packet to the Destination Node. The IP layer on the Destination Node examines the Destination Options header and finds the Truncation Eligible option. The Truncation Eligible option requires no action by the Destination Node. Therefore, the Destination Node processes the next header and delivers the packet to an upper-layer protocol.

Subsequently, the same upper-layer protocol on the Source Node causes the IP layer to emit another packet. This packet is identical to the first, except that the total packet length is 2000 bytes. Because the packet length is greater than the actual PMTU, this packet cannot be delivered without encountering an MTU issue.

The IP layer on the source node forwards the packet to Router 1. Router 1 forwards the packet to Router 2, but the Router 2 cannot forward the packet because its length exceeds the MTU of the next link in the path (i.e., 1500 bytes). Because an MTU issue has been encountered, Router 2 examines the Destination Options header, searching for either a Truncation Eligible option or a Truncated Packet option. (Normally, the Router 2 would ignore the Destination Options header).

Because Router 2 finds one of the above-mentioned options, it:

- Sends an ICMP PTB message to the Source Node. The ICMP PTB message contains an MTU field indicating the MTU of the next link in the path (i.e., 1500 bytes).
- Truncates the packet, so that its total length equals the MTU of the next link in the path.
- Updates the IPv6 Payload Length.
- Overwrites all instances of the Truncation Eligible option with a Truncated Packet option.
- Updates the upper-layer checksum (if possible)
- Forwards the packet to the Destination Node.

The IP layer on the Destination Node receives the packet and examines the Destination Options header. Because it finds the Truncated Packet option, it discards the packet and sends an ICMP PTB message
to the Source Node. The MTU field in the ICMP PTB message represents the length of the received packet.

When the Source Node receives the ICMP PTB message, it updates its PMTU estimate, as per [RFC8201].

7. Additional Truncation Considerations

A packet can be truncated multiple times. In the Reference Topology (Figure 1), assume that the Source Node sends a 5000 byte packet to the Destination Node. Using the procedures described in Section 6, Router 1 truncates this packet to 4000 bytes and Router 2 truncates it again, to 1500 bytes.

A truncated packet MUST contain the basic IPv6 header, all extension headers and the first upper-layer header. When an intermediate node cannot forward a packet due to MTU issues, and the total length of the basic IPv6 header, all extension headers, and first upper-layer header exceeds the MTU of the next link in the path, the intermediate node MUST discard the packet and send and ICMP PTB message to the source node. It MUST NOT truncate the packet.

A truncated packet MUST NOT include the Fragment header. When an intermediate node cannot forward a packet due to MTU issues, and the packet contains a Fragment header, the intermediate node MUST discard the packet and send and ICMP PTB message to the source node. It MUST NOT truncate the packet.

A truncated packet must have a total length that is greater than or equal to the IPv6 minimum link MTU.

8. Backwards Compatibility

Section 6 of this document assumes that all nodes recognize the Truncation Eligible and Truncated Packet options. This section explores backwards compatibility issues, where one or more nodes do not recognize the above-mentioned options.

An intermediate node that does not recognize the above-mentioned options behaves exactly as described in [RFC8200]. When it receives a packet that does not cause an MTU issue, it processes the packet. When it receives a packet that causes an MTU issue, it discards the packet and sends an ICMP PTB message to the source node. In neither case does the intermediate node examine the Destination Options header or truncate the packet.

A destination node that does not recognize the Truncation Eligible option also behaves exactly as described in [RFC8200]. When it
receives a packet that contains the Truncation Eligible option, its behavior is determined by the highest-order two bits of the Option Type (i.e., the "act" bits). Because the "act" bits are equal to "00", the destination node skips over the option and continues to process the packet. This is exactly what the destination node would have done if it had recognized the Truncation Eligible option.

A destination node that does not recognize the Truncated Packet option also behaves exactly as described in [RFC8200]. When it receives a packet that contains the Truncated Packet option, its behavior is determined by the highest-order two bits of the Option Type (i.e., the "act" bits). Because the "act" bits are equal to "10", the destination node discards the packet and sends an ICMP Parameter Problem, Code 2, message to the packet’s Source Address, pointing to the Truncated Packet option. The destination node does not emit an ICMP PTB message.

The source node takes appropriate action when it receives the ICMP Parameter Problem message.

9. Checksum Considerations

When an intermediate node truncates a packet, it SHOULD update the upper-layer checksum, if possible. This is desirable because it increases the probability that the truncated packet will be delivered to the destination node.

Middleboxes residing downstream of the intermediate node may attempt to validate the upper-layer checksum. If validation fails, they may discard the packet without sending an ICMP message.

10. Invalid Packet Types

The following packet types are invalid:

- Packets that contain the Fragment header and the Truncation Eligible option.
- Packets that contain the Fragment header and the Packet Truncated option.
- Packets that contain the Truncation Eligible option and the Packet Truncated option.
- Packets that specify an Option Data Length greater than 0 in the Truncation Eligible option.
Packets that specify an Option Data Length greater than 0 in the 
Truncated Packet option.

Packets that have a total length less than the IPv6 minimum link 
MTU and contain the Packet Truncated option.

If an intermediate node cannot forward one of the above-mentioned packets because of an MTU issue, its behavior is as described in [RFC8200]. The intermediate node discards the packet and sends an ICMP PTB message to the source node. It does not truncate or forward the packet.

When the destination node receives one of the above-mentioned packets, it MUST:

- Discard the packet
- Send an ICMP Parameter Problem, Code 2, message to the packet’s Source Address, pointing to the first invalid option.

The destination node MUST NOT send an ICMP PTB message.

11. Network Considerations

The procedures described herein rely upon the network’s ability:

- To convey packets that contain destination options from the source node to the destination node.
- To convey ICMP Parameter Problem messages in the reverse direction.

Operational experience [RFC7872] reveals that a significant number of networks drop packets that contain IPv6 destination options. Likewise, many networks drop ICMP Parameter Problem messages.

[I-D.bonica-6man-unrecognized-opt] describes procedures that upper-layer protocols can execute to verify that the above-mentioned requirements are satisfied. Upper-layer protocols can execute these procedures before emitting packets that contain the Truncation Eligible option.

12. Encapsulating Security Payload Considerations

An IPv6 packet can contain both:

- An Encapsulating Security Payload (ESP) [RFC4303] header.
Truncation options (i.e., the Truncation Eligible or Truncated Packet options).

In this case, the packet MUST contain a Destination Options header that precedes the ESP. That Destination Options header contains the truncation options and is not protected by the ESP. The packet MAY also contain another Destination Options header that follows the ESP. That Destination Options header is protected by the ESP and MUST NOT contain the truncation options.

As per RFC 4303, a packet can contain two Destination Options headers, one preceding the ESP and one following the ESP.

13. Extension Header Considerations

According to [RFC8200], the following IPv6 extension headers can contain options:

- The Hop-by-hop Options header.
- The Destination Options header.

The Hop-by-hop option can be examined by each node along the path to a packet’s destination. Destination options are examined by the destination node only. However, [RFC2473] provides a precedent for intermediate nodes examining the Destination options on an exception basis. (See the Tunnel Encapsulation Limit.)

The truncation options described herein are examined by:

- Intermediate nodes, on an exception basis (i.e., when the packet cannot be forwarded due to MTU issues).
- The Destination node.

Therefore, the above-mentioned options can be processed most efficiently when they are contained by the Destination Option header. When contained by the Destination Options header, the above-mentioned options are examined by intermediate nodes on an exception basis, only when they are relevant. If contained by the Hop-by-hop Options header, they are always examined by intermediate nodes, even when they are irrelevant.

14. Security Considerations

PMTUD is vulnerable to ICMP PTB forgery attacks. The procedures described herein do nothing to mitigate that vulnerability.
The procedures described herein are susceptible to a new variation on that attack, in which an attacker forges a truncated packet. In this case, the attackers cause the Destination Node to produce an ICMP PTB message on their behalf. To some degree, this vulnerability is mitigated, because the Destination Node will not emit an ICMP PTB message in response to a truncated packet whose length is less than the IPv6 minimum link MTU.

In order to mitigate denial of service attacks, intermediate nodes MUST rate limit the number of packets that they truncate per second.

15. IANA Considerations

IANA is requested to allocate the following codepoints from the Destination Options and Hop-by-hop Options registry (https://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml#ipv6-parameters-2).

- Truncation Eligible ("act-bits" are "00. "chg-bit" can be either 0 or 1.)
- Truncated Packet ("act-bits" are "10". "chg-but can be either 0 or 1.)

16. Acknowledgements

Special thanks to Mike Heard, Geoff Huston, Joel Jaeggli, Tom Jones, Andy Smith, Jinmei Tatuya, and Reji Thomas who reviewed and commented on this document.

17. References

17.1. Normative References


17.2. Informative References

[I-D.bonica-6man-unrecognized-opt]

[ RFC2473 ]

[ RFC7690 ]

[ RFC7872 ]

Authors' Addresses

John Leddy
Unaffiliated

Email: john@leddy.net
SOCKS Protocol Version 6
draft-olteanu-intarea-socks-6-06

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 September 12, 2019.

Copyright Notice

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

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

Table of Contents

1. Introduction ................................................. 3
   1.1. Revision log ........................................ 4
2. Requirements language ......................................... 8
3. Mode of operation ........................................... 8
4. Requests ................................................ 10
5. Version Mismatch Replies ..................................... 12
6. Authentication Replies ........................................ 12
7. Operation Replies ............................................ 13
   7.1. Handling CONNECT ...................................... 14
   7.2. Handling BIND ......................................... 15
   7.3. Handling UDP ASSOCIATE ................................... 15
   7.3.1. Proxying UDP servers ................................ 17
8. SOCKS Options ............................................... 17
   8.1. Stack options ........................................ 17
   8.1.1. IP TOS options ..................................... 19
   8.1.2. TFO options ......................................... 19
   8.1.3. Multipath TCP options ................................ 20
   8.1.4. MPTCP Scheduler options .............................. 20
   8.1.5. Listen Backlog options ................................ 21
   8.2. Authentication Method options ............................ 22
   8.3. Authentication Data options ............................. 23
   8.4. Session options ........................................ 24
   8.4.1. Session initiation ................................... 24
   8.4.2. Further SOCKS Requests ............................... 26
   8.4.3. Tearing down the session ............................. 27
   8.5. Idempotence options .................................... 28
   8.5.1. Requesting a fresh token window ..................... 29
   8.5.2. Spending a token ..................................... 30
   8.5.3. Handling Token Window Advertisements .................. 32
9. Username/Password Authentication ............................ 32
10. TCP Fast Open on the Client-Proxy Leg ...................... 32
11. False Starts ................................................ 33
12. Security Considerations ..................................... 33
   12.1. Large requests ....................................... 33
   12.2. Replay attacks ....................................... 34
   12.3. Resource exhaustion ................................... 34
13. IANA Considerations .......................................... 34
14. Acknowledgements ............................................ 34
15. References ................................................ 35
   15.1. Normative References .................................. 35
   15.2. Informative References ................................. 35
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.
o The protocol can leverage the aforementioned options to support 0-RTT authentication schemes.

1.1. Revision log

Typos and minor clarifications are not listed.

draft-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 Shrank vendor-specific option range to 32 (down from 64).

o Removed reference to dropping initial data. (It could no longer be done as of -05.)
o Initial data size capped at 16KB.

o Application data is never encrypted by SOCKS 6. (It can still be encrypted by the TLS layer under SOCKS.)

o Messages now carry the total length of the options, rather than the number of options. Limited options length to 16KB.

o Security Considerations
   * Updated the section to reflect the smaller maximum message size.
   * Added a subsection on resource exhaustion.

draft-05

o Limited the "slow" authentication negotiations to one (and Authentication Replies to 2)

o Revamped the handling of the first bytes in the application data stream
   * False starts are now recommended. (Added the "False Start" section.)
   * Initial data is only available to clients willing to do "slow" authentication. Moved the "Initial data size" field from Requests to Authentication Method options.
   * Initial data size capped at 2^13. Initial data can no longer be dropped by the proxy.
   * The TFO option can hint at the desired SYN payload size.

o Request: clarified the meaning of the Address and Port fields.

o Better reverse TCP proxy support: optional listen backlog for TCP BIND

o TFO options can no longer be placed inside Operation Replies.

o IP TOS stack option

o Suggested a range for vendor-specific options.

o Revamped UDP functionality
* Now using fixed UDP ports
* DTLS support

- Stack options: renamed Proxy-Server leg to Proxy-Remote leg

draft-04

- Moved Token Expenditure Replies to the Authentication Reply.
- Shifted the Initial Data Size field in the Request, in order to make it easier to parse.

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.
  o NOOP commands now trigger Operation Replies.
  o Renamed Authentication options to Authentication Data options.
  o Authentication Data options are no longer mandatory.
  o Authentication methods are now advertised via options.
  o Shifted some Request fields.
- Option range for vendor-specific options.
- Socket options.
- Password authentication.
- Salt options.

draft-01
- 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
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        | Command | Port | Address | Address  |
| Major | Minor | Code |      | Type |          |
|-------+-------+------|------+---------+----------|
| 1     | 1     | 1    | 2     | 1   | Variable |
|-------+-------+------|------+---------+----------|
+-------------------+---------+------+---------+----------+
| Options | Options |    |
| Length |          |    |
|--------+----------+    |
| 2      | Variable |    |
|--------+----------+    |
```

Figure 2: SOCKS 6 Request

- Version: The major byte is 0x06, and the minor byte is 0x00.
- 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.
  * IPv6: a 16-byte IPv6 address

- Port: the port in network byte order.

- 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 option (see Section 8.2).
5. Version Mismatch Replies

Upon receipt of a request starting with a version number other than 6.0, the proxy sends the following response:

```
+---------------+
|    Version    |
| Major | Minor |
+-------+-------+
|   1   |   1   |
+-------+-------+
```

Figure 3: SOCKS 6 Version Mismatch Reply

- **Version**: The major byte is 0x06, and the minor byte is 0x00.
- 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:

```
+---------------+-----------------+----------+----------+
|    Version    | Type            | Method   | Options  |
| Major | Minor |      |        | Length  |          |
+-------+-------+------+--------+---------+----------+
|   1   |   1   |  1   |   1    |    2    | Variable |
+-------+-------+------+--------+---------+----------+
```

Figure 4: SOCKS 6 Authentication Reply

- **Version**: The major byte is 0x06, and the minor byte is 0x00.
- **Type**:
  - 0x00: authentication successful.
  - 0x01: further authentication needed.
- **Method**: The chosen authentication method.
- **Options Length**: the total size of the SOCKS options that appear in the Options field. MUST NOT exceed 16KB.
- **Options**: see Section 8.
Multihomed clients SHOULD cache the chosen method on a per-interface basis and SHOULD NOT include Authentication Data options related to any other methods in further requests originating from the same interface.

If the server signals that further authentication is needed and selects "No Acceptable Methods", 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 either signal success or that there are no more acceptable authentication methods.

7. Operation Replies

After the authentication negotiations are complete, the proxy sends an Operation Reply:

```
+-------+------+---------+----------+---------+----------+
| Reply | Bind | Address |   Bind   | Options | Options  |
| Code  | Port |  Type   | Address  | Length  |          |
+-------+------+---------+----------+---------+----------+
|   1   |  2   |    1    | Variable |    2    | Variable |
+-------+------+---------+----------+---------+----------+
```

Figure 5: SOCKS 6 Operation Reply

- 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

- **Bind Port**: the proxy bound port in network byte order.
- **Address Type**:
  * 0x01: IPv4
  * 0x03: Domain Name
  * 0x04: IPv6
- **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.
  * IPv6: a 16-byte IPv6 address
- **Options Length**: the total size of the SOCKS options that appear in the Options field. MUST NOT exceed 16KB.
- **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:

```
+----------------+
| Association ID |
+----------------+
|        4       |
+----------------+
```

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:

```
+---------------+-------------+------+---------+----------+
|    Version    | Association | Port | Address | Address  |
| Major | Minor | ID      |      |  Type   |          |
+-------+-------+---------+------+---------+----------+
|   1   |   1   |  4      |  2   |    1    | Variable |
+-------+-------+---------+------+---------+----------+
```

Figure 7: SOCKS 6 Datagram Header

- Version: The major byte is 0x06, and the minor byte is 0x00.
- 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:

```
+--------+
| Status |
+--------+
|   1    |
+--------+
```

- 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.

8. SOCKS Options

SOCKS options have the following format:

```
+-------+--------+-------------+
| Kind  | Length | Option Data |
+-------+--------+-------------+
|  1    |   2    | Variable    |
+-------+--------+-------------+
```

Figure 9: SOCKS 6 Option

- Kind: Allocated by IANA. (See Section 13.)
- Length: The total length of the option.
- Option Data: The contents are specific to each option kind.

Unless otherwise noted, client and proxy implementations MAY omit supporting any of the options described in this document. Upon encountering an unsupported option, a SOCKS endpoint MUST silently ignore it.

8.1. Stack options

Stack options can be used by clients to alter the behavior of the protocols on top of which SOCKS is running, as well the protocols used by the proxy to communicate with the remote host (i.e. IP, TCP,
UDP). A Stack option can affect either the proxy’s protocol on the client-proxy leg or on the proxy-remote leg. Clients can only place Stack options inside SOCKS Requests.

Proxies MAY include Stack options in their Operation Replies to signal their behavior. Said options MAY be unsolicited, i.e., the proxy MAY send them to signal behaviour that was not explicitly requested by the client.

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.

```
+------+--------+--------+--------+------+----------+
| Kind | Length |  Leg   | Level  | Code |   Data   |
+------+--------+--------+--------+------+----------+
|  1   |   2    |  2 bits|  6 bits|  1   | Variable |
+---------------------------------------+
```

Figure 10: Stack Option

- Kind: 0x01 (Stack option)
- Length: The length of the option.
- Leg:
  - 0x1: Client-Proxy Leg
  - 0x2: Proxy-Remote Leg
  - 0x3: Both Legs
- Level:
  - 0x01: IP: options that apply to either IPv4 or IPv6
  - 0x02: IPv4
  - 0x03: IPv6
  - 0x04: TCP
  - 0x05: UDP
8.1.1. IP TOS options

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Leg</th>
<th>Level</th>
<th>Code</th>
<th>TOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2 bits</td>
<td>6 bits</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 11: IP TOS Option

- Kind: 0x01 (Stack option)
- Length: 4
- Leg: Either 0x01, 0x02, or 0x03 (Client-Proxy, Proxy-Remote or Both legs)
- Level: 0x04 (TCP).
- Code: 0x01
- TOS: The IP TOS code

The client can use IP TOS options to request that the proxy use a certain value for the IP TOS field. Likewise, the proxy can use IP TOS options to advertise the TOS values being used.

8.1.2. TFO options

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Leg</th>
<th>Level</th>
<th>Code</th>
<th>Payload Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2 bits</td>
<td>6 bits</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 12: TFO Option

- Kind: 0x01 (Stack option)
- Length: 4
- Leg: 0x02 (Proxy-Remote leg).
8.1.3. Multipath TCP options

In case of a CONNECT command, the proxy can inform the client that the connection to the server is an MPTCP connection.

+----------------+--------+--------+--------+------+
| Kind | Length | Leg    | Level  | Code |
|-------|--------|--------|--------|------+
| 1    | 2      | 2 bits | 6 bits | 1    |
+----------------+--------+--------+--------+------+

Figure 13: Multipath TCP Option

o Kind: 0x01 (Stack option)

o Length: 4

o Leg: 0x02 (Proxy-Remote leg)

o Level: 0x04 (TCP).

o Code: 0x02

8.1.4. MPTCP Scheduler options

In case of a CONNECT or BIND command, a client can use an MPTCP Scheduler option to indicate its preferred scheduler for the connection.
A proxy can use an MPTCP Scheduler option to inform the client about what scheduler is in use.

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Leg</th>
<th>Level</th>
<th>Code</th>
<th>Scheduler</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2 bits</td>
<td>6 bits</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 14: MPTCP Scheduler Option

- Kind: 0x01 (Stack option)
- Length: 5
- Leg: Either 0x01, 0x02, or 0x03 (Client-Proxy, Proxy-Remote or Both legs).
- Level: 0x04 (TCP)
- Code: 0x03
- Scheduler:
  * 0x01: Lowest latency first
  * 0x02: Redundant

8.1.5. Listen Backlog options

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Leg</th>
<th>Level</th>
<th>Backlog</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2 bits</td>
<td>6 bits</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 15: Listen Backlog Option

- Kind: 0x01 (Stack option)
- Length: 5
- Leg: 0x02 (Proxy-Remote leg)
- Level: 0x04 (TCP)
o Code: 0x04

o Backlog: The length of the listen backlog. MUST be greater than 1.

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 a proxy can not or will not honor a Listen Backlog option, it does so silently.

8.2. Authentication Method options

Authentication Method options are placed in SOCKS Requests to advertise supported authentication methods. In case of a CONNECT Request, they are also used to specify the amount of initial data supplied before any method-specific authentication negotiations take place.

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Initial Data Length</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>Variable</td>
</tr>
</tbody>
</table>

Figure 16: Authentication Method Option

o Kind: 0x02 (Authentication Method option)
8.3. Authentication Data options

Authentication Data options carry method-specific authentication data. They can be part of SOCKS Requests and Authentication Replies.

Authentication Data options have the following format:

+--------+--------+--------+---------------------+
| Kind   | Length | Method | Authentication Data |
+--------+--------+--------+---------------------+
| 1      | 2      | 1      | Variable            |
+--------+--------+--------+---------------------+

Figure 17: Authentication Data Option

- Kind: 0x03 (Authentication Data option)
- Length: The length of the option.
- Method: The number of the authentication method. These numbers are assigned by IANA.
- Authentication Data: The contents are specific to each method.

Clients SHOULD only place one Authentication Data option per authentication method. Server implementations MAY silently ignore all Authentication Data options for the same method aside from an arbitrarily chosen one.
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.

Session Options have the following format:

```
+-----------------------------+-----------------------------+
<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Type</th>
<th>Session Option Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>Variable</td>
</tr>
</tbody>
</table>
+-----------------------------+-----------------------------+
```

Figure 18: Session Option

- Kind: 0x04 (Session option)
- Length: The length of the option.
- Type:
  - 0x01: Session Request
  - 0x02: Session ID
  - 0x02: Session Teardown
  - 0x04: Session OK
  - 0x05: Session Invalid
  - 0x06: Session Untrusted
- Session Option Data: The contents are specific to each type.

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:
## Session Request Option

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x04</td>
<td>4</td>
<td>0x01</td>
</tr>
</tbody>
</table>

### Figure 19: Session Request Option

- **Kind**: 0x04 (Session option)
- **Length**: 4
- **Type**: 0x01 (Session Request)

The proxy then replies with a Session ID Option in the successful Operation Reply:

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Type</th>
<th>Session ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x04</td>
<td>Variable</td>
<td>0x02</td>
<td>Variable</td>
</tr>
</tbody>
</table>

### Figure 20: Session ID Option

- **Kind**: 0x04 (Session option)
- **Length**: The length of the option.
- **Type**: 0x02 (Session ID)
- **Session ID**: An opaque sequence of bytes specific to the session. MUST be at least one byte in size.

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:
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 cannot 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 20).

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 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:

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 21: Session Untrusted Option

- Kind: 0x04 (Session option)
- Length: 5
- Type: 0x06 (Session Untrusted)
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 and the Method field to 0xff). Further, it SHALL contain a Session Invalid option:

```
+------+--------+------+
| Kind | Length | Type |
+------+--------+------+
|  1   |   2    |  1   |
+----------------------------------

Figure 23: Session Invalid Option
```

- Kind: 0x04 (Session option)
- Length: 5
- Type: 0x05 (Session Invalid)

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.

Clients can signal that a session is no longer needed, and can be torn down, by sending a Session Teardown option in addition to the Session ID option:
Figure 24: Session Teardown Option

- Kind: 0x04 (Session option)
- Length: 4
- Type: 0x02 (Session Teardown)

After sending such an option, the client MUST assume that the session is no longer valid.

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. Windows can be shifted (i.e., have their base increased, while retaining their size) unilaterally by the proxy.

Requesting and spending tokens is done via Idempotence options:

Figure 25: Idempotence Option

- Kind: 0x05 (Idempotence option)
- Length: The length of the option.
o Type:
   * 0x01: Token Request
   * 0x02: Token Window Advertisement
   * 0x03: Token Expenditure
   * 0x04: Token Expenditure Reply

o Option Data: The contents are specific to each type.

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 fresh token window

A client can obtain a fresh window of tokens by sending a Token Request option as part of a SOCKS Request:

```
+------+--------+------+-------------+
| Kind | Length | Type | Window Size |
+------+--------+------+-------------+
|  1   |   2    |  1   |      4      |
+------+--------+------+-------------+
```

Figure 26: Token Request

o Kind: MUST be allocated by IANA. (See Section 13.)

o Length: 7

o Type: 0x00 (Token Request)

o Window Size: The requested window size.
If a token window is issued, the proxy then includes a Token Window Advertisement option in the corresponding successful Authentication Reply:

+-----------------+--------+--------------+---------------+
| Kind | Length | Type | Window Base | Window Size |
+-----------------+--------+--------------+---------------+
| 1 | 2 | 1 | 4 | 4 |
+-----------------+--------+--------------+---------------+

Figure 27: Token Window Advertisement

- Kind: 0x05 (Idempotence option)
- Length: 11
- Type: 0x01 (Token Grant)
- Window Base: The first token in the window.
- Window Size: The window size. This value SHOULD be lower or equal to 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.

8.5.2. Spending a token

The client can attempt to spend a token by including a Token Expenditure option in its SOCKS request:

+-----------------+--------+--------------+
| Kind | Length | Type | Token |
+-----------------+--------+--------------+
| 1 | 2 | 1 | 4 |
+-----------------+--------+--------------+

Figure 28: Token Expenditure

- Kind: 0x05 (Idempotence option)
- Length: 7
- Type: 0x02 (Token Expenditure)
Clients SHOULD prioritize spending the smaller tokens.

The proxy responds by sending a Token Expenditure Reply option as part of the successful Authentication Reply:

```
+------+--------+------+---------------+
| Kind | Length | Type | Response Code |
| 1    | 2      | 1    | 1             |
+------+--------+------+---------------+
```

Figure 29: Token Expenditure Response

- Kind: 0x05 (Idempotence option)
- Length: 4
- Type: 0x03 (Token Expenditure Response)
- Response Code:
  * 0x01: Success: The token was spent successfully.
  * 0x02: No Window: The proxy does not have a token window associated with the client.
  * 0x03: Out of Window: The token is not within the window.
  * 0x04: Duplicate: The token has already been spent.

If eligible, the token is spent before attempting to honor the Request. If the token is not eligible for spending, the proxy MUST NOT attempt to honor the client’s Request; further, it MUST indicate a General SOCKS server failure in the Operation Reply.

Proxy implementations SHOULD also send a Token Window Advertisement if:

- the token is out of window, or
- by the proxy’s internal logic, successfully spending the token caused the window to shift.

Proxy implementations SHOULD NOT shift the window’s base beyond the highest unspent token.
Proxy implementations MAY include a Token Window Advertisement in any Authentication Reply that indicates success.

8.5.3. Handling Token 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 unsolicited Token Window Advertisements with a Window Base less than the previously known value.

9. Username/Password Authentication

Username/Password authentication is carried out as in [RFC1929]. Clients can also attempt to authenticate by placing the Username/Password request in an Authentication Data Option.

+-----------------+--------+-----------------+---------------------------+
| Kind | Length | Method | Username/Password request |
+-----------------+--------+-----------------+---------------------------+
|  1   |   2    |   1    |          Variable         |
+-----------------+--------+-----------------+---------------------------+

Figure 30: Password authentication via a SOCKS Option

- Kind: 0x03 (Authentication Data option)
- Length: The length of the option.
- Method: 0x02 (Username/Password).
- Username/Password request: The Username/Password request, 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 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, 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 1-byte option kinds for SOCKS 6 options. Further, this document requests the following option kinds:

- Stack options: 0x01
- Authentication Method options: 0x02
- Authentication Data options: 0x03
- Session options: 0x04
- Idempotence options: 0x05
- A range for vendor-specific options: 0xE0-0xFF

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

Email: vladimir.olteanu@cs.pub.ro
Path MTU discovery solution space
draft-troan-6man-pmtu-solution-space-00

Abstract

Path MTU discovery has turned out to be a thorny problem that has haunted the Internet community for decades. Lately there has been some work both at the transport layer and at the network layer. This memo lists the solutions the author is aware of from the perspective of the network layer.

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 1, 2019.

Copyright Notice

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

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

Path MTU discovery has turned out to be harder than expected. In IPv6 we set out following the same model as for IPv4. The sending host maintains a MTU cache, that is updated based on received ICMP PMTUD messages. That solution has a few shortcomings:

- Sending of ICMP PMTUD messages is throttled in routers [RFC4443]
- It’s not efficient if links along the path have decreasingly smaller MTU, then multiple rounds of large packet, resulting ICMP PMTUD happens.
- ICMP might be ignored by host stacks / applications
- As ICMP looks different than application traffic, it might be blocked by routers.
- Doesn’t work well in an anycast scenario (but what does).

2. Requirements / Goals

1. Avoid MTU black-holes [RFC2923].
2. Detect the Path MTU in single round trip.
3. Adapt to varying MTU over the connection life time.
4. The signalling of the MTU back to the sender must be indistinguishable from application traffic to lessen risk of filtering.
5. Design a mechanism that ensures that neither MTU probes nor MTU signalling back to sender are more likely to be dropped than other application traffic.
6. Must be deployable and anchored in transport / application areas.
   Otherwise https://xkcd.com/927/
7. [Optional?] Support neighbors on the same link which support higher MTU than link MTU see [I-D.van-beijnum-multi-mtu]

3. Network layer solutions for Path MTU discovery

- PMTUD [RFC8201]
- On-path fragmentation, IPv4 style. We know this one.
Packet truncation. [I-D.leddy-6man-truncate]. The source sets a truncation eligible flag in the packet, routers on the path may truncate if the packet is too big, and sets a truncated done flag. Then the receiver signals the learnt forward MTU back to the sender. Either via existing ICMP PMTUD or a transport layer option. This is an example of a solution which does not require the sender having to accept packets from intermediate nodes.

MTU recording. Probe packets are sent, either as part of data packets, if those are guaranteed not to exceed MTU. Some trigger in the header (ECN like flags) or a HBH option is required for the router to record the smallest MTU along the path. Application / Transport would have to periodically include the probe trigger in data packets to detect changes in path MTU.

3.1. Common problems

How is the router along the path "triggered" to put this packet on the exception path? For current and the truncation scheme it’s a simple check in the forwarding path for the size of packet versus outgoing interface MTU. For e.g. a recording MTU mechanism it would have to be flags in the IPv6 header or an HBH option.

How should the forward path MTU be signalled back to the sender? The signal should look like any other application traffic to avoid filtering or is it sufficient to avoid sending from intermitent nodes.

4. Solutions at other layers

In addition there are solutions at the transport layer, that work in co-hort or independently of the network layer solutions. [RFC4821] and [I-D.ietf-tsvwg-datagram-plpmtud].

One could also imagine other solutions, e.g. to include MTU in router advertisements in BGP, so that a BGP speaker could calculate the end to end MTU across the set of administrative domains.

5. Conclusion

What are our options? Even if we developed a new PMTU mechanism, IP stacks must deal with networks where the new mechanism isn’t yet deployed. Will a new mechanism be so much better that it provides enough value for it to be deployed? Or should we at the network layer just punt this to transport?
6. References

[I-D.ietf-tsvwg-datagram-plpmtud]
Fairhurst, G., Jones, T., Tuexen, M., and I. Ruengeler,
"Packetization Layer Path MTU Discovery for Datagram Transports",
draft-ietf-tsvwg-datagram-plpmtud-04 (work in progress), September 2018.

[I-D.leddy-6man-truncate]
Leddy, J. and R. Bonica, "IPv6 Packet Truncation",
draft-leddy-6man-truncate-04 (work in progress), June 2018.

[I-D.van-beijnum-multi-mtu]
Beijnum, I., "Extensions for Multi-MTU Subnets",
draft-van-beijnum-multi-mtu-05 (work in progress), March 2016.

[RFC2923] Lahey, K., "TCP Problems with Path MTU Discovery",

STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006,
<https://www.rfc-editor.org/info/rfc4443>.

RFC 4821, DOI 10.17487/RFC4821, March 2007,

"Path MTU Discovery for IP version 6",
STD 87, RFC 8201, DOI 10.17487/RFC8201, July 2017,

Author’s Address

Ole Troan
Cisco Systems
Philip Pedersens vei 1
Lysaker 1366
Norway

Email: ot@cisco.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 queuing 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 HFF or 2 seconds using DPP, whereas it should be considered as NQB for the entire duration.

9. Acknowledgements

TBD

10. IANA Considerations

This draft proposes the registration of a standardized DSCP = 0x2A to denote Non-Queue-Building behavior.

11. Security Considerations

There is no incentive for an application to mismark its packets as NQB (or vice versa). If a queue-building flow were to mark its packets as NQB, it could experience excessive packet loss (in the case that queue-protection is not supported by a node) or it could receive no benefit (in the case that queue-protection is supported). If a non-queue-building flow were to fail to mark its packets as NQB, it could suffer the latency and loss typical of sharing a queue with capacity seeking traffic.
The NQB signal is not integrity protected and could be flipped by an on-path attacker. This might negatively affect the QoS of the tampered flow.

12. Informative References


Internet-Draft          Non Queue Building Flows              March 2019

             Iyengar, Ed., "Controlled Delay Active Queue Management",
             RFC 8289, DOI 10.17487/RFC8289, January 2018,

[ RFC8290 ] Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys,
             J., and E. Dumazet, "The Flow Queue CoDel Packet Scheduler
             and Active Queue Management Algorithm", RFC 8290,
             DOI 10.17487/RFC8290, January 2018,

Author’s Address

Greg White
CableLabs
858 Coal Creek Circle
Louisville, CO 80027
US

Email: g.white@cablelabs.com