Interactions between Low Latency, Low Loss, Scalable Throughput (L4S) and Differentiated Services
draft-briscoe-tsvwg-l4s-diffserv-01

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on January 3, 2019.

Copyright Notice

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

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

Table of Contents

1. Introduction .................................................. 2
   1.1. Terminology .............................................. 3
   1.2. Document Roadmap ......................................... 4
2. Architectural Comparison of L4S and Diffserv .......................... 4
   2.1. Overlaps between L4S and Diffserv ....................... 4
   2.2. Differences between L4S and Diffserv ................... 4
3. Low Latency Diffserv Classes within a DualQ Bandwidth Pool .......... 5
4. DualQ Bandwidth Pool within a Hierarchy of (Diffserv) ............... 9
   4.1. DualQ Complemented by an Assured Bandwidth Service ........ 10
   4.2. DualQ Complemented by a Guaranteed Low Latency Service . 12
   4.3. DualQ Complemented by a Scavenger Service .............. 13
5. Coupling More than Two AQMs within a Bandwidth Pool ................ 14
6. Best Practice for Classification and Marking ........................ 14
   6.1. Never Re-Mark a DSCP ................................... 14
   6.2. Classification Order ..................................... 15
      6.2.1. Classification Order: Problem ....................... 15
      6.2.2. Classification Order: Solutions .................... 15
7. Policing and Traffic Conditioning .................................. 15
8. IANA Considerations ............................................. 16
9. Security Considerations .......................................... 16
10. Comments Solicited ............................................. 16
11. Acknowledgements ............................................... 16
12. References ..................................................... 16
   12.1. Normative References .................................. 16
   12.2. Informative References ................................ 16
Appendix A. Open Issues ............................................. 18
Author's Address .................................................. 18

1. Introduction

The Low Latency Low Loss Scalable throughput (L4S) Internet service
[I-D.ietf-tsvwg-l4s-arch] provides a new Internet service that could
eventually replace best efforts, but with ultra-low queuing delay and
loss. A structure called the Dual-Queue Coupled AQM manages to
provide the L4S service alongside a second queue for Classic Internet
traffic, but without prejudging the bandwidth allocations between
them. L4S is orthogonal to allocation of bandwidth, so it can be
complemented by various bandwidth allocation approaches without
prejudging which one.
The Differentiated Services (Diffserv) architecture [RFC2475] provides for various service classes, some defined globally, others defined locally per network domain. Certain of these service classes offer low latency and low loss, as well as differentiated allocation of bandwidth.

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

1.1. Terminology

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

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

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

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

Pure L4S: L4S without unresponsive traffic.

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


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

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

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

1.2. Document Roadmap

(ToDo)

2. Architectural Comparison of L4S and Diffserv

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

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

2.1. Overlaps between L4S and Diffserv

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

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

2.2. Differences between L4S and Diffserv

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

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

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

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

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

3. Low Latency Diffserv Classes within a DualQ Bandwidth Pool

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

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

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

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

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

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

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

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

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

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

Notes for Table 1:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4.1. DualQ Complemented by an Assured Bandwidth Service

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

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

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

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

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

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

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

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

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

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

4.2. DualQ Complemented by a Guaranteed Low Latency Service

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

- Token bucket
- Rate/burst limiter

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>--------</td>
<td>-------------------</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------</td>
</tr>
</tbody>
</table>

Guaranteed low latency service is complemented by a DualQ with a guaranteed low latency service. The operator classifies certain packets into the guaranteed latency queue, perhaps by class of service, source address or 5-tuple flow ID.

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

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

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

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

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

Briscoe
Expires January 3, 2019

[Page 12]
all entry points to the network. For an access link into a network, these two alternative would amount to the same thing.

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

4.3. DualQ Complemented by a Scavenger Service

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

```
  ,--------++               Conditional
    L   .->|---. priority
  DualQ --/---++   c\.-.scheduler
          (Coupling ( )--.
    b/w < ~      (   )--.
      ----+--
  pool  |   \   |---' Priority
      \   |---' 1\.-.scheduler
         \   ( )-->
           \   /'-'
             \-''
  Bandwidth|scavenger |------'
         +--------'
```

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

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

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

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

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

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

![Diagram of Coupling the Classic AQM to Multiple L4S AQMs]

Figure 5: Coupling the Classic AQM to Multiple L4S AQMs

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

6. Best Practice for Classification and Marking

6.1. Never Re-Mark a DSCP

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

6.2.1. Classification Order: Problem

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

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

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

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

6.2.2. Classification Order: Solutions

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

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

7. Policing and Traffic Conditioning

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

This specification contains no IANA considerations.

9. Security Considerations

(ToDo)

10. 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.

11. Acknowledgements

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

12. References

12.1. Normative References


12.2. Informative References


Appendix A. Open Issues

- The Abstract promises "in which cases one can stand alone without needing the other", but that’s TBA in the text.
- Answer the interaction question between Diffserv and L4S the other way round as well: Which global PHBs is a DualQ applicable to?
- Document Roadmap TBA
- Mapping to 802.11 user priorities (or LTE QCIs)? Not strictly within the scope, but perhaps desirable to add, or at least to mention how L4S (experimental) would affect RFC8325 which gives (standards track) mappings between Diffserv and 802.11.
- Identify L4S-friendly rate policers
- Comparison between L4S policing and Diffserv traffic conditioning is TBA
- Security Considerations are TBA (largely depends on the previous bullet)

Author’s Address

Bob Briscoe
CableLabs
UK

Email: ietf@bobbriscoe.net
URI: http://bobbriscoe.net/
The Impact of Transport Header Confidentiality on Network Operation and Evolution of the Internet

draft-fairhurst-tsvwg-transport-encrypt-10

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 February 28, 2019.

Copyright Notice

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

This document describes implications 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 expected implications of transport protocol design and network operation. It
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 the interference with Internet traffic have led to a rapidly expanding deployment of encryption to protect end-user privacy, in protocols like QUIC [I-D.ietf-quic-transport], but also expected to form a basis of future 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.
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 some 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 evolution of the protocol).

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; or devices that inspect, and change, TCP MSS options that can interfere with path MTU discovery.

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 imply 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 may 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 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 understanding of 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.

A level of ossification of the transport header can offer trade-offs around authentication, and confidentiality of transport protocol headers and has the potential to explicitly support for other uses of this header information. 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 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.

Providing confidentiality of the transport payload, but leaving some, or all, of the transport headers unencrypted, possibly with authentication, can provide the majority of the privacy and security benefits while allowing some measurement.

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 may 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 what they cannot interpret. 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 may imply "don't touch" to some, and could limit a trouble-shooting response to "can't help, no trouble found". The additional mechanisms that will need to be introduced to help reconstruct transport-level metrics 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 the impact may, 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. For example, one approach could be to employ an existing transport protocol that reveals little information (e.g., UDP), and perform traditional transport functions at higher layers protecting the confidentiality of transport information. Such a design, 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 performance cannot be independently verified by all parties. The ability of other stakeholders to review code can help develop deeper insight. 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 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 important to demonstrate regulatory compliance in some jurisdictions, and provides an important 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 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, and on the ability to verify that others also conform.

o Operational practice: Published transport specifications allow operators to check compliance. This can bring assurance to those operating networks, often avoiding the need to deploy complex techniques that routinely monitor and manage TCP/IP 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.

o 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 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, 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. 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. Any new Internet transport need to provide appropriate transport mechanisms and operational support to assure the resulting traffic can not result in
persistent congestion collapse [RFC2914]. This document suggests that the balance between information exposed and concealed should be carefully considered when specifying new 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;

- The protocol and version of the header need to be visible. As protocols evolve over time and there may be a need to introduce new transport headers. This may require interpretation of protocol version information or connection setup information;

- The location and syntax of any observed transport headers needs 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 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 signalling protocols or through the use of magic numbers placed in the first byte(s) of the datagram payload.

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 uses observed traffic to makes inferences from transport headers to derive these measurements. 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, length) allows derivation of volume measures per-application, to characterise the traffic that uses a network segment or the pattern of network usage. This may 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 (providing detail of trends rather than the volume per subscriber).

Loss Rate and Loss Pattern: Flow loss rate may be derived (e.g., from sequence number) and has been used as a metric for
performance assessment and to characterise transport behaviour. Understanding the 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]), inadequate provision of traffic preemption. Understanding flow loss rate requires either maintaining per flow packet counters or by observing sequence numbers in transport headers. Loss can be monitored at the interface level by devices in the network. It is often important 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 (observing loss reports, e.g., RTP Control Protocol (RTCP) [RFC3550], TCP SACK) can increase understanding of the impact of loss and help identify cases where loss may have been wrongly identified, or the transport did not require the lost packet. It is sometimes more important 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 Protocol, RTP, headers [RFC3550]).

Latency: Latency is a key performance metric that impacts application response time and user-perceived response time. It often indirectly impacts throughput and flow completion time. Latency determines the reaction time of the transport protocol itself, impacting flow setup, congestion control, loss recovery, and other transport mechanisms. The observed latency can have many components [Latency]. Of these, unnecessary/unwanted queuing in network buffers has often been observed as a significant factor. Once the cause of unwanted latency has been identified, this can often be eliminated.
To measure latency across a part of a path, an observation point 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 has been used to locate a source of latency, e.g., by observing cases where the ratio of median to minimum RTT is large for a part of a path.

The service offered by 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. 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 flow reordering 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 promise 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.

Operational tools to detect mis-ordered packet flows and quantify the degree or reordering. 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 important 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.

3.1.3. Metrics derived from Network Layer Headers

Some transport information is made visible in the network-layer protocol header. These header fields are not encrypted and have been utilised to make flow observations.

Use of IPv6 Network-Layer Flow Label: Endpoints are encouraged to expose flow information in the IPv6 Flow Label field of the network-layer header (e.g., [RFC8085]). This can be used to inform network-layer queuing, forwarding (e.g., for Equal Cost Multi-Path, ECMP, routing, and Link Aggregation, LAG). This can provide useful information to assign packets to flows in the data collected by measurement campaigns. Although important to characterising a path, it does not directly provide performance data.

Use 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. This can be used to inform network-layer queuing and forwarding, and can also provide information on the relative importance of packet information collected by measurement campaigns, but does not directly provide performance data.

This field provides explicit information that can be used in place of inferring traffic requirements (e.g., by inferring QoS requirements from port information via a multi-field classifier).
The DSCP value can therefore 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.

Use of Explicit Congestion Marking: ECN [RFC3168] is an optional transport mechanism that uses a code point in the network-layer header. Use of ECN can offer gains in terms of increased throughput, reduced delay, and other benefits when used over a path that includes equipment that supports an AQM method that performs Congestion Experienced (CE) marking of IP packets [RFC8087].

ECN exposes the presence of congestion on a network path to the transport and network layer. The reception of CE-marked packets can therefore be used to monitor the presence and estimate the level of incipient congestion on the upstream portion of the path from the point of observation (Section 2.5 of [RFC8087]). Because ECN marks are carried in the IP protocol header, it is much easier to measure ECN than to measure packet loss. However, interpreting the marking behaviour (i.e., assessing congestion and diagnosing faults) requires context from the transport layer (path RTT, visibility of loss - that could be due to queue overflow, congestion response, etc) [RFC7567].

Some ECN-capable network devices can provide richer (more frequent and fine-grained) indication of their congestion state. Setting congestion marks proportional to the level of congestion (e.g., Data Center TCP, DCTP [RFC8257], and Low Latency Low Loss Scalable throughput, L4S, [I-D.ietf-tsvwg-l4s-arch].

Use of ECN requires a transport to feed back reception information on the path towards the data sender. Exposure of this Transport ECN feedback provides an additional powerful tool to understand ECN-enabled AQM-based networks [RFC8087].

AQM and ECN offer a range of algorithms and configuration options, it is therefore important for tools to be available to network operators and researchers to understand the implication of configuration choices and transport behaviour as use of ECN increases and new methods emerge [RFC7567] [RFC8087]. ECN monitoring is expected to become important as AQM is deployed that supports ECN [RFC8087].
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 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. 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 can be 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 Measurement

Often measurements can only be understood in the context of the other flows that share a bottleneck. A simple example is monitoring of AQM. For example, FQ-CODEL [RFC8290], combines sub queues (statistically assigned per flow), management of the queue length (CODEL), flow-scheduling, and a starvation prevention mechanism. Usually such algorithms are designed to be self-tuning, but current methods typically employ heuristics that can result in more loss under certain path conditions (e.g., large RTT, effects of multiple bottlenecks [RFC7567]).
In-network measurements can distinguish between upstream and downstream metrics with respect to a measurement point. These are particularly useful for locating the source of problems or to assess the performance of a network segment or a particular device configuration. 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 configurations in their networks. Data is also important to equipment vendors who need 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 may 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 not be possible 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 inform operational practice.

While active measurements may be used in-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 measurement. However, passive measurements may rely on observing transport headers.

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 to comply 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 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 understanding of the dynamics of a flow and its congestion control behaviour. Analysing observed packet sequence numbers can be used to help 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.

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.

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 operator’s ability to observe transport performance, and may 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). Although many network operators utilise transport information as a part of their operational practice, the network will not break because transport headers are encrypted, and this may require alternative tools may need to be developed and deployed.

3.3.1. Examples of measurements

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 may involve active injection of test traffic to complete a measurement. However, most operators do not have access to user equipment, and injection of test traffic may be associated with costs in running such tests (e.g., the implications of bandwidth 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 in operational networks to make the measurements.

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 to those of wired networks. A radio-based network (e.g., cellular mobile, enterprise WiFi, satellite access/backhaul, 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. Observing Headers to Implement Network Policy

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]). Traffic that cannot be classified, will typically receive a default treatment.

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.

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 may 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], the DSCP and ECN.
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 of 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 TCP/IP 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.


### 4.1. 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 observes these fields. An integrity check can not prevent in-network modification, but can avoid a receiving 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.
4.2. 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 (Section 4.1).

Examples of encrypting the payload include Transport Layer Security (TLS) over TCP [RFC5246] [RFC7525], Datagram TLS (DTLS) over UDP [RFC6347] [RFC7525], and TCPcrypt [I-D.ietf-tcpinc-tcpcrypt], which permits opportunistic encryption of the TCP transport payload.

4.3. Encrypting the Transport Header

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.

4.4. Authenticating Transport Information and Selectively Encrypting the Transport Header

A transport protocol design can encrypt selected header fields, while also choosing to authenticate fields in the 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 may 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.
4.5. 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.

5. Addition of Transport Information to Network-Layer Protocol Headers

Transport protocol information can be made visible in a network-layer header. This has the advantage that this information can then be observed by in-network devices. This has the advantage that a single header can support all transport protocols, but there may also be less desirable implications of separating the operation of the transport protocol from the measurement framework.

Some measurements may 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.

Another example of a network-layer approach is the IPv6 Performance and Diagnostic Metrics (PDM) Destination Option [RFC8250]. This allows a sender to optionally include a destination option that carries header fields that can be used to observe timestamps and packet sequence numbers. This information could be authenticated by receiving transport endpoints when the information is added at the sender and visible at the receiving endpoint, although methods to do this have not currently been proposed. This method needs to be explicitly enabled at the sender.

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
advantage to expose the information that is utilised by the protocol itself, and could manipulate this 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. 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).

When encryption conceals information in the transport header, it could be possible for an applications to provide summary data on performance and usage of the network. This data could be made available to other actors. However, this data needs to contain sufficient detail to understand (and possibly reconstruct the network traffic pattern for further testing) and to be correlated with the configuration of the network paths being measured.

Sharing information between actors needs also to consider the privacy of the user and the incentives for providing accurate and detailed information. 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. This becomes important when considering changes to transport protocols, changes in network infrastructure, or the emergence of new traffic patterns.
6.2. Characterising "Unknown" Network Traffic

The patterns and types of traffic that share Internet capacity changes with time as networked applications, usage patterns and protocols continue to evolve.

If "unknown" or "uncharacterised" traffic patterns form a small part of the traffic aggregate passing through a network device or segment of the network the path, the dynamics of the uncharacterised traffic may not have a significant collateral impact on the performance of other traffic that shares this network segment. Once the proportion of this traffic increases, the need to monitor the traffic and determine if appropriate safety measures need to be put in place.

Tracking the impact of new mechanisms and protocols requires traffic volume to be measured and new transport behaviours to be identified. This is especially true of protocols operating over a UDP substrate. The level and style of encryption needs to be considered in determining how this activity is performed. On a shorter timescale, information may also need to be collected to manage denial of service attacks against the infrastructure.

6.3. Accountability and Internet Transport Protocols

Information provided by tools observing transport headers can be used to classify traffic, and to limit the network capacity used by certain flows. Operators can potentially use this information to prioritise or de-prioritise certain flows or classes of flow, with potential implications for network neutrality, or to rate limit malicious or otherwise undesirable flows (e.g., for Distributed Denial of Service, DDoS, protection, or to ensure compliance with a traffic profile 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 is expected to 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 reduces the level of precision with which flows can be classified and conditioning mechanisms are applied (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 Research, Development and Deployment

The majority of present Internet applications use two well-known transport protocols: e.g., TCP and UDP. Although TCP represents the majority of current traffic, some important real-time applications use UDP, and much of this traffic utilises RTP format headers in the payload of the UDP datagram. Since these protocol headers have been fixed for decades, a range of tools and analysis methods have became common and well-understood. Over this period, the transport protocol headers have mostly changed slowly, and so also the need to develop tools track new versions of the protocol.

Looking ahead, there will be a need to update these protocols and to develop and deploy new transport mechanisms and protocols. There are both opportunities and also challenges to the design, evaluation and deployment of new transport protocol mechanisms.

Integrity checks can protect an endpoint from undetected modification of protocol fields by network devices, whereas encryption and obfuscation 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.

Hiding headers can limit the ability to measure and characterise traffic. Measurement data is increasingly being used to inform design decisions in networking research, during development of new mechanisms and protocols and in standardisation. 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). 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, DCTP, and methods proposed for L4S). The growth and diversity of applications and protocols using the Internet also continues to expand. For each new method or application it is desirable to build a body of data reflecting its behaviour under a wide range of deployment scenarios, traffic load, and interactions with other deployed/candidate methods.

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

The majority of present Internet applications use two well-known transport protocols: e.g., TCP and UDP. Although TCP represents the majority of current traffic, some important real-time applications have used 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. Over this period, the transport protocol headers have mostly changed slowly, and so also the need to develop tools track new versions of the protocol.

Confidentiality and strong integrity checks have properties that are being incorporated into new protocols and which have important benefits. The pace of development of transports using the WebRTC data channel and the rapid deployment of QUIC prototype transports can both be attributed to using a combination of UDP transport and confidentiality of the UDP 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 an operational issue 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 needs to be developed to catch-up with the changes. If the currently deployed tools and methods are no longer relevant and performance may not be correctly measured. 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 mis-classified and then receiving a default treatment by the network.

As a part of its design a new protocol specification therefore needs to weigh the benefits of ossifying common headers, versus the potential demerits of exposing specific information that could be observed along the network path to provide tools to manage new variants of protocols. Several scenarios to illustrate different ways this could evolve are provided below:

- One scenario is when transport protocols provide consistent information to the network by intentionally exposing a part of the transport header. The design fixes the format of this information between versions of the protocol. This ossification of the transport header allows an operator to establish tooling and procedures that enable it to provide consistent traffic management as the protocol evolves. In contrast to TCP (where all protocol information is exposed), evolution of the transport is facilitated by providing cryptographic integrity checks of the transport header fields (preventing undetected middlebox changes) and encryption of other protocol information (preventing observation within the network, or incentivising the use of the exposed information, rather than inferring information from other characteristics of the flow traffic). The exposed transport information can be used by operators to provide troubleshooting, measurement and any necessary functions appropriate to the class of traffic (priority, retransmission, reordering, circuit breakers, etc).

- An alternative scenario adopts different design goals, with a different outcome. A protocol that encrypts all header information forces network operators to act independently from
apps/transport developments to provide the transport information they need. A range of approaches may proliferate, as in current networks, operators can add a shim header to each packet as a flow as it crosses the network; other operators/managers could develop heuristics and pattern recognition to derive information that classifies flows and estimates quality metrics for the service being used; some could decide to rate-limit or block traffic until new tooling is in place. In many cases, the derived information can be used by operators to provide necessary functions appropriate to the class of traffic (priority, retransmission, reordering, circuit breakers, etc). Troubleshooting, and measurement becomes more difficult, and more diverse. This could require additional information beyond that visible in the packet header and when this information is used to inform decisions by on-path devices it can lead to dependency on other characteristics of the flow. In some cases, operators might need access to keying information to interpret encrypted data that they observe. Some use cases could demand use of transports that do not use encryption.

The outcome could have significant implications on the way the Internet architecture develops. It exposes a risk that significant actors (e.g., developers and transport designers) achieve more control of the way in which the Internet architecture develops. In particular, there is a possibility that designs could evolve to significantly benefit of customers for a specific vendor, and that communities with very different network, applications or platforms could then suffer at the expense of benefits to their vendors own customer base. In such a scenario, there could be no incentive to support other applications/products or to work in other networks leading to reduced access for new approaches.

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 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 imply 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 "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 headers 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 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, Jana Iyengar, Mirja Kuehlewind, Kathleen Moriarty, Al Morton, Chris Seal, Joe Touch, Brian Trammell, 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-ippm-ioam-data]
[I-D.ietf-quic-transport]

[I-D.ietf-taps-transport-security]

[I-D.ietf-tcpinc-tcpcrypt]

[I-D.ietf-tsvwg-l4s-arch]

[I-D.thomson-quic-grease]

[I-D.trammell-plus-abstract-mech]


Fairhurst & Perkins Expires February 28, 2019 [Page 33]


Appendix A. Revision information

-00 This is an individual draft for the IETF community.

-01 This draft was a result of walking away from the text for a few days and then reorganising the content.

-02 This draft fixes textual errors.

-03 This draft follows feedback from people reading this draft.

-04 This adds an additional contributor and includes significant reworking to ready this for review by the wider IETF community Colin Perkins joined the author list.

Comments from the community are welcome on the text and recommendations.

-05 Corrections received and helpful inputs from Mohamed Boucadair.

-06 Updated following comments from Stephen Farrell, and feedback via email. Added a draft conclusion section to sketch some strawman scenarios that could emerge.

-07 Updated following comments from Al Morton, Chris Seal, and other feedback via email.

-08 Updated to address comments sent to the TSVWG mailing list by Kathleen Moriarty (on 08/05/2018 and 17/05/2018), Joe Touch on 11/05/2018, and Spencer Dawkins.

-09 Updated security considerations.

-10 Updated references, split the Introduction, and added a paragraph giving some examples of why ossification has been an issue.

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/
Abstract

We detail the implementation of a network scheduler that aims at isolating time constrained and elastic traffic flows from best-effort traffic. This scheduler inherits from the priority scheduler (PS) but dynamically changes the priority of one or several queues. Usual implementations of rate scheduler schemes (such as WRR, DRR, ...) do not allow to efficiently guarantee the capacity dedicated to both AF and BE classes as they mostly provide soft bounds. This means excessive margin is used to ensure the capacity requested and this impacts the number of additional users that could be accepted in the network. To cope with this issue, this memo presents a credit based scheduler mechanism called Priority Switching Scheduler (PSS) that allows a more predictable output rate per traffic class.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 1, 2018.
1. Introduction

1.1. Context and Motivation

To share the capacity offered by a link, many fair schedulers have been developed, such as Weighted Fair Queuing, Weighted Round Robin or Deficit Round Robin. However, with these well-known solutions, the output rate of a given queue depends on the amount of traffic crossing other queues. Our proposal aims at reducing the uncertainty of the output rate of selected queues, we call them in the following controlled queues. Additionally, compared to previous cited schemes, this solution is simpler to implement mainly because it does not require a virtual clock, and more flexible thanks to the wide possibilities offered by the setting of different priorities.
1.2. 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].

1.3. Priority Switching Scheduler in a nutshell

As illustrated in Figure 1, the principle of PSS is based on the use of credit counters (detailed in the following) to change the priority of one or several queues. The idea follows a proposal made by the TSN Task group named Burst Limiting Shaper [BLS]. For each controlled queue $i$, each priority denoted $p[i]$, changes between two values denoted $p_{low}[i]$ and $p_{high}[i]$, depending on the associated credit counter, i.e., credit[i]. Then a Priority Scheduler is used for the dequeuing process, e.g., among the queues with available traffic, the first packet of the queue with the highest priority is dequeued.

The main idea is that changing the priorities adds fairness to the Priority Scheduler. Depending on the credit counter parameters, the amount of capacity available to a controlled queue is bounded between a minimum and a maximum value. Consequently, good parameterization is very important to prevent starvation of lower priority queues.

Figure 1: PSS in a nutshell
The service obtained for the controlled queue with the switching priority is more predictable and corresponds to the minimum between a desired capacity and the residual capacity left by higher priorities. The impact of the input traffic sporadicity from higher classes is thus transferred to non-active PSS queues with a lower priority.

Finally, PSS offers much flexibility as both i) controlled queues with a guaranteed capacity (when two priorities are set), ii) and queues scheduled with a simple Priority Scheduler (when only one priority is set) can jointly be enabled.

2. Priority Switching Scheduler

2.1. Specification

The PSS algorithm defines for each queue q a low priority, p_low[q], and a high priority, p_high[q]. Each PSS controlled queue q with p_high[q] < p_low[q] is associated to a credit counter credit[q] which manages the priority switching. Each credit counter is defined by:

- a minimum level: 0;
- a maximum level: LMs[q];
- a resume level: LRs[q]
- a reserved capacity: BWs[q]
- an idle slope: Iidle[q] = C * BWs[q];
- a sending slope: Isend[q] = C - Iidle[q];

The available capacity is mostly impacted by the guaranteed capacity BWs[q]. Hence BWs[q] should be set to the desired capacity plus a margin taking into account the additional packet due to non-preemption as explained below:

The value of LMs[q] can negatively impact on the guaranteed available capacity. The maximum level determines the size of the maximum sending windows, i.e., the maximum uninterrupted transmission time of the controlled queue packets before a priority switching. The impact of the non-preemption is as a function of the value of LMs[q]. The smaller the LMs[q], the larger the impact of the non-preemption is. For example, if the number of packets varies between 4 and 5, the variation of the output traffic is around 25% (i.e., going from 4 to 5 corresponds to a 25% increase). If the number of packets sent varies between 50 and 51, the variation of the output traffic is around 2%.
The credit allows to keep track of the packet transmissions. However, there are two cases keeping track of the transmission raises an issue: when the credit is saturated at LMs[q] or at 0. In both cases, packets are transmitted without gained or consumed credit. Nevertheless, the resume level can be used to decrease the times when the credit is saturated at 0. If the resume level is 0, then as soon as the credit reaches 0, the priority is switched and the credit saturates at 0 due to the non-preemption of the current packet. On the contrary, if LRs[q]>0, then during the transmission of the non-preempted packet, the credit keeps on decreasing before reaching 0 as illustrated in Figure 2.

Hence, the proposed value for LRs[q] is LRs[q] = Lmax(MC(q)) \times BWs[q], with MC(q) the queues such as k in MC(q) \rightarrow p_{low}[q] > (p_{low}[k] or p_{high}[k]) > p_{high}[q], and Lmax(qs) the maximum size of the queues qs. With this value, there is no credit saturation at 0 due to non-preemption.

Finally, we propose to use the following parameters of a controlled queue q:

- BWs[q] = desired_BWs[q] + 1/(N-1)
- LMs[q] = (N-1) \times Lmax(q) \times (1 - BWs[q])
- LRs[q] = Lmax(MC(q)) \times BWs[q]

with N the maximum number of packet of queue q set uninterrupted (taking into account the non-preemption) and desired_BWs[q] the percentage of desired available capacity.

A similar parameter setting is described in [Globecom17], to transform WRR parameter into PSS parameters, in the specific case of 3-classes DiffServ architecture.

The priority change depends on the credit counter as follows:

- initially, the credit counter starts at 0;
- the change of priority p[q] of queue q occurs in two cases:
  * if p[q] = p_{high}[q] and the credit reaches LMs[q];
  * if p[q] = p_{low}[q] and credit reaches LRs[q];
- when a packet of queue q is transmitted, the credit increases with a rate Isend[q], else the credit decreases with a rate Iidle[q];
Internet-Draft        Priority Switching Scheduler              May 2018

- when the credit reaches LMs[q], it remains at this level until the
  end of the transmission of the current packet;

- when the credit reaches 0, it remains at this level until the
  start of the transmission of a queue q packet.

Figure 2 and Figure 3 show two examples of credit and priority
changes of a given queue q. First, in Figure 2, we show an example
when the controlled queue q sends its traffic continuously until the
priority change. Then other traffic is also sent uninterruptedly
until the priority changes back. In Figure 3, we propose a more
complex behaviour. First, this figure shows when a packet with a
priority higher than p_high[q] is available, this packet is sent
before the traffic of class q. Secondly, when no traffic with a
priority lower than p_low[q] is available, then traffic of queue q
can be sent. This highlight the non-blocking nature of PSS and that
p[q] = p_high[q] (resp. p[q] = p_low[q]) does not necessarily mean
that traffic of queue q is being sent (resp. not being sent).

\begin{figure}
\centering
\begin{tabular}{c|c|c|c}
\hline
credit[q] & p_high[q] & p_low[q] & p_high[q] \\
LMs[q] & - - - - - - - - - - & - - - - - - - & - - - - - - - \\
      & + & + & + \\
Isend & [q] & + & [q] \\
      & + & + & + \\
      & + & + & + \\
LRs[q] & + & + & + \\
          & + & + & + \\
0 & - - - - - & - - - - - & - - - - - & - - - - - > \\
\hline
\end{tabular}
\caption{First example of queue q credit and priority behaviors}
\end{figure}
Finally, for the dequeuing process, a Priority Scheduler selects the appropriate packet using the current p[q] values, e.g., among the queues with available traffic, the first packet of the queue with the highest priority is dequeued.

2.2. Implementation

The new dequeuing algorithm is presented in the PSS Algorithm. The credit of each queue q, denoted credit[q], and the dequeuing timer denoted timerDQ[q] are initialized to zero. The initial priority is set to the high value p_high[q]. First, for each queue with p_high[q] > p_low[q], the difference between the current time and the time stored in timerDQ[q], is computed (lines 2 and 3). The duration \( dtime[q] \) represents the time elapsed since the last credit update, during which no packet of the controlled queue q was sent, we call this the idle time. Then, if \( dtime[q] > 0 \), the credit is updated by removing the credit gained during the idle time that just occurred (lines 4 and 5). Next, timerDQ[q] is set to the current time to keep track of the time the credit is last updated (line 6). If the credit reaches LRs[q], the priority changes to its high value (lines 7 and 8). Then, with the updated priorities, the priority scheduler performs as usual: each queue is checked for dequeuing, highest priority first (lines 12 and 13). When a queue q is selected with p_high[q] < p_low[q], the credit expected to be consumed is added to credit[q] variable (line 16). The time taken for the packet to be dequeued is added to the variable timerDQ[q] (lines 13 and 14) so the transmission time of the packet will not be taken into account in the idle time \( dtime[q] \) (line 2). If the credit reaches LMs[q], the
priority changes to its low value (lines 18 and 19). Finally, the packet is dequeued (line 22).

Inputs: credits, timerDQs, C, LMs, LRs, BWs, p_highs, p_lows

```plaintext
1   currentTime = getCurrentTime()
2   for each queue q with p_high[q] < p_low[q] do:
3      dtime[q] = currentTime-timerDQ[q]
4      if dtime[q]>0 then:
5         credit[q] = max(credit[q]-dtime[q].C.BWs[q],0)
6         dtime[q] = currentTime
7         if credit[q]<LRs[q] and p[q] = p_low[q] then:
8            p[q] = p_high[q]
9         end if
10      end if
11   end for
12   for each priority level pl, highest first do:
13      if length(queue(pl))>0 then:
14         q=queue(pl)
15         if p_high[q] < p_low[q] then:
16            credit[q] = min(LMs[q],
17                              credit[q]+size(head(q)).(1-BWs[q]))
18            timerDQ[q] = currentTime+size(head(q))/C
19            if credit >= LMs[q] and p[q] = p_high[q] then:
20               p[q] = p_low[q]
21         end if
22         dequeue(head(q))
23      break
24     end if
25   end for
```

Figure 4: PSS algorithm

PSS algorithm also implements the following functions:

- `getCurrentTime()` uses a timer to return the current time;
- `queue(pl)` returns the queue associated to priority pl;
- `head(q)` returns the first packet of queue q;
- `size(f)` returns the size of packet f;
- `dequeue(f)` activates the dequeuing event of packet f.
3. Usecase: benefit of using FSS in a Diffserv core network

3.1. Motivation

The DiffServ architecture defined in [RFC4594] and [RFC2475] proposes a scalable mean to deliver IP quality of service (QoS) based on handling traffic aggregates. This architecture follows the philosophy that complexity should be delegated to the network edges while simple functionalities should be located in the core network. Thus, core devices only perform differentiated aggregate treatments based on the marking set by edge devices.

Keeping aside policing mechanisms that might enable edge devices in this architecture, a DiffServ stateless core network is often used to differentiate time-constrained UDP traffic (e.g. VoIP or VoD) and TCP bulk data transfer from all the remaining best-effort (BE) traffic called default traffic (DF). The Expedited Forwarding (EF) class is used to carry UDP traffic coming from time-constrained applications (VoIP, Command/Control, ...); the Assured Forwarding (AF) class deals with elastic traffic as defined in [RFC4594] (data transfer, updating process, ...) while all other remaining traffic is classified inside the default (DF) best-effort class.

The first and best service is provided to EF as the priority scheduler attributes the highest priority to this class. The second service is called assured service and is built on top of the AF class where elastic traffic such as TCP traffic, is intended to achieve a minimum level of throughput. Usually, the minimum assured throughput is given according to a negotiated profile with the client. The throughput increases as long as there are available resources and decreases when congestion occurs. As a matter of fact, a simple priority scheduler is insufficient to implement the AF service. TCP traffic increases until reaching the capacity of the bottleneck due to its opportunistic nature of fetching the full remaining capacity. In particular, this behaviour could lead to starve the DF class.

To prevent a starvation and ensure to both DF and AF a minimum service rate, the router architecture proposed in [RFC5865] uses a rate scheduler between AF and DF classes to share the residual capacity left by the EF class. Nevertheless, one drawback of using a rate scheduler is the high impact of EF traffic on AF and DF. Indeed, the residual capacity shared by AF and DF classes is directly impacted by the EF traffic variation. As a consequence, the AF and DF class services are difficult to predict in terms of available capacity and latency.

To overcome these limitations and make AF service more predictable, we propose here to use the newly defined Priority Switching Scheduler...
(PSS). Figure 5 shows an example of the Data Plane Priority core network router presented in [RFC5865] modified with a PSS. The EF queues have the highest priorities to offer the best service to real-time traffic. The priority changes set the AF priorities either higher (3,4) or lower (6,7) than CS0 (5), leading to capacity sharing. Another example with only 3 queues is described in [Globecom17]. Thank to the increase predictability, for the same minimum guaranteed rate, the PSS reserves a lower percentage of the capacity than a rate scheduler. This leaves more remaining capacity that can be guaranteed to other users.

```
<table>
<thead>
<tr>
<th>queues</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Admitted EF</td>
<td>p[AEF]= 1</td>
</tr>
<tr>
<td>Unadmitted EF</td>
<td>p[UEF]= 2</td>
</tr>
<tr>
<td>AF1</td>
<td>p_high[AF1]=3 and p_low[AF1]= 6</td>
</tr>
<tr>
<td>AF2</td>
<td>p_high[AF2]=4 and p_low[AF2]= 7</td>
</tr>
<tr>
<td>CS0</td>
<td>p[CS0]= 5</td>
</tr>
</tbody>
</table>
```

Figure 5: PSS applied to Data Plane Priority (we borrow the syntax from RCF5865)

3.2. New service offered

The new service we seek to obtain is:

- for EF, the full capacity of the output link;
- for AF the minimum between a desired capacity and the residual capacity left by EF;
- for DF (CS0), the residual capacity left by EF and AF.

As a result, the AF class has a more predictable available capacity, while the unpredictability is reported on the DF class. With good parametrization, both classes also have a minimum rate ensured. Parameterization and simulations results concerning the use of a similar scheme for core network scheduling are available in [Globecom17].
4. Security Considerations

There are no specific security exposure with PSS that would extend those inherent in default FIFO queuing or in static priority scheduling systems. However, following the DiffServ usecase proposed in this memo and in particular the illustration of the integration of PSS as a possible implementation of the architecture proposed in [RFC5865], most of the security considerations from [RFC5865] and more generally from the differentiated services architecture described in [RFC2475] still hold.

5. Acknowledgements

This document was the result of collaboration and discussion among a large number of people. In particular the authors wish to thank Nicolas Kuhn and David Black for reviewing this draft. At last but not least, a very special thanks to Fred Baker for his help.

6. References

6.1. Normative References


6.2. Informative References


Authors’ Addresses

Fred Baker
Santa Barbara, California  93117
USA

Email: FredBaker.IETF@gmail.com

Anais Finzi
ISAE-SUPAERO
10 Avenue Edouard Belin
Toulouse  31400
France

Phone: 0033561338735
Email: anais.finzi@isae-supraero.fr

Fabrice Frances
ISAE-SUPAERO
10 Avenue Edouard Belin
Toulouse  31400
France

Email: fabrice.frances@isae-suprero.fr

Emmanuel Lochin
ISAE-SUPAERO
10 Avenue Edouard Belin
Toulouse  31400
France

Email: emmanuel.lochin@isae-supraero.fr
A New Congestion Control in Bandwidth Guaranteed Network

draft-han-tsvwg-cc-00

Abstract

In bandwidth guaranteed networks, network resources are reserved before a TCP session starts transmitting data. This draft proposes a new TCP congestion control algorithm used in bandwidth guaranteed networks. It is an extension to the current TCP standards.

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 4, 2018.

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

The original IP protocol suite was designed to support best-effort data transmission. With the development of the Internet, congestion became a real problem. To avoid congestion in the Internet, TCP uses congestion-avoidance algorithms to keep hosts from pumping too much traffic into the network. Over the past 40 years there have been various algorithms and optimizations proposed to solve this problem, including TCP-RENO [RFC5681], TCP-NewReno [RFC6582] [RFC6675], TCP-Cubic [RFC8312] and BBR [I-D.cardwell-iccrg-bbr-congestion-control] etc.

In bandwidth guaranteed networks, network resources are reserved before transmitting data. This draft proposes a new congestion control algorithm that should be used in bandwidth guaranteed networks to improve TCP throughput. The following is a list of key differences between this new algorithm and classic TCP congestion control [RFC5681]:

- It doesn’t have a slow start, after a TCP session is successfully initiated its congestion window (cwnd) jumps to CIR and the host is allowed to transmit data. This is based on the assumption that network resources have been reserved in bandwidth guaranteed networks.

Han, et al. Expires September 4, 2018 [Page 2]
During congestion avoidance, cwnd stays between CIR (Committed Information Rate) and PIR (Peak Information Rate). If there is no packet loss due to congestion, cwnd has a flat top rate as PIR.

OAM is used together with duplicate ACKs to detect whether a packet loss is due to congestion or random failure.

This draft is organized as follows. Section 2 defines terminologies used in this draft. Section 3 provides background information for Bandwidth Guaranteed Networks. Section 4 explains the details of the new congestion control algorithm.

2. Terminology and Notation

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].

Some of the following terms are defined the same as [RFC5681], and they are copied here for readability.

FULL-SIZED SEGMENT: A segment that contains the maximum number of data bytes permitted (i.e., a segment containing SMSS bytes of data).

RECEIVER WINDOW (rwnd): The most recently advertised receiver window.

CONGESTION WINDOW (cwnd): A TCP state variable that limits the amount of data a TCP can send. At any given time, a TCP MUST NOT send data with a sequence number higher than the sum of the highest acknowledged sequence number and the minimum of cwnd and rwnd.

Sender Maximum Segment Size (SMSS): The SMSS is the size of the largest segment that the sender can transmit. This value can be based on the maximum transmission unit of the network, the path MTU discovery [RFC1191, RFC4821] algorithm, RMSS (see next item), or other factors. The size does not include the TCP/IP headers and options.

RECEIVER MAXIMUM SEGMENT SIZE (RMSS): The RMSS is the size of the largest segment the receiver is willing to accept. This is the value specified in the MSS option sent by the receiver during connection startup. Or, if the MSS option is not used, it is 536 bytes [RFC1122]. The size does not include the TCP/IP headers and options.
INITIAL WINDOW (IW): The initial window is the size of the sender’s congestion window after the three-way handshake is completed.

RESTART WINDOW (RW): The restart window is the size of the congestion window after a TCP restarts transmission after an idle period.

ssthresh: Slow Start Threshold.

OAM: Operations, Administrations, and Maintenance.

RTT: Round-Trip Time.

CIR: Committed Information Rate.

PIR: Peak Information Rate.

3. Bandwidth Guaranteed Network

With the development of new applications, such as AR/VR, the network is required to provide bandwidth guaranteed services. There have been various solutions, including out-of-band signaling protocols such as RSVP [RFC2205] and NSIS [RFC4080], and in-band-signaling as proposed in [I-D.han-6man-in-band-signaling-for-transport-qos]. The common objective of all these solutions is to have network resources/bandwidth reserved before data is transmitted. The details of how the resource is reserved are out of the scope of this draft, however it is assumed that in bandwidth guaranteed networks there have been network resources (bandwidths, queues etc.) dedicated to the TCP flows, and data is guaranteed at CIR rate. When data rate is between CIR and PIR shared resources are used, and traffic above CIR rate is not guaranteed. No traffic above PIR rate will be allowed to enter the network.

The proposed congestion control also requires that OAM (Operations, administration and management) is used to constantly report on the network condition parameters. Before a TCP session is started, important network parameters need to be detected by OAM, such as number of hops, Round Trip Time (RTT). This might be done through setting up a measuring TCP connection. The measuring TCP connection does not have user data, and it is only used to measure the key network parameters. As the network status is constantly changing, after a TCP session is established, these parameters need to be updated. This requires a sender to periodically or consistently embed TCP data packet with OAM [I-D.han-6man-in-band-signaling-for-transport-qos] [I-D.ietf-ippm-ioam-data] to detect current buffer depth, RTT etc.
It is important that OAM needs to be able to detect if any device’s buffer depth has exceeded the pre-configured threshold, as this is an indication of potential congestion and packet drop. When this happens, OAM should send a possible congestion alarm to the TCP sender. In case the retransmit timer expires on this TCP sender, if a possible congestion alarm has been received it means a packet is dropped due to congestion. Otherwise it is possible that this packet drop might due to some physical failure. The OAM details are out of the scope of this draft. Please refer to other related drafts.

In summary, in bandwidth guaranteed networks resources are reserved before transmitting data, and OAM is used to get network statistics. The new congestion control proposed in this draft is to be used in this kind of bandwidth guaranteed networks.

4. New Congestion Control

[RFC5681] defines a set of TCP congestion algorithms: slow start, congestion avoidance, fast retransmit and fast recovery. The proposed congestion control in this draft is an extension to RFC 5681, and it only differs in the congestion control algorithm on the sender side.

4.1. Receiver Advertised Window Size

Receiver’s advertised window (rwnd) is a receiver-side limit on the amount of outstanding data, so a sender should not send data more than this window size. It is calculated as the following:

\[ rwnd = \text{AdvertisedWND} = \text{MaxRcvBuffer} - (\text{LastByteRcvd} - \text{LastByteRead}) \]

4.2. MinBandwidthWND and MaxBandwidthWND

Same as [RFC5681], on the sender side, the congestion window (cwnd) is the sender-side limit on the amount of data that the sender can transmit before receiving an acknowledgement (ACK). Considering both the sender and the receiver side, the effective sending window is always the minimum of cwnd and rwnd:

\[ \text{EffectiveWND} = \min(cwnd, rwnd) \]

A TCP sender MUST NOT send data more than the minimum of cwnd and rwnd.

Slow-start is commonly used in TCP at the beginning of a transfer or after a loss repair as the network conditions are unknown, hence this slow probing is necessary to determine the available network capacity in order to avoid inappropriately sending large burst of data into
the network and cause congestion. A detailed discussion about initial window setting is provided in [RFC3390].

RTT is the time taken to send a packet to the destination plus receiving a response packet (ACK). Since the network status is constantly changing, RTT also varies. [RFC6298] specifies how RTT should be sampled and updated. In this new algorithm RTT is updated using the following formula:

\[
RTT = a \times \text{old RTT} + (1-a) \times \text{new RTT} \quad (0 < a < 1) \quad (1)
\]

The initial RTT can be achieved using a measure TCP connection, or configured based on historical data.

In bandwidth guaranteed network since resources are already allocated and the network status is known through OAM [I-D.han-6man-in-band-signaling-for-transport-qos], it is safe to remove slow-start and allow a host to start sending traffic at the rate of CIR after the TCP session is established.

There are two important window sizes, the MinBandwidthWND and the MaxBandwidthWND are calculated as below:

\[
\text{MinBandwidthWND} = \text{CIR} \times \frac{\text{RTT}}{\text{MSS}} \quad (2)
\]

\[
\text{MaxBandwidthWND} = \text{PIR} \times \frac{\text{RTT}}{\text{MSS}} \quad (3)
\]

In bandwidth guaranteed networks, after a TCP session is established, the sender can start transmitting data at an initial window size, which is equal to MinBandwidthWND:

\[
\text{cwnd} = \text{MinBandwidthWND} \\
\text{IW} = \min (\text{cwnd}, \text{rwnd})
\]

If the receiver window (rwnd) is not a limiting factor, the sender will start sending data at CIR rate. This is a key difference from the classic TCP slow-start, which usually starts from sending one or two packets [RFC5681].

4.3. Congestion Avoidance

In TCP-Reno, a TCP enters congestion avoidance mode after slow-start. In bandwidth guaranteed networks, there is no slow-start, so a TCP enters congestion avoidance mode right after the initial start.

During congestion avoidance, for approximately per round-trip time when a valid ACK packet is received, cwnd is increased by one until it reaches MaxBandwidthWND.
If (cwnd < MaxBandwidthWND) {
    cwnd += 1;
} else {
    cwnd = MaxBandwidthWND;
}

Once the cwnd reaches MaxBandwidthWND, it stays constant at MaxBandwidthWND until packet loss is detected. This is another major difference from [RFC5681]. In [RFC5681] congestion avoidance period, the cwnd keeps increasing until a TCP sender detects segment loss. However, in this new congestion control algorithm, the cwnd stays constant at MaxBandwidthWND until there is packet loss detected.

This means a TCP sender is never allowed to send data at a rate larger than PIR, and it’s different from TCP Reno.

4.4. Fast Retransmit and Fast Recovery

Same as defined [RFC5681], a TCP receiver SHOULD send an immediate duplicate ACK when an out-of-order segment arrives. The TCP sender detects and repair loss based on incoming duplicate ACKs. If 3 duplicate ACKs are received, the sender uses it as an indication that a segment has been lost, and will perform a retransmission of the lost segment.

In TCP-Reno [RFC5681], after the fast retransmit of what appears to be the lost segment, fast recovery is used to continue to transmit new segments at a reduced rate ssthresh.

In the new congestion control algorithm, upon receiving duplicate ACKs the fast retransmit and fast recovery follow the below rules:

- When a sender receives the first and second duplicate ACKs, same as [RFC5681], the cwnd is not changed, and the sender continues to send traffic.

- When a sender receives the third duplicated ACK, if the retransmission timer has not expired and a previous OAM congestion alarm has been received it is likely a segment is lost due to congestion. The sender will perform a retransmission of the lost segment, and the cwnd is set to be MinBandwidthWND.

- When a sender receives the third duplicated ACK, but no previous OAM congestion alarm has been received, then it is considered that a segment is lost due to random failure not congestion. In this case the cwnd is not changed.
Compared to [RFC5681], where in case of network congestion the new cwnd is set to be ssthresh, which is usually half of the old cwnd. In this new congestion control, in case there is a segment loss detected as described above, the new cwnd is set to be MinBandwidthWND as in equation (2).

4.5. Timeout

If a retransmission timer [RFC6298] in a TCP sender expires, in bandwidth guaranteed networks no matter duplicate ACK received or not, this most likely indicates a physical failure.

In this case, the cwnd is set to be one, and the TCP sender will retransmit the lost segment. This packet also services the function of probing network status. If there is really a network failure, no ACK will be received and the retransmission timer will expire again. Upon receiving an expected ACK after the retransmission, it means the network has recovered, and the cwnd will be set to be MinBandwidthWND as in equation (2).

4.6. Idle Recovery

It is defined in [RFC5681] that a TCP session should use slow start to restart transmission after a long idle period more than one retransmission timeout, and the RW (Restart Window) is the minimum of IW and cwnd.

In this proposal, the same rule is still followed. However due to the fact that there is no slow start needed in bandwidth guaranteed networks, and the IW in this new congestion control is set to be MinBandwidthWND, a TCP sender can start transmitting data at CIR rate after a long idle.

5. IANA Considerations

NA.

6. Security Considerations

This proposal makes no change to the underlying security of TCP. More information about TCP security concerns can be found in [RFC5681].

7. References
7.1. Normative References


7.2. Informative References


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

[I-D.han-6man-in-band-signaling-for-transport-qos]

[I-D.ietf-ippm-ioam-data]
Acknowledgments

The authors wish to thank xxxx for their helpful comments and suggestions.

Authors’ Addresses

Lin Han
Huawei
2330 Central Expressway
Santa Clara  CA  95050
USA

EMail: lin.han@huawei.com

Yingzhen Qu
Huawei
2330 Central Expressway
Santa Clara  CA  95050
USA

EMail: yingzhen.qu@huawei.com

Thomas Nadeau
Lucid Vision
Hampton  NH  03842
USA

EMail: tnadeau@lucidvision.com
Firewall and Service Tickets
draft-herbert-fast-03

Abstract

This document describes the Firewalls and Service Tickets protocol. A ticket is data that accompanies a packet and indicates a granted right to traverse a network or a request for network service to be applied. Applications request tickets from a local agent in the network and attach issued tickets to packets. Firewall tickets are issued to grant packets the right to traverse a network; service tickets indicate the desired service to be applied to a packets. A single ticket may provide both firewall and service ticket information. Tickets are sent in IPv6 Hop-by-Hop options.

Status of this Memo

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

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

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

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

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

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

Table of Contents

1 Introduction ........................................ 4
2 Motivation ........................................ 4
  2.1 Current mechanisms ............................... 4
    2.1.1 Stateful firewalls and proxies .............. 4
    2.1.2 QoS signaling ................................ 5
    2.1.3 Deep packet inspection ....................... 5
  2.2 Proposals for applications to signal the network .... 5
    2.2.1 SPUD/PLUS .................................. 5
    2.2.2 Path aware networking ....................... 7
  2.3 Emerging use cases .............................. 7
3 Architecture ....................................... 8
  3.1 Example packet flow .............................. 9
  3.2 Requirements ................................... 10
4 Packet format ..................................... 11
  4.1 Option format .................................. 11
  4.2 Option types ................................... 12
  4.3 Ticket format .................................. 12
5 Operation .......................................... 13
  5.1 Ticket types and ordering ...................... 13
  5.2 Origin application processing ................. 14
    5.2.1 Ticket requests ............................. 14
    5.2.2 Ticket identification ....................... 14
    5.2.3 Ticket use ................................ 15
    5.2.4 Ticket agent delegation .................... 15
  5.3 Origin network processing ...................... 15
  5.4 Peer host processing ............................ 16
  5.5 Processing reflected tickets .................. 16
    5.5.1 Network processing ......................... 17
    5.5.2 Host processing ............................. 17
  5.6 Handling dropped extension headers ............ 17
    5.6.1 Mitigation for dropped extension headers .... 17
    5.6.2 Fallback for dropped extension headers ........ 18
6 Implementation considerations ........................................... 19
   6.1 Origin applications ............................................... 19
   6.2 Ticket reflection ................................................. 19
7 Security Considerations ............................................... 19
8 IANA Considerations .................................................. 20
9 References .............................................................. 20
   9.1 Normative References ............................................. 20
   9.2 Informative References .......................................... 21
Author's Address .......................................................... 21
1 Introduction

Firewall and Service Tickets (FAST) is a technique to allow an application to signal to the network requests for admission and services for packets. A ticket is data that is attached to a packet by the source node, and it is then inspected and validated by certain intermediate nodes in a network. Tickets express a grant or right for packets to traverse a network or have services applied to them.

An application requests tickets for admission or services from a ticket agent in their local network. The agent issues tickets to the application which in turn attaches these to its packets. In the forwarding path, intermediate network nodes may interpret tickets and apply requested services on packets.

Tickets are validated for authenticity by the network and contain an expiration time so that they cannot be easily forged. Tickets do not have a global interpretation, they can only be interpreted within the network or local domain ([LIMDOM]) that issues them. In order to apply services to inbound packets for a communication, remote peers reflect received tickets in packets they send without interpreting them. Tickets are stateless within the network, however they can be used to attain per flow semantics. Firewall and service tickets are non-transferable and revocable.

Tickets are coded in IPv6 Hop-by-Hop options.

2 Motivation

This section presents the motivation for Firewall and Service Tickets.

2.1 Current mechanisms

Current solutions for controlling admission to the network and requesting services are mostly ad hoc and architecturally limiting.

2.1.1 Stateful firewalls and proxies

Stateful firewalls and proxies are the predominantly deployed techniques to control access to a network on a per flow basis. While they provide some benefits of security, they break the end-to-end model and have otherwise restricted the Internet in several ways:

- They require parsing over transport layer headers in the fast path of forwarding.
- They are limited to work only with a handful of protocols and
protocol features thereby ossifying protocols.

- They break the ability to use multi-homing and multi-path. All packets for a flow must traverse a specific network device in both directions of a communication.

- They can break end to end security. NAT for instance breaks the TCP authentication option.

- They have created single points of failure and become network bottlenecks.

2.1.2 QoS signaling

In the current Internet, there is little coordination between hosts and the network to provide services based on characteristics of the application. Differentiated services provides an IP layer means to classify and manage traffic, however it is lacking in richness of expression and lacks a ubiquitous interface that allows applications to request service with any granularity. Without additional state, there is no means for the network infrastructure to validate that a third party application has requested QoS that adheres to network policies.

2.1.3 Deep packet inspection

Some network devices perform Deep Packet Inspection (DPI) into the application data to determine whether to admit packets or what services to apply. For instance, HTTP is commonly parsed to determine URL, content type, and other application level information the network is interested in. DPI can only be effective with the application layer protocols that a device is programmed to parse. More importantly, application level DPI is effectively obsoleted in the network due the pervasive use of TLS. TLS interception and SSL inspection, whereby an intermediate node implements a proxy that decrypts a TLS session and re-encrypts, is considered a security vulnerability [TLSCERT].

2.2 Proposals for applications to signal the network

This section surveys some proposals to address the need for applications to signal the network.

2.2.1 SPUD/PLUS

SPUD (Session Protocol Underneath Datagrams) [SPUD] and its successor PLUS (Path Layer UDP Substrate) [PLUS] proposed a UDP based protocol to allow applications to signal a rich set of characteristics and
service requirements to the network.

SPUD had a number of drawbacks:

- SPUD is based on a specific protocol used over UDP. This requires applications to change to use a new protocol. In particular SPUD is incompatible with TCP which is the predominant transport protocol on the Internet.

- SPUD requires that intermediate nodes parse and process UDP payloads. Since UDP port numbers do not have global meaning [RFC7605] there is the possibility of misinterpretation and of silent data corruption if intermediate nodes modify UDP payloads. SPUD attempts to mitigate this issue with the use of magic numbers, however that can only ever be probabilistically correct.

- SPUD included stateful flow tracking in the network. This problematic because:
  - Not all communications have well defined connection semantics. For instance, a unidirectional data stream has no connection semantics at all.
  - Stateful network devices breaks multi-homing and multi-path; they assume that all packets of a flow in both directions are seen by the node doing tracking flow state. Stateful firewalls, for instance, require all packets for a flow to always go through the same device in both directions. This disallows flexibility and optimized traffic flow that a multi-homed network affords.
  - Maintaining per flow state in the network is an obvious scaling problem.
  - Keepalives to maintain a network state in a device, such as those sent to prevent a NAT state from being evicted, carry no useful information to the end user and in large numbers can become a source of congestion.

- The meta data information in SPUD would have global definition. This problematic because:
  - Application specific information could be leaked to unknown and untrusted parties.
  - Establishing a specification on what data should be conveyed in SPUD will be difficult. Different service providers may want different pieces of information, applications may also
have different ideas about what information is safe to make visible.

2.2.2 Path aware networking

Path aware networking (PAN) [PAN] is an IRTF effort to allow applications to select paths through the Internet for their traffic. The idea is that in choosing different paths an application can select from the different characteristics that are associated with different paths. Path aware networking requires a means to express the various paths and their associated characteristics to applications. In the data path, a method to express desired path for a packet is needed—this presumably could be by a mechanism such as segment routing.

PAN and FAST have similar characteristics, particularly with respect to the need for applications and networks to communicate about network path characteristics. However, where PAN presumably endeavors to allow path selection by an application, FAST allows applications to select their desired path characteristics and it is up to the network to select the actual path. This distinction is important to maximize flexibility, especially in situations where providing any detailed path information to untrusted end device is a security risk (which would be the typical case in a provider network or open Internet).

2.3 Emerging use cases

In a typical client/server model of serving content, end host clients communicate with servers on the Internet. Clients are typically user devices that are connected to the Internet through a provider network. In the case of mobile devices, such as smart phones, the devices are connected to the Internet through a mobile provider network. Content providers (web servers and content caches) tend to be more directly connected to the Internet, the largest of which can connect at exchange points.

Provider networks can be architected to provide different services and levels of services to their users based on characteristics of applications. For example, a mobile carrier network can provide different latency and throughput guarantees for different types of content. A network may offer different services for optimizing video: streaming an HD movie might need high throughput but not particularly low latency; a live video chat might have lower throughput demands but have stringent low latency requirements.

The emerging 3GPP standard for 5G defines a set of mechanisms to provide a rich array of services for users. These mechanisms employ
Network Function Virtualization (NFV), Service Function Chaining (SFC), and network slices that divide physical network resources into different virtualized slices to provide different services. To make use of these mechanisms, the applications running in UEs (User Equipment) will need to indicate desired services of the RAN (Radio Access Network). For instance, a video chat application may request bounded latency that is implemented by the network as a network slice; so packets sent by the application should be mapped to that network slice.

Note that network services requested by applications are relevant to both packets sent by an end node and those sent from a peer towards the end node. For the latter case, the network needs to be able to map packets sent from hosts on the Internet to the services requested by the receiving application.

3 Architecture

The figure below illustrates an example network path between two hosts on the Internet. Each host connects to the Internet via a provider network, and provider networks are connected in the Internet by transit networks.

```
+--------+    (     )      __________
| User 1 +---( Provider A )--( Transit )--( Provider B )---+ User 2 |
+--------+    (__________)    (       )    (__________)    +--------+
```

Figure 1

Within each provider network, services may be provided on behalf of the users of the network. In the figure above, Provider 1 may provide services and service agreements for users in its network including User 1; and likewise Provider B can provide services to users in its network including User 2. Transit networks service all users and don’t typically provide user specific services or service differentiation.

Services provided by different provider networks may be very different and dependent on the implementation of the network as well as the policies of the provider.

Based on this model, services and service differentiation can be considered local to each network provider. FAST is a mechanism whereby each user and application can request from its local provider the services to be applied to its traffic. A request for service is made to a FAST "ticket agent". The contents of the request describe
the services that application desires. The ticket agent responds with
a "ticket" that the application sets in its packets. When a packet is
sent by the application with a ticket attached, the ticket is
interpreted in the provider network to allow the packet to traverse
the network and to map the packet to the appropriate services. The
ticket is only relevant to the provider network that issued the
ticket, to the application itself and nodes outside of the provider
network the ticket is an uninterpretable opaque object.

To facilitate network traversal and service mapping in the reverse
direction for a flow, that is packets sent from a peer host, peer
hosts reflect tickets without modification or interpretation. This is
done by saving the ticket received in packets of a flow and attaching
that as a reflected ticket to packets being sent on the flow.

The use of tickets may be bilateral for a flow so that each peer
requests service from its local network. Therefore packets may
contain two types of tickets: one that is set by the sending host to
signal its local provider network, and the other is the reflected
ticket that is a signal to the provider network of the peer endpoint.

Tickets are scoped values, they only have meaning in the network in
which they were issued. The format, meaning, and interpretation of
tickets is network specific. By mutual agreement, two networks may
share the policy and interpretations of tickets. For instance, there
could be an agreement between two provider networks to interpret each
others tickets or to use a common format.

3.1 Example packet flow

Referencing the diagram in figure 1, consider that User 1 is
establishing a video chat with User 2 and wishes to have low latency
service for video applied by its local network (Provider 1). The flow
of events may be:

1. User 1 makes a ticket request to a ticket agent of Provider A
   that describes the video application and may include detailed
   characteristics such as resolution, frame rate, latency, etc.

2. The ticket agent issues a ticket to User 1 that indicates that
   packets of the flow have a right to traverse the network and
   the services to be applied to the packets of the flow.

3. The video chat application sends packets with the ticket
   attached for the video chat.

4. The first hop node in Provider A’s network interprets the
ticket in packets and applies the appropriate services (e.g.
sets diffserv, forwards on a network slice, encapsulates in MPLS, encapsulates with segment routing, etc.).

5. Packets traverse Provider A’s network with the appropriate services being applied.

6. Packets traverse transit networks and Provider B network, the attached tickets are ignored.

7. Packets are received at User 2. Attached tickets are saved in the context of the flow for the video chat.

8. User 2’s video chat application sends packets to User 1. The last ticket previously received from User 1 is now reflected in these packets.

9. Packets traverse Provider B network and transit networks, the reflected ticket is ignored.

10. An ingress node in Provider A’s network interprets the reflected ticket and applies appropriate services to the packets for traversing the local network.

11. Packets are forwarded within Provider’s A network with the appropriate services applied.

12. Packets are received at the host for User 1. The reflected ticket is validated by comparing the received ticket with that being sent for the flow. If the ticket is determined valid then the packet is accepted.

3.2 Requirements

The requirements for Firewall and Service Tickets are:

- Tickets SHOULD be stateless within the network. In particular intermediate nodes MUST NOT be required to create and maintain state for transport layer connections.

- Tickets MUST work in a multi-homed and multi-path environments.

- Outside of the network that issued a ticket, tickets MUST be opaque and obfuscated so that no application specific information is derivable.

- Tickets MUST work with any transport protocol as well as in the presence of any IP protocol feature (e.g. other extension headers are present).
o Tickets SHOULD minimize the changes to an application. Their use should be an "add-on" to the existing communications of an application.

o Tickets MUST deter spoofing and other misuse that might result in illegitimate use of network services or denial of service attack.

o Tickets MUST be contained in the IP layer protocol. In particular, tickets MUST NOT require parsing transport layer headers.

o Tickets MUST allow services to be applied in the return path of a communication. In a client/server application it is often the packets in the reverse path that require the most service (for instance if a video is being streamed to a client).

o A fallback MUST be present to handle the case that extension headers are dropped within the network or a peer node does not reflect tickets. A fallback allows functional communications but provides it in a potentially degraded mode of service.

4 Packet format

A ticket is sent in a Hop-by-Hop option.

4.1 Option format

The format of an Hop-by-Hop option containing a ticket is:

```
+---------------+---------------+---------------+
|  Option Type  |  Opt Data Len | Prop  | Rsvd |     Type      |
+---------------+---------------+---------------+-------+---------------+
|               |               |       |      |               |
+---------------+---------------+---------------+
|                | Ticket        |               |
+---------------+---------------+---------------+
```

Fields:

o Option type: Type of Hop-by-Hop option. This document proposes two possible values for ticket: an unmodifiable and a modifiable variant.

o Opt Data Len: Length of the option data field. The option data is comprised the Prop, Rsvd, and Type fields and the ticket
data.

- Prop: Indicates properties of the ticket for reflection and origin. Possible values are:
  - 0x0: Ticket from origin, don’t reflect at receiver
  - 0x1: Ticket from origin, reflect at receiver
  - 0x2: Reflected ticket
  - 0x3-0xf: Reserved

- Type: The type and format of the ticket. This value is used by nodes in the origin network to interpret the rest of the ticket data. Values for this field are specific to the network that issues the ticket.

4.2 Option types

There are two option numbers requested for the ticket option: 0x0F and 0x2F. The latter allows modification by network nodes. Since tickets are secured, only the nodes in the network that created a ticket will be able to modify it.

4.3 Ticket format

A ticket encodes service parameters that describe the desired services as well as additional fields that would be used to provide privacy and integrity.

The format of a ticket is defined by the network in which the ticket is issued. A ticket should be obfuscated or encrypted for privacy so that only the local network can interpret it. It should be uninterpretable to any nodes outside the network and to the application or host that is granted a ticket. It should be resistant to spoofing so that an attacker cannot illegitimately get service by applying a ticket seen on other flows.

It is recommended that tickets are encrypted and each ticket has an expiration time. For instance, a ticket may be created by encrypting the ticket data with an expiration time and using the source address, destination address, and a shared key as the key for encryption.
For example, a ticket with an expiration time may have the format:

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Expiration time                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
˜                     Service parameters                        ˜
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Where the expiration time is in a format understood by the local network nodes which maintain synchronized time. The Service parameters are relevant to local network nodes and describe the services to be applied. The service parameters could simply be a set of flags for services, an index to a service profile known by the network nodes, or possibly have more elaborate structure that could indicate numerical values for characteristics that have a range. The service parameters could also include a type field to allow a network to define different representations of service parameters.

A simple ticket containing a service protocol index might be:

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Expiration time                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Service Profile Index                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Where Type indicates the type of ticket and in this case indicates it is a service profile index. Service Profile Index could be an index into a table that describes the services to be applied.

5 Operation

5.1 Ticket types and ordering

There are three types of tickets that may be contained in a packet:

- origin tickets that are not reflected
- origin tickets to be reflected
- reflected tickets

Origin tickets are those set by an application that was issued a ticket and have an additional property indicating whether they are to
be reflected by a peer host. Reflected tickets are those that have been received and reflected by a peer host.

A sender SHOULD set at most one of each type of ticket in a packet. If different types of ticket are sent within a single packet they SHOULD have the following ordering:

1. origin tickets that are not reflected
2. origin tickets to be reflected
3. reflected tickets

If a packet contains more than one ticket option with the same ticket type, only the first option appearing in the list for the ticket type is processed. Additional options for the same ticket type are parsed but not processed.

5.2 Origin application processing

An origin application requests tickets, sets them in packets, and validates reflected tickets.

5.2.1 Ticket requests

An application that wishes to use network services first requests tickets from a ticket agent. The application request could be in the form of an XML structure with a canonical elements (the definition is outside the scope of this document). The application makes a request to the ticket agent for the local network. This could be done via a web service using REST APIs. Internally in the host, the ticket agent might be accessed through a library that interfaces to a ticket daemon that in turn arbitrates requests between the applications and a ticket agent in the network.

An issued ticket is opaque to the application and the application should not attempt to interpret it or take any other action other than attaching the ticket to its packets.

A ticket agent MAY provide both a origin ticket not to be reflected and one that is to be reflected. The intent is that different tickets can be used between the outbound and inbound paths for the flow. In the case that two tickets are provided, the origin ticket not to be reflected MUST appear first in the options list.

5.2.2 Ticket identification

Tickets are valid for a specific IP source and destination address
for which they were issued. Transport layer ports and other transport layer information are not included ticket identification, however an application can request tickets and validate reflected tickets on a per flow basis. Issued tickets are stored in the flow context and the saved information is used to validate reflected tickets.

5.2.3 Ticket use

When the ticket agent issues a ticket, the application sets the ticket as a Hop-by-Hop option. This is typically done by setting a socket option on a socket (in the case of TCP) or by indicating the option in the ancillary data when sending on an unconnected socket (in the case of UDP). The application SHOULD continue to use the same ticket for the flow until it is updated with a new ticket.

The ticket agent SHOULD return an expiration time with the ticket. An application can use the ticket until the expiration time, at which point it can request a new ticket to continue communications. In order to make the ticket transition process seamless an application MAY request a new ticket before the old one expires.

5.2.4 Ticket agent delegation

A network MAY delegate creation of tickets to hosts in a limited fashion. This would entail the network ticket agent issuing a master ticket to a host ticket agent which in turn can use the master ticket to create a limited number of tickets for its own use. The details of ticket agent delegation are outside the scope of this document.

5.3 Origin network processing

When a packet with a ticket enters a network, a network node can determine if the ticket originated in its network and must be processed. This is done by considering the origin of the ticket and the source or destination IP address. For an origin ticket (i.e. ticket is not reflected), the source address is considered. If the source address is local to the network then the ticket can be interpreted. For a reflected ticket, the destination address is considered. If the destination address is local to the network then the ticket can be interpreted.

If a ticket origin is determined to be the local network then the ticket is processed. The ticket is decrypted if necessary and the expiration time is checked. If the ticket is verified to be authentic and valid then the packet is mapped to be processed by the requested services. For instance, in a 5G network the packet may be forwarded on a network slice for the characteristics the application has
If an origin ticket cannot be verified, for instance the ticket cannot be authenticated, then the ticket SHOULD be ignored and the packet processed as though no ticket were present.

Note that there are logically only two ingress points into the network at which a provider needs to process tickets: when a local user sends a packet into the provider network with an origin ticket, and when a packet from an external network enters the provider’s network with a reflected ticket. Any ticket should be processed at most once within a network. Once a ticket is processed and mapped to the network’s service mechanisms it should not need further examination.

If there is more than one origin ticket present, then the first one encountered is processed and any additional origin tickets SHOULD be ignored by a network node. Note that this will be the case if a ticket agent issued both a origin ticket not to be reflected and one to be reflected; the ticket not to be reflected should appear first in the packet and thus would be the one processed by the node.

If there is more than one reflected ticket present, then the first one encountered is processed and any additional reflected tickets SHOULD be ignored.

5.4 Peer host processing

When a host receives a packet with a ticket whose type is "from origin and needs to be reflected", it SHOULD save the ticket in its flow context and reflect it on subsequent packets. When the application reflects the option, it copies the whole option and only modifies the type to indicate a "reflected ticket". The application SHOULD continue to reflect the ticket until a different one is received from the origin or a packet without a service ticket option is received on the flow. Note that the latest ticket that is received is the one to be reflected, if packets have been received out of order for a flow it is possible that the reflected ticket is from an earlier packet in a flow.

If there is more than one origin ticket to be reflected present, then the first one encountered is processed and any additional origin tickets to be reflected SHOULD be ignored.

A peer host MUST ignore received origin tickets that are not to be reflected.

5.5 Processing reflected tickets
5.5.1 Network processing

When a packet with a reflected ticket enters the origin network of the ticket, the ticket MUST be processed. The ticket is validated. Validation entails decoding or decrypting the ticket and checking the expiration time. If the ticket is valid and has not expired time then the packet is verified for forwarding.

A network MAY accept expired reflected tickets for some configurable period after the expiration time. Rate limiting SHOULD be applied to packets with expired reflected tickets. Accepting expired tickets is useful in the case that a connection goes idle and after sometime the remote peer starts to send. The ticket it reflects may be expired and presumably the receiving host will quickly respond with a new ticket.

5.5.2 Host processing

Upon receiving a packet with a reflected ticket an end host SHOULD validate the ticket before accepting the packet. This verification is done by comparing the received ticket to that which is set to be sent on the corresponding flow. If the tickets do not match then the packet is dropped and the event SHOULD be logged.

A host SHOULD retain and validate expired tickets that are reflected to allow a peer time to receive and reflect an updated ticket.

5.6 Handling dropped extension headers

The downside of using IPv6 extension headers on the Internet is that they are currently not completely reliable. Some intermediate nodes will drop extension headers with rates described in [RFC7872].

5.6.1 Mitigation for dropped extension headers

There are some mitigating factors for this problem:

- A provider network that implements tickets would need to ensure that extension headers are at least usable within their network.
- Transit networks are less likely to arbitrarily drop packets with extension headers.
- Many content providers, especially the larger ones, may be directly connected to the Internet. For example, front end web servers may be co-located as exchange points.
- The requirement that nodes must process Hop-by-hop options has been relaxed in [RFC8200]. It is permissible for intermediate
nodes to ignore them.

- Increased deployment of IPv6 and viable use cases of extension headers, such as described here, may motivate vendors to fix issues with extension headers.

### 5.6.2 Fallback for dropped extension headers

Since the possibility that extension headers are dropped cannot be completely eliminated, a fallback is included for use with tickets.

When an application connects to a new destination for which it has no history about the viability of extension headers, it can perform a type of Happy Eyeballs probing. The concept is for a host to send a number of packets with and without tickets. The application can observe whether packets with tickets are being dropped or not being reflected.

There are a few possible outcomes of this process:

- A packet with a ticket is dropped and an ICMP for extension headers [ICMPEH] processing limits is received. This is a strong signal that extension headers are not viable to the destination and should not be used for the flow.

- A packet with a ticket is dropped and no ICMP error is received. This is a signal that extension headers may not be usable. If such drops are observed for all or a significant fraction of packets and there are no drops for packets that were sent without tickets, then extension headers should be considered not viable for the flow.

- Packets with tickets are not being dropped, however tickets are not being reflected. This is a signal that the peer application does not support reflection. Tickets may be sent, however they are only useful in the outbound path.

- Packets with tickets are not being dropped and tickets are properly being reflected. Tickets are useful in both directions.

If extension headers are found to not be viable or tickets are not being properly reflected, a possible fallback is to not use tickets. In this case, communications might remain functional, however they would be operate in a degraded mode of service. The network may fallback to creating per flow state in the network; the ticket that an application sent with packets during probing could be used to instantiate the service characteristics maintained in a flow state.
6 Implementation considerations

6.1 Origin applications

Existing client applications can be modified to request tickets and set them in packets. The OS networking stack may need some small changes or configuration to enable an application to specify the option for its packets.

The interface to the ticket agent would likely be via a library API.

For a connected socket (TCP, SCTP, or connected UDP socket), a Hop-by-Hop option can be set on the socket via the setsockopt system call in BSD socket. For an unconnected socket (UDP) the ticket option can be set as ancillary data in the sendmsg system call.

Happy Eyeballs for extension headers, described in section 5.6.2, could be implemented in the networking stack for a connection oriented transport protocol such a TCP. For connectionless protocols, probing could be handled by an application library.

6.2 Ticket reflection

To perform ticket reflection, servers must be updated. In the case of a connected socket (TCP, SCTP, or a connected UDP socket) this can be done as relatively minor change to the kernel networking stack which would be transparent to applications. For unconnected UDP, an application could receive the ticket as part of the ancillary data in recvmsg system call, and then send the reflected ticket in a reply using ancillary data in sendmsg.

7 Security Considerations

There are two main security considerations:

- Leakage of content specific information to untrusted third parties must be avoided.
- Tickets cannot be forged, illegitimately used, or otherwise abused.

Tickets may be visible to the Internet including untrusted and unknown networks in the path of sent packets. Therefore, tickets should be encrypted or obfuscated by the origin network.

Tickets need to have an expiration time, must be resistant to forgery, and must be nontransferable. A ticket should be valid for the specific source and destination addresses that it was issued for.
Tickets are revocable by implemented a black-list contained revoked tickets.

8 IANA Considerations

IANA is requested to assigned the following Hop-By-Hop options:

<table>
<thead>
<tr>
<th>Hex Value</th>
<th>Binary value</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0F</td>
<td>00 0 01111</td>
<td>Firewall and Service Ticket</td>
<td>This document</td>
</tr>
<tr>
<td>0x2F</td>
<td>00 1 01111</td>
<td>Modifiable Firewall and Service Ticket</td>
<td>This document</td>
</tr>
</tbody>
</table>

IANA is requested to set up a registry for the Ticket property. These types are 4 bit values. New values for 0x3-0xf are assigned via Standards Action [RFC5226].

<table>
<thead>
<tr>
<th>Ticket type</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>Ticket from origin and don’t reflect</td>
<td>This document</td>
</tr>
<tr>
<td>0x1</td>
<td>Ticket from origin and reflect</td>
<td>This document</td>
</tr>
<tr>
<td>0x2</td>
<td>Reflected ticket</td>
<td>This document</td>
</tr>
</tbody>
</table>

9 References

9.1 Normative References


[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an

9.2 Informative References


[PAN] Trammell, B., "Open Questions in Path Aware Networking", draft-trammell-panrg-questions-02, December 2017

[ICMPEH] Herbert, T., "ICMPv6 errors for discarding packets due to processing limits", draft-herbert-6man-icmp-limits-03, January 2018

[LIMDOM] Carpenter, B. and Liu, B., "Limited Domains and Internet Protocols", draft-carpenter-limited-domains-03, June 2018

Author's Address

Tom Herbert
Quantonium
Santa Clara, CA
USA

Email: tom@quantonium.net
DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)

draft-ietf-tsvwg-aqm-dualq-coupled-06

Abstract

Data Centre TCP (DCTCP) was designed to provide predictably low queuing latency, near-zero loss, and throughput scalability using explicit congestion notification (ECN) and an extremely simple marking behaviour on switches. However, DCTCP does not co-exist with existing TCP traffic---DCTCP is so aggressive that existing TCP algorithms approach starvation. So, until now, DCTCP could only be deployed where a clean-slate environment could be arranged, such as in private data centres. This specification defines ‘DualQ Coupled Active Queue Management (AQM)’ to allow scalable congestion controls like DCTCP to safely co-exist with classic Internet traffic. The Coupled AQM ensures that a flow runs at about the same rate whether it uses DCTCP or TCP Reno/Cubic, but without inspecting transport layer flow identifiers. When tested in a residential broadband setting, DCTCP achieved sub-millisecond average queuing delay and zero congestion loss under a wide range of mixes of DCTCP and ‘Classic’ broadband Internet traffic, without compromising the performance of the Classic traffic. The solution also reduces network complexity and eliminates network configuration.

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 19, 2019.

Copyright Notice

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

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

Table of Contents

1. Introduction ........................................... 3
   1.1. Problem and Scope ............................... 3
   1.2. Terminology ..................................... 5
   1.3. Features ........................................ 6
2. DualQ Coupled AQM ............................... 7
   2.1. Coupled AQM ................................... 7
   2.2. Dual Queue ..................................... 8
   2.3. Traffic Classification .......................... 8
   2.4. Overall DualQ Coupled AQM Structure .......... 9
   2.5. Normative Requirements for a DualQ Coupled AQM 11
      2.5.1. Functional Requirements ................... 11
         2.5.1.1. Requirements in Unexpected Cases ... 12
      2.5.2. Management Requirements ................. 13
3. IANA Considerations .............................. 14
4. Security Considerations ......................... 14
   4.1. Overload Handling ............................. 14
      4.1.1. Avoiding Classic Starvation: Sacrifice L4S Throughput
             or Delay? ................................ 15
      4.1.2. Congestion Signal Saturation: Introduce L4S Drop or
             Delay? .................................... 16
   4.1.3. Protecting against Unresponsive ECN-Capable Traffic . 17
5. Acknowledgements .................................... 17
6. References .......................................... 18
   6.1. Normative References .......................... 18
   6.2. Informative References ....................... 18
Appendix A. Example DualQ Coupled PI2 Algorithm .. 21
1. Introduction

1.1. Problem and Scope

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 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 and, if all the traffic on a link is marked as EF, Diffserv cannot reduce the delay of any of it. In contrast, the Low Latency Low Loss Scalable throughput (L4S) approach removes the causes of any unnecessary queuing 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.

It seems that further changes to the network alone will now yield diminishing returns. Data Centre TCP (DCTCP [RFC8257]) teaches us that a small but radical change to TCP is needed to cut two major outstanding causes of queuing delay variability:

1. the ‘sawtooth’ varying rate of TCP itself;
2. the smoothing delay deliberately introduced into AQMs to permit bursts without triggering losses.

The former causes a flow’s round trip time (RTT) to vary from about 1 to 2 times the base RTT between the machines in question. The latter delays the system’s response to change by a worst-case (transcontinental) RTT, which could be hundreds of times the actual RTT of typical traffic from localized CDNs.

Latency is not our only concern:

3. It was known when TCP was first developed that it would not scale to high bandwidth-delay products.

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 TCPs such as DCTCP cause ‘classic’ TCP to starve itself, which is why they have been confined to private data centres or research testbeds (until now).
This document specifies a ‘DualQ Coupled AQM’ extension that solves the problem of coexistence between scalable and classic flows, without having to inspect flow identifiers. The AQM is not like flow-queuing approaches [RFC8290] that classify packets by flow identifier into numerous separate queues in order to isolate sparse flows from the higher latency in the queues assigned to heavier flow. In contrast, the AQM exploits the behaviour of scalable congestion controls like DCTCP so that every packet in every flow sharing the queue for DCTCP-like traffic can be served with very low latency.

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

The overall L4S architecture is described in [I-D.ietf-tsvwg-l4s-arch]. The supporting papers [PI2] and [DCttH15] give the full rationale for the AQM’s design, both discursively and in more precise mathematical form.

1.2. 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]. 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.

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

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

Low-Latency, Low-Loss and Scalable (L4S, denoted by subscript L): The 'L4S' service is intended for a set of congestion controls with scalable properties such as DCTCP (e.g. Relentless [Mathis09]).
Either service can cope with a proportion of unresponsive or less-responsive traffic as well (e.g. DNS, VoIP, etc), just as a single queue AQM can. The DualQ Coupled AQM behaviour is similar to a single FIFO queue with respect to unresponsive and overload traffic.

1.3. Features

The AQM couples marking and/or dropping across the two queues such that a flow will get roughly the same throughput whichever it uses. Therefore both queues can feed into the full capacity of a link and no rates need to be configured for the queues. The L4S queue enables scalable congestion controls like DCTCP to give stunningly low and predictably low latency, without compromising the performance of competing 'Classic' Internet traffic. Thousands of tests have been conducted in a typical fixed residential broadband setting. Typical experiments used base round trip delays up to 100ms between the data centre and home network, and large amounts of background traffic in both queues. For every L4S packet, the AQM kept the average queuing delay below 1ms (or 2 packets if serialization delay is bigger for slow links), and no losses at all were introduced by the AQM. Details of the extensive experiments will be made available [PI2] [DCttH15].

Subjective testing was also conducted using a demanding panoramic interactive video application run over a stack with DCTCP enabled and deployed on the testbed. Each user could pan or zoom their own high definition (HD) sub-window of a larger video scene from a football match. Even though the user was also downloading large amounts of L4S and Classic data, latency was so low that the picture appeared to stick to their finger on the touchpad (all the L4S data achieved the same ultra-low latency). With an alternative AQM, the video noticeably lagged behind the finger gestures.

Unlike Diffserv Expedited Forwarding, the L4S queue does not have to be limited to a small proportion of the link capacity in order to achieve low delay. The L4S queue can be filled with a heavy load of capacity-seeking flows like DCTCP and still achieve low delay. The L4S queue does not rely on the presence of other traffic in the Classic queue that can be 'overtaken'. It gives low latency to L4S traffic whether or not there is Classic traffic, and the latency of Classic traffic does not suffer when a proportion of the traffic is L4S. The two queues are only necessary because DCTCP-like flows cannot keep latency predictably low and keep utilization high if they are mixed with legacy TCP flows.

The experiments used the Linux implementation of DCTCP that is deployed in private data centres, without any modification despite its known deficiencies. Nonetheless, certain modifications will be
necessary before DCTCP is safe to use on the Internet, which are
recorded in Appendix A of [I-D.ietf-tsvwg-ecn-l4s-id]. However, the
focus of this specification is to get the network service in place.
Then, without any management intervention, applications can exploit
it by migrating to scalable controls like DCTCP, which can then
evolve _while_ their benefits are being enjoyed by everyone on the
Internet.

2. DualQ Coupled AQM

There are two main aspects to the approach:

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

2.1. Coupled AQM

In the 1990s, the 'TCP formula' was derived for the relationship
between TCP’s congestion window, cwnd, and its drop probability, p.
To a first order approximation, cwnd of TCP Reno is inversely
proportional to the square root of p.

TCP Cubic implements a Reno-compatibility mode, which is the only
relevant mode for typical RTTs under 20ms as long as the throughput
of a single flow is less than about 500Mb/s. Therefore it can be
assumed that Cubic traffic behaves similarly to Reno (but with a
slightly different constant of proportionality), and the term
‘Classic’ will be used for the collection of Reno-friendly traffic
including Cubic in Reno mode.

The supporting paper [PI2] includes the derivation of the equivalent
rate equation for DCTCP, for which cwnd is inversely proportional to
p (not the square root), where in this case p is the ECN marking
probability. DCTCP is not the only congestion control that behaves
like this, so the term ‘L4S’ traffic will be used for all similar
behaviour.

In order to make a DCTCP flow run at roughly the same rate as a Reno
TCP flow (all other factors being equal), the drop or marking
probability for Classic traffic, p_C has to be distinct from the
marking probability for L4S traffic, p_L (in contrast to RFC3168
which requires them to be the same). It is necessary to make the
Classic drop probability p_C proportional to the square of the L4S
marking probability p_L. This makes the Reno flow rate roughly equal
the DCTCP flow rate, because it squares the square root of p_C in the
Reno rate equation to make it proportional to the straight $p_L$ in the DCTCP rate equation.

Stating this as a formula, the relation between Classic drop probability, $p_C$, and L4S marking probability, $p_L$ needs to take the form:

$$p_C = \left( \frac{p_L}{k} \right)^2$$

(1)

where $k$ is the constant of proportionality.

2.2. Dual Queue

Classic traffic typically builds a large queue to prevent under-utilization. Therefore a separate queue is provided for L4S traffic, and it is scheduled with priority over Classic. Priority is conditional to prevent starvation of Classic traffic.

Nonetheless, coupled marking ensures that giving priority to L4S traffic still leaves the right amount of spare scheduling time for Classic flows to each get equivalent throughput to DCTCP flows (all other factors such as RTT being equal). The algorithm achieves this without having to inspect flow identifiers.

2.3. Traffic Classification

Both the Coupled AQM and DualQ mechanisms need an identifier to distinguish L and C packets. A separate draft [I-D.ietf-tsvwg-ecn-l4s-id] recommends using the ECT(1) codepoint of the ECN field as this identifier, having assessed various alternatives. An additional process document has proved necessary to make the ECT(1) codepoint available for experimentation [RFC8311].

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

In addition, other identifiers could be used to classify certain additional packet types into the L queue, that are deemed not to risk harming the L4S service. For instance addresses of specific applications or hosts (see [I-D.ietf-tsvwg-ecn-l4s-id]), specific Diffserv codepoints such as EF (Expedited Forwarding) and Voice-Admit
service classes (see [I-D.briscoe-tsvwg-l4s-diffserv]) or certain protocols (e.g. ARP, DNS).

Note that the DualQ Coupled AQM only reads these classifiers, it MUST NOT re-mark or alter these identifiers (except for marking the ECN field with the CE codepoint - with increasing frequency to indicate increasing congestion).

2.4. Overall DualQ Coupled AQM Structure

Figure 1 shows the overall structure that any DualQ Coupled AQM is likely to have. This schematic is intended to aid understanding of the current designs of DualQ Coupled AQMs. However, it is not intended to preclude other innovative ways of satisfying the normative requirements in Section 2.5 that minimally define a DualQ Coupled AQM.

The classifier on the left separates incoming traffic between the two queues (L and C). Each queue has its own AQM that determines the likelihood of dropping or marking \( p_L \) and \( p_C \). Nonetheless, the AQM for Classic traffic is implemented in two stages: i) a base stage that outputs an internal probability \( p' \) (pronounced p-prime); and ii) a squaring stage that outputs \( p_C \), where

\[
p_C = (p')^2.
\]  
(2)

This allows \( p_L \) to be coupled to \( p_C \) by marking L4S traffic proportionately to the intermediate output from the first stage. Specifically, the output of the base AQM is coupled across to the L queue in proportion to the output of the base AQM:

\[
p_{CL} = kp',
\]  
(3)

where \( k \) is the constant coupling factor (see Appendix C) and \( p_{CL} \) is the output from the coupling between the C queue and the L queue.

It can be seen in the following that these two transformations of \( p' \) implement the required coupling given in equation (1) earlier. Substituting for \( p' \) from equation (3) into (2):

\[
p_C = \left( \frac{p_{CL}}{k} \right)^2.
\]
be driven by the coupling, that is \( p_L = p_{CL} \). So, whenever the
coupling is needed, as required from equation (1):

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

Legend: \( \Longrightarrow \) traffic flow; \( \longrightarrow \) control dependency.

Figure 1: DualQ Coupled AQM Schematic

After the AQMs have applied their dropping or marking, the scheduler
forwards their packets to the link, giving priority to L4S traffic.
Priority has to be conditional in some way (see Section 4.1). Simple
strict priority is inappropriate otherwise it could lead the L4S
queue to starve the Classic queue. For example, consider the case
where a continually busy L4S queue blocks a DNS request in the
Classic queue, arbitrarily delaying the start of a new Classic flow.

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

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

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

Both AQMs regulate their queue in units of time not bytes. As already explained, this ensures configuration can be invariant for different drain rates. With AQMs in a dualQ structure this is particularly important because the drain rate of each queue can vary rapidly as flows for the two queues arrive and depart, even if the combined link rate is constant.

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

2.5. Normative Requirements for a DualQ Coupled AQM

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

2.5.1. Functional Requirements

In the Dual Queue, L4S packets MUST be given priority over Classic, although priority MUST be bounded in order not to starve Classic traffic.

Whatever identifier is used for L4S experiments, [I-D.ietf-tsvwg-ecn-l4s-id] defines the meaning of an ECN marking on L4S traffic, relative to drop of Classic traffic. In order to prevent starvation of Classic traffic by scalable L4S traffic, it says, "The likelihood that an AQM drops a Not-ECT Classic packet (\(p_C\)) MUST be roughly proportional to the square of the likelihood that it would have marked it if it had been an L4S packet (\(p_L\))." In other words, in any DualQ Coupled AQM, the power to which \(p_L\) is raised in Eqn. (1) MUST be 2. The term 'likelihood' is used to allow for marking and dropping to be either probabilistic or deterministic.

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

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

(ToDo: The requirements for scalable congestion controls on the Internet (termed the TCP Prague requirements) [I-D.ietf-tsvwg-ecn-l4s-id] are not necessarily final. If the aggressiveness of DCTCP is not defined as the benchmark for scalable controls on the Internet, the recommended value of k will also be subject to change.)

The choice of k is a matter of operator policy, and operators MAY choose a different value using Table 1 and the guidelines in Appendix C.

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

2.5.1.1. Requirements in Unexpected Cases

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

- If a packet that does not carry an ECT(1) or CE codepoint is classified into the L queue:
* if the packet is ECT(0), the L AQM SHOULD apply drop using a drop probability appropriate to Classic congestion control and appropriate to the target delay in the L queue

* if the packet is Not-ECT, the appropriate action depends on whether some other function is protecting the L queue from misbehaving flows (e.g. per-flow queue protection or latency policing):

  + If separate queue protection is provided, the L AQM SHOULD ignore the packet and forward it unchanged, meaning it should not calculate whether to apply congestion notification and it should neither drop nor CE-mark the packet (for instance, the operator might classify EF traffic that is unresponsive to drop into the L queue, alongside responsive L4S-ECN traffic)

  + if separate queue protection is not provided, the L AQM SHOULD apply drop using a drop probability appropriate to Classic congestion control and appropriate to the target delay in the L queue

  o If a packet that carries an ECT(1) codepoint is classified into the C queue:

    * the C AQM SHOULD apply CE-marking using the coupled AQM probability $p_{CL} (= k'p')$.

If the DualQ Coupled AQM has detected overload, it will signal congestion solely using drop, irrespective of the ECN field.

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

2.5.2. Management Requirements

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

  o Optional packet classifier(s) to use in addition to the ECN field (see Section 2.3);
- Expected typical RTT (a parameter for typical or target queuing delay in each queue might be configurable instead);
- Expected maximum RTT (a stability parameter that depends on maximum RTT might be configurable instead);
- Coupling factor, k;
- The limit to the conditional priority of L4S (scheduler-dependent, e.g. the scheduler weight for WRR, or the time-shift for time-shifted FIFO);
- The maximum Classic ECN marking probability, p_Cmax, before switching over to drop.

An experimental DualQ Coupled AQM SHOULD allow the operator to monitor the following operational statistics:

- Bits forwarded (total and per queue per sample interval), from which utilization can be calculated
- Q delay (per queue over sample interval)
- Total packets arriving, enqueued and dequeued (per queue per sample interval)
- ECN packets marked, non-ECN packets dropped, ECN packets dropped (per queue per sample interval), from which marking and dropping probabilities can be calculated
- Time and duration of each overload event.

The type of statistics produced for variables like Q delay (mean, percentiles, etc.) will depend on implementation constraints.

3. IANA Considerations

This specification contains no IANA considerations.

4. Security Considerations

4.1. Overload Handling

Where the interests of users or flows might conflict, it could be necessary to police traffic to isolate any harm to the performance of individual flows. However it is hard to avoid unintended side-effects with policing, and in a trusted environment policing is not...
necessary. Therefore per-flow policing needs to be separable from a basic AQM, as an option under policy control.

However, a basic DualQ AQM does at least need to handle overload. A useful objective would be for the overload behaviour of the DualQ AQM to be at least no worse than a single queue AQM. However, a trade-off needs to be made between complexity and the risk of either traffic class harming the other. In each of the following three subsections, an overload issue specific to the DualQ is described, followed by proposed solution(s).

Under overload the higher priority L4S service will have to sacrifice some aspect of its performance. Alternative solutions are provided below that each relax a different factor: e.g. throughput, delay, drop. Some of these choices might need to be determined by operator policy or by the developer, rather than by the IETF. {ToDo: Reach consensus on which it is to be in each case.}

4.1.1. Avoiding Classic Starvation: Sacrifice L4S Throughput or Delay?

Priority of L4S is required to be conditional to avoid total throughput starvation of Classic by heavy L4S traffic. This raises the question of whether to sacrifice L4S throughput or L4S delay (or some other policy) to mitigate starvation of Classic:

Sacrifice L4S throughput: By using weighted round robin as the conditional priority scheduler, the L4S service can sacrifice some throughput during overload to guarantee a minimum throughput service for Classic traffic. The scheduling weight of the Classic queue should be small (e.g. 1/16). Then, in most traffic scenarios the scheduler will not interfere and it will not need to - the coupling mechanism and the end-systems will share out the capacity across both queues as if it were a single pool. However, because the congestion coupling only applies in one direction (from C to L), if L4S traffic is over-aggressive or unresponsive, the scheduler weight for Classic traffic will at least be large enough to ensure it does not starve.

In cases where the ratio of L4S to Classic flows (e.g. 19:1) is greater than the ratio of their scheduler weights (e.g. 15:1), the L4S flows will get less than an equal share of the capacity, but only slightly. For instance, with the example numbers given, each L4S flow will get \((15/16)/19 = 4.9\%\) when ideally each would get \(1/20=5\%\). In the rather specific case of an unresponsive flow taking up a large part of the capacity set aside for L4S, using WRR could significantly reduce the capacity left for any responsive L4S flows.
Sacrifice L4S Delay: To control milder overload of responsive traffic, particularly when close to the maximum congestion signal, the operator could choose to control overload of the Classic queue by allowing some delay to ‘leak’ across to the L4S queue. The scheduler can be made to behave like a single First-In First-Out (FIFO) queue with different service times by implementing a very simple conditional priority scheduler that could be called a "time-shifted FIFO" (see the Modifier Earliest Deadline First (MEDF) scheduler of [MEDF]). This scheduler adds tshift to the queue delay of the next L4S packet, before comparing it with the queue delay of the next Classic packet, then it selects the packet with the greater adjusted queue delay. Under regular conditions, this time-shifted FIFO scheduler behaves just like a strict priority scheduler. But under moderate or high overload it prevents starvation of the Classic queue, because the time-shift (tshift) defines the maximum extra queuing delay of Classic packets relative to L4S.

The example implementation in Appendix A can implement either policy.

4.1.2. Congestion Signal Saturation: Introduce L4S Drop or Delay?

To keep the throughput of both L4S and Classic flows roughly equal over the full load range, a different control strategy needs to be defined above the point where one AQM first saturates to a probability of 100% leaving no room to push back the load any harder. If k>1, L4S will saturate first, but saturation can be caused by unresponsive traffic in either queue.

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

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

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

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

The example implementation in Appendix A applies only the "drop on saturation" policy.

4.1.3. Protecting against Unresponsive ECN-Capable Traffic

Unresponsive traffic has a greater advantage if it is also ECN-capable. The advantage is undetectable at normal low levels of drop/marking, but it becomes significant with the higher levels of drop/marking typical during overload. This is an issue whether the ECN-capable traffic is L4S or Classic.

This raises the question of whether and when to switch off ECN marking and use solely drop instead, as required by both Section 7 of [RFC3168] and Section 4.2.1 of [RFC7567].

Experiments with the DualPI2 AQM (Appendix A) have shown that introducing 'drop on saturation' at 100% L4S marking addresses this problem with unresponsive ECN as well as addressing the saturation problem. It leaves only a small range of congestion levels where unresponsive traffic gains any advantage from using the ECN capability, and the advantage is hardly detectable [DualQ-Test].

5. Acknowledgements

Thanks to Anil Agarwal, Sowmini Varadhan’s and Gabi Bracha for detailed review comments particularly of the appendices and suggestions on how to make our explanation clearer. Thanks also to Greg White and Tom Henderson for insights on the choice of schedulers and queue delay measurement techniques.

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

6.1. Normative References


6.2. 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-00 (work in progress), March 2018.


(To appear)


Appendix A. Example DualQ Coupled PI2 Algorithm

As a first concrete example, the pseudocode below gives the DualPI2 algorithm. DualPI2 follows the structure of the DualQ Coupled AQM framework in Figure 1. A simple step threshold (in units of queuing time) is used for the Native L4S AQM, but a ramp is also described as an alternative. And the PI2 algorithm [PI2] is used for the Classic AQM. PI2 is an improved variant of the PIE AQM [RFC8033].

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

A full open source implementation for Linux is available at: https://github.com/olgabo/dualpi2.

A.1. Pass #1: Core Concepts

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

- initialization code (Figure 2) that sets parameter defaults (the API for setting non-default values is omitted for brevity)
- enqueue code (Figure 3)
- dequeue code (Figure 4)
- code to regularly update the base probability (p) used in the dequeue code (Figure 5).

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

- scheduler(), which selects between the head packets of the two queues; the choice of scheduler technology is discussed later;
o `cq.len()` or `lq.len()` returns the current length (aka. backlog) of
the relevant queue in bytes;

o `cq.time()` or `lq.time()` returns the current queuing delay (aka.
sojourn time or service time) of the relevant queue in units of
time;

Queuing delay could be measured directly by storing a per-packet
time-stamp as each packet is enqueued, and subtracting this from the
system time when the packet is dequeued. If time-stamping is not
easy to introduce with certain hardware, queuing delay could be
predicted indirectly by dividing the size of the queue by the
predicted departure rate, which might be known precisely for some
link technologies (see for example [RFC8034]).

In our experiments so far (building on experiments with PIE) on
broadband access links ranging from 4 Mb/s to 200 Mb/s with base RTTs
from 5 ms to 100 ms, DualPI2 achieves good results with the default
parameters in Figure 2. The parameters are categorised by whether
they relate to the Base PI2 AQM, the L4S AQM or the framework
coupling them together. Variables derived from these parameters are
also included at the end of each category. Each parameter is
explained as it is encountered in the walk-through of the pseudocode
below.
1:  dualpi2_params_init(...) { % Set input parameter defaults
2:    % PI2 AQM parameters
3:    target = 15 ms % PI AQM Classic queue delay target
4:    Tupdate = 16 ms % PI Classic queue sampling interval
5:    alpha = 10 Hz^2 % PI integral gain
6:    beta = 100 Hz^2 % PI proportional gain
7:    p_Cmax = 1/4 % Max Classic drop/mark prob
8:    % Derived PI2 AQM variables
9:    alpha_U = alpha * Tupdate % PI integral gain per update interval
10:   beta_U = beta * Tupdate % PI prop’nal gain per update interval
11: }
12: % DualQ Coupled framework parameters
13:  k = 2 % Coupling factor
14:  % scheduler weight or equival’t parameter (scheduler-dependent)
15:  limit = MAX_LINK_RATE * 250 ms % Dual buffer size
16: % L4S AQM parameters
17:  T_time = 1 ms % L4S marking threshold in time
18:  T_len = 2 * MTU % Min L4S marking threshold in bytes
19: % Derived L4S AQM variables
20:  p_Lmax = min(k*sqrt(p_Cmax), 1) % Max L4S marking prob
21: }

Figure 2: Example Header Pseudocode for DualQ Coupled PI2 AQM

The overall goal of the code is to maintain the base probability (p), which is an internal variable from which the marking and dropping probabilities for L4S and Classic traffic (p_L and p_C) are derived. The variable named p in the pseudocode and in this walk-through is the same as p’ (p-prime) in Section 2.4. The probabilities p_L and p_C are derived in lines 3, 4 and 5 of the dualpi2_update() function (Figure 5) then used in the dualpi2_dequeue() function (Figure 4). The code walk-through below builds up to explaining that part of the code eventually, but it starts from packet arrival.

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

Figure 3: Example Enqueue Pseudocode for DualQ Coupled PI2 AQM
```plaintext
1:  dualpi2_dequeue(lq, cq, pkt) {     % Couples L4S & Classic queues
2:    while ( lq.len() + cq.len() > 0 )
3:      if ( scheduler() == lq ) {
4:        lq.dequeue(pkt)                      % Scheduler chooses lq
5:        if ( ((lq.time() > T_time)              % step marking ...
6:              AND (lq.len() > T_len))
7:          OR (p_CL > rand()) )             % ...or linear marking
8:          mark(pkt)
9:      } else {
10:       cq.dequeue(pkt)                      % Scheduler chooses cq
11:       if ( p_C > rand() ) {               % probability p_C = p^2
12:         if ( ecn(pkt) == 0 ) {           % if ECN field = not-ECT
13:           drop(pkt)                                % squared drop
14:           continue        % continue to the top of the while loop
15:         }                                  % squared mark
16:       mark(pkt)
17:     }
18:   }
19:   return(pkt)                      % return the packet and stop
20: }                                  % no packet to dequeue
21: }

Figure 4: Example Dequeue Pseudocode for DualQ Coupled PI2 AQM

When packets arrive, first a common queue limit is checked as shown in line 2 of the enqueuing pseudocode in Figure 3. Note that the limit is deliberately tested before enqueue to avoid any bias against larger packets (so the actual buffer has to be one MTU larger than limit). If limit is not exceeded, the packet will be classified and enqueued to the Classic or L4S queue dependent on the least significant bit of the ECN field in the IP header (line 5). Packets with a codepoint having an LSB of 0 (Not-ECT and ECT(0)) will be enqueued in the Classic queue. Otherwise, ECT(1) and CE packets will be enqueued in the L4S queue. Optional additional packet classification flexibility is omitted for brevity (see [I-D.ietf-tsvwg-ecn-l4s-id]).

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

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

- If an L4S packet is scheduled, lines 5 to 8 mark the packet if either the L4S threshold (T_time) is exceeded, or if a random marking decision is drawn according to p_CL (maintained by the dualpi2_update() function discussed below). This logical ‘OR’ on a per-packet basis implements the max() function shown in Figure 1 to couple the outputs of the two AQMs together. The L4S threshold is usually in units of time (default T_time = 1 ms). However, on slow links the packet serialization time can approach the threshold T_time, so line 6 sets a floor of T_len (=2 MTU) to the threshold, otherwise marking is always too frequent on slow links.

- If a Classic packet is scheduled, lines 10 to 17 drop or mark the packet based on the squared probability p_C.

There is some concern that using a step function for the Native L4S AQM requires end-systems to smooth the signal for a lot longer - until its fidelity is sufficient. The latency benefits of a ramp are being investigated as a simple alternative to the step. This ramp would be similar to the RED algorithm, with the following differences:

- The min and max of the ramp are defined in units of queueing delay, not bytes, so that configuration remains invariant as the queue departure rate varies.

- It uses instantaneous queueing delay without smoothing (smoothing is done in the end-systems).

- Determinism is being experimented with instead of randomness; to reduce the delay necessary to smooth out the noise of randomness from the signal. For each packet, the algorithm would accumulate p’_L in a counter and mark the packet that took the counter over 1, then subtract 1 from the counter and continue.

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

This ramp algorithm would require two configuration parameters (min and max threshold in units of queuing time), in contrast to the single parameter of a step.
1: dualpi2_update(lq, cq, target) { % Update p every Tupdate
2:   curq = cq.time() % use queuing time of first-in Classic packet
3:   p = p + alpha_U * (curq - target) + beta_U * (curq - prevq)
4:   p_CL = p * k % Coupled L4S prob = base prob * coupling factor
5:   p_C = p^2 % Classic prob = (base prob)^2
6:   prevq = curq
7: }

Figure 5: Example PI-Update Pseudocode for DualQ Coupled PI2 AQM

The base probability (p) is kept up to date by the core PI algorithm in Figure 5, which is executed every Tupdate.

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

The algorithm centres on line 3, which is a classical Proportional-Integral (PI) controller that alters p dependent on: a) the error between the current queuing delay (curq) and the target queuing delay (‘target’ - see [RFC8033]); and b) the change in queuing delay since the last sample. The name ‘PI’ represents the fact that the second factor (how fast the queue is growing) is _P_roportional to load while the first is the _I_ntegral of the load (so it removes any standing queue in excess of the target).

The two ‘gain factors’ in line 3, alpha_U and beta_U, respectively weight how strongly each of these elements ((a) and (b)) alters p. They are in units of ‘per second of delay’ or Hz, because they transform differences in queuing delay into changes in probability.

alpha_U and beta_U are derived from the input parameters alpha and beta (see lines 5 and 6 of Figure 2). These recommended values of alpha and beta come from the stability analysis in [PI2] so that the AQM can change p as fast as possible in response to changes in load without over-compensating and therefore causing oscillations in the queue.

alpha and beta determine how much p ought to change if it was updated every second. It is best to update p as frequently as possible, but the update interval (Tupdate) will probably be constrained by hardware performance. For link rates from 4 - 200 Mb/s, we found Tupdate=16ms (as recommended in [RFC8033]) is sufficient. However small the chosen value of Tupdate, p should change by the same amount per second, but in finer more frequent steps. So the gain factors
used for updating \( p \) in Figure 5 need to be scaled by \((T_{update}/1s)\), which is done in lines 9 and 10 of Figure 2). The suffix \('_U'\) represents 'per update time' (\( T_{update} \)).

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

Because the coupled L4S marking probability \( (p_{CL}) \) is factored up by \( k \), the dynamic gain parameters \( \alpha \) and \( \beta \) are also inherently factored up by \( k \) for the L4S queue, which is necessary to ensure that Classic TCP and DCTCP controls have the same stability. So, if \( \alpha \) is 10 Hz\(^2\), the effective gain factor for the L4S queue is \( k*\alpha \), which is 20 Hz\(^2\) with the default coupling factor of \( k=2 \).

Unlike in PIE [RFC8033], \( \alpha_U \) and \( \beta_U \) do not need to be tuned every \( T_{update} \) dependent on \( p \). Instead, in PI2, \( \alpha_U \) and \( \beta_U \) are independent of \( p \) because the squaring applied to Classic traffic tunes them inherently. This is explained in [PI2], which also explains why this more principled approach removes the need for most of the heuristics that had to be added to PIE.

(ToDo: Scaling \( \beta \) with \( T_{update} \) and scaling both \( \alpha \) & \( \beta \) with RTT)

A.2. Pass #2: Overload Details

Figure 6 repeats the dequeue function of Figure 4, but with overload details added. Similarly Figure 7 repeats the core PI algorithm of Figure 5 with overload details added. The initialization and enqueue functions are unchanged.

In line 7 of the initialization function (Figure 2), the default maximum Classic drop probability \( p_{Cmax} = 1/4 \) or 25%. This is the point at which it is deemed that the Classic queue has become persistently overloaded, so it switches to using solely drop, even for ECN-capable packets. This protects the queue against any unresponsive traffic that falsely claims that it is responsive to ECN marking, as required by [RFC3168] and [RFC7567].

Line 21 of the initialization function translates this into a maximum L4S marking probability \( (p_{Lmax}) \) by rearranging Equation (1). With a coupling factor of \( k=2 \) (the default) or greater, this translates to a maximum L4S marking probability of 1 (or 100%). This is intended to ensure that the L4S queue starts to introduce dropping once marking saturates and can rise no further. The ‘TCP Prague’ requirements [I-D.ietf-tsvwg-ecn-l4s-id] state that, when an L4S congestion
control detects a drop, it falls back to a response that coexists with 'Classic' TCP. So it is correct that the L4S queue drops packets proportional to $p^2$, as if they are Classic packets.

Both these switch-overs are triggered by the tests for overload introduced in lines 4b and 12b of the dequeue function (Figure 6). Lines 8c to 8g drop L4S packets with probability $p^2$. Lines 8h to 8i mark the remaining packets with probability $p_{CL}$.

Lines 2c to 2d in the core PI algorithm (Figure 7) deal with overload of the L4S queue when there is no Classic traffic. This is necessary, because the core PI algorithm maintains the appropriate drop probability to regulate overload, but it depends on the length of the Classic queue. If there is no Classic queue the naive algorithm in Figure 5 drops nothing, even if the L4S queue is overloaded - so tail drop would have to take over (lines 3 and 4 of Figure 3).

If the test at line 2a finds that the Classic queue is empty, line 2d measures the current queue delay using the L4S queue instead. While the L4S queue is not overloaded, its delay will always be tiny compared to the target Classic queue delay. So $p_L$ will be driven to zero, and the L4S queue will naturally be governed solely by threshold marking (lines 5 and 6 of the dequeue algorithm in Figure 6). But, if unresponsive L4S source(s) cause overload, the DualQ transitions smoothly to L4S marking based on the PI algorithm. And as overload increases, it naturally transitions from marking to dropping by the switch-over mechanism already described.
1:  dualpi2_dequeue(lq, cq) { % Couples L4S & Classic queues, lq & cq
2:    while ( lq.len() + cq.len() > 0 )
3:      if ( scheduler() == lq ) {
4a:       lq.dequeue(pkt)
4b:       if ( p_CL < p_Lmax ) { % Check for overload saturation
5:          if ( ((lq.time() > T_time) % step marking ...
6:                AND (lq.len > T_len))
7:              OR (p_CL > rand()) ) % ...or linear marking
8a:            mark(pkt)
8b:        } else { % overload saturation
8c:           if ( p_C > rand() ) {
8d:              drop(pkt) % revert to Classic drop due to overload
8e:           } continue % continue to the top of the while loop
8g:        }
8h:       if ( p_CL > rand() ) % probability p_CL = k * p
8i:         mark(pkt) % linear marking of remaining packets
8j:     }
9:   } else {
10:    cq.dequeue(pkt)
11:      if ( p_C > rand() ) { % probability p_C = p^2
12a:        if ( (ecn(pkt) == 0) % ECN field = not-ECT
12b:           OR (p_C >= p_Cmax) ) { % Overload disables ECN
13:             drop(pkt) % squared drop, redo loop
14:           } continue % continue to the top of the while loop
15:        }
16:     mark(pkt) % squared mark
17:   }
18: }
19: return(pkt) % return the packet and stop
20: }
21: return(NULL) % no packet to dequeue
22: }

Figure 6: Example Dequeue Pseudocode for DualQ Coupled PI2 AQM
(Including Integer Arithmetic and Overload Code)
1: dualpi2_update(lq, cq, target) { % Update p every Tupdate
2a:   if ( cq.len() > 0 )
2b:     curq = cq.time() % use queuing time of first-in Classic packet
2c:   else % Classic queue empty
2d:     curq = lq.time() % use queuing time of first-in L4S packet
3:    p = p + alpha_U * (curq - target) + beta_U * (curq - prevq)
4:    p_CL = p * k % Coupled L4S prob = base prob * coupling factor
5:    p_C = p^2 % Classic prob = (base prob)^2
6:    prevq = curq
7: }

Figure 7: Example PI-Update Pseudocode for DualQ Coupled PI2 AQM
(Including Overload Code)

The choice of scheduler technology is critical to overload protection
(see Section 4.1).

- A well-understood weighted scheduler such as weighted round robin
  (WRR) is recommended. The scheduler weight for Classic should be
  low, e.g. 1/16.

- Alternatively, a time-shifted FIFO could be used. This is a very
  simple scheduler, but it does not fully isolate latency in the L4S
  queue from uncontrolled bursts in the Classic queue. It works by
  selecting the head packet that has waited the longest, biased
  against the Classic traffic by a time-shift of tshift. To
  implement time-shifted FIFO, the "if (scheduler() == lq )" test in
  line 3 of the dequeue code would simply be replaced by "if (
  lq.time() + tshift >= cq.time() )". For the public Internet a
  good value for tshift is 50ms. For private networks with smaller
  diameter, about 4*target would be reasonable.

- A strict priority scheduler would be inappropriate, because it
  would starve Classic if L4S was overloaded.

Appendix B. Example DualQ Coupled Curvy RED Algorithm

As another example of a DualQ Coupled AQM algorithm, the pseudocode
below gives the Curvy RED based algorithm we used and tested.
Although we designed the AQM to be efficient in integer arithmetic,
to aid understanding it is first given using real-number arithmetic.
Then, one possible optimization for integer arithmetic is given, also
in pseudocode. To aid comparison, the line numbers are kept in step
between the two by using letter suffixes where the longer code needs
extra lines.
1: dualq_dequeue(lq, cq) { % Couples L4S & Classic queues, lq & cq
2:   if ( lq.dequeue(pkt) ) { 
3a:     p_L = cq.sec() / 2^S_L
3b:   if ( lq.byt() > T )
3c:     mark(pkt)
3d:   elif ( p_L > maxrand(U) )
4:     mark(pkt)
5:   return(pkt) % return the packet and stop here
6: }
7:   while ( cq.dequeue(pkt) ) {
8a:     alpha = 2^(-f_C)
8b:     Q_C = alpha * pkt.sec() + (1-alpha) * Q_C % Classic Q EWMA
9a:     sqrt_p_C = Q_C / 2^S_C
9b:   if ( sqrt_p_C > maxrand(2*U) )
10:       drop(pkt) % Squared drop, redo loop
11:   else
12:     return(pkt) % return the packet and stop here
13: }
14:   return(NULL) % no packet to dequeue
15: }
16: maxrand(u) { % return the max of u random numbers
17:   maxr=0
18:   while (u-- > 0)
19:     maxr = max(maxr, rand()) % 0 <= rand() < 1
20:   return(maxr)
21: }

Figure 8: Example Dequeue Pseudocode for DualQ Coupled Curvy RED AQM

Packet classification code is not shown, as it is no different from Figure 3. Potential classification schemes are discussed in Section 2.3. The Curvy RED algorithm has not been maintained to the same degree as the DualPI2 algorithm. Some ideas used in DualPI2 would need to be translated into Curvy RED, such as i) the conditional priority scheduler instead of strict priority ii) the time-based L4S threshold; iii) turning off ECN as overload protection; iv) Classic ECN support. These are not shown in the Curvy RED pseudocode, but would need to be implemented for production. (ToDo)

At the outer level, the structure of dualq_dequeue() implements strict priority scheduling. The code is written assuming the AQM is applied on dequeue (Note 1). Every time dualq_dequeue() is called, the if-block in lines 2-6 determines whether there is an L4S packet to dequeue by calling lq.dequeue(pkt), and otherwise the while-block in lines 7-13 determines whether there is a Classic packet to dequeue, by calling cq.dequeue(pkt). (Note 2)
In the lower priority Classic queue, a while loop is used so that, if the AQM determines that a classic packet should be dropped, it continues to test for classic packets deciding whether to drop each until it actually forwards one. Thus, every call to dualq_dequeue() returns one packet if at least one is present in either queue, otherwise it returns NULL at line 14. (Note 3)

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

L4S: If the test at line 2 determines there is an L4S packet to dequeue, the tests at lines 3a and 3c determine whether to mark it. The first is a simple test of whether the L4S queue (lq.byt() in bytes) is greater than a step threshold T in bytes (Note 4). The second test is similar to the random ECN marking in RED, but with the following differences: i) the marking function does not start with a plateau of zero marking until a minimum threshold, rather the marking probability starts to increase as soon as the queue is positive; ii) marking depends on queuing time, not bytes, in order to scale for any link rate without being reconfigured; iii) marking of the L4S queue does not depend on itself, it depends on the queuing time of the _other_ (Classic) queue, where cq.sec() is the queuing time of the packet at the head of the Classic queue (zero if empty); iv) marking depends on the instantaneous queuing time (of the other Classic queue), not a smoothed average; v) the queue is compared with the maximum of U random numbers (but if U=1, this is the same as the single random number used in RED).

Specifically, in line 3a the marking probability p_L is set to the Classic queueing time qc.sec() in seconds divided by the L4S scaling parameter 2^S_L, which represents the queuing time (in seconds) at which marking probability would hit 100%. Then in line 3d (if U=1) the result is compared with a uniformly distributed random number between 0 and 1, which ensures that marking probability will linearly increase with queueing time. The scaling parameter is expressed as a power of 2 so that division can be implemented as a right bit-shift (>>) in line 3 of the integer variant of the pseudocode (Figure 9).

Classic: If the test at line 7 determines that there is at least one Classic packet to dequeue, the test at line 9b determines whether to drop it. But before that, line 8b updates Q_C, which is an exponentially weighted moving average (Note 5) of the queuing time in the Classic queue, where pkt.sec() is the instantaneous queueing time of the current Classic packet and alpha is the EWMA constant for the classic queue. In line 8a, alpha is represented as an integer power of 2, so that in line 8 of the integer code
the division needed to weight the moving average can be
implemented by a right bit-shift (>> f_C).

Lines 9a and 9b implement the drop function. In line 9a the
averaged queuing time Q_C is divided by the Classic scaling
parameter 2^S_C, in the same way that queuing time was scaled for
L4S marking. This scaled queuing time is given the variable name
sqrt_p_C because it will be squared to compute Classic drop
probability, so before it is squared it is effectively the square
root of the drop probability. The squaring is done by comparing
it with the maximum out of two random numbers (assuming U=1).
Comparing it with the maximum out of two is the same as the
logical 'AND' of two tests, which ensures drop probability rises
with the square of queuing time (Note 6). Again, the scaling
parameter is expressed as a power of 2 so that division can be
implemented as a right bit-shift in line 9 of the integer
pseudocode.

The marking/dropping functions in each queue (lines 3 & 9) are two
cases of a new generalization of RED called Curvy RED, motivated as
follows. When we compared the performance of our AQM with fq_CoDel
and PIE, we came to the conclusion that their goal of holding queuing
delay to a fixed target is misguided [CRED_Insights]. As the number
of flows increases, if the AQM does not allow TCP to increase queuing
delay, it has to introduce abnormally high levels of loss. Then loss
rather than queuing becomes the dominant cause of delay for short
flows, due to timeouts and tail losses.

Curvy RED constrains delay with a softened target that allows some
increase in delay as load increases. This is achieved by increasing
drop probability on a convex curve relative to queue growth (the
square curve in the Classic queue, if U=1). Like RED, the curve hugs
the zero axis while the queue is shallow. Then, as load increases,
it introduces a growing barrier to higher delay. But, unlike RED, it
requires only one parameter, the scaling, not three. The disadvantage
of Curvy RED is that it is not adapted to a wide range of RTTs.
Curvy RED can be used as is when the RTT range to support is limited
otherwise an adaptation mechanism is required.

There follows a summary listing of the two parameters used for each
of the two queues:

Classic:

    S_C : The scaling factor of the dropping function scales Classic
          queuing times in the range [0, 2^(S_C)] seconds into a dropping
          probability in the range [0,1]. To make division efficient, it
          is constrained to be an integer power of two;
To smooth the queuing time of the Classic queue and make multiplication efficient, we use a negative integer power of two for the dimensionless EWMA constant, which we define as $\alpha = 2^{-f_C}$.

As for the Classic queue, the scaling factor of the L4S marking function scales Classic queueing times in the range $[0, 2^{S_L}]$ seconds into a probability in the range $[0,1]$. Note that $S_L = S_C + k'$, where $k'$ is the coupling between the queues. So $S_L$ and $k'$ count as only one parameter; $k'$ is related to $k$ in Equation (1) (Section 2.1) by $k=2^k'$, where both $k$ and $k'$ are constants. Then implementations can avoid costly division by shifting $p_L$ by $k'$ bits to the right.

The queue size in bytes at which step threshold marking starts in the L4S queue.

(ToDo: These are the raw parameters used within the algorithm. A configuration front-end could accept more meaningful parameters and convert them into these raw parameters.)

From our experiments so far, recommended values for these parameters are: $S_C = -1$; $f_C = 5$; $T = 5 \times$ MTU for the range of base RTTs typical on the public Internet. [CRED_Insights] explains why these parameters are applicable whatever rate link this AQM implementation is deployed on and how the parameters would need to be adjusted for a scenario with a different range of RTTs (e.g. a data centre) (ToDo incorporate a summary of that report into this draft). The setting of $k$ depends on policy (see Section 2.5 and Appendix C respectively for its recommended setting and guidance on alternatives).

There is also a cUrviness parameter, $U$, which is a small positive integer. It is likely to take the same hard-coded value for all implementations, once experiments have determined a good value. We have solely used $U=1$ in our experiments so far, but results might be even better with $U=2$ or higher.

Note that the dropping function at line 9 calls maxrand($2U$), which gives twice as much curviness as the call to maxrand($U$) in the marking function at line 3. This is the trick that implements the square rule in equation (1) (Section 2.1). This is based on the fact that, given a number $X$ from 1 to 6, the probability that two dice throws will both be less than $X$ is the square of the probability that one throw will be less than $X$. So, when $U=1$, the L4S marking function is linear and the Classic dropping function is squared. If
U=2, L4S would be a square function and Classic would be quartic. And so on.

The maxrand(u) function in lines 16-21 simply generates u random numbers and returns the maximum (Note 7). Typically, maxrand(u) could be run in parallel out of band. For instance, if U=1, the Classic queue would require the maximum of two random numbers. So, instead of calling maxrand(2*U) in-band, the maximum of every pair of values from a pseudorandom number generator could be generated out-of-band, and held in a buffer ready for the Classic queue to consume.

```c
1: dualq_dequeue(lq, cq) { % Couples L4S & Classic queues, lq & cq
2:     if ( lq.dequeue(pkt) ) {
3:        if ((lq.byt() > T) || ((cq.ns() >> (S_L-2)) > maxrand(U)))
4:           mark(pkt)
5:        return(pkt) % return the packet and stop here
6:     }
7:     while ( cq.dequeue(pkt) ) {
8:         Q_C += (pkt.ns() - Q_C) >> f_C           % Classic Q EWMA
9:        if ( (Q_C >> (S_C-2) ) > maxrand(2*U) )
10:          drop(pkt)                     % Squared drop, redo loop
11:        else
12:          return(pkt)           % return the packet and stop here
13:     }
14:    return(NULL)                           % no packet to dequeue
15: }
```

Figure 9: Optimised Example Dequeue Pseudocode for Coupled DualQ AQM using Integer Arithmetic

Notes:

1. The drain rate of the queue can vary if it is scheduled relative to other queues, or to cater for fluctuations in a wireless medium. To auto-adjust to changes in drain rate, the queue must be measured in time, not bytes or packets [CoDel]. In our Linux implementation, it was easiest to measure queuing time at dequeue. Queuing time can be estimated when a packet is enqueued by measuring the queue length in bytes and dividing by the recent drain rate.

2. An implementation has to use priority queueing, but it need not implement strict priority.

3. If packets can be enqueued while processing dequeue code, an implementer might prefer to place the while loop around both queues so that it goes back to test again whether any L4S packets arrived while it was dropping a Classic packet.
4. In order not to change too many factors at once, for now, we keep the marking function for DCTCP-only traffic as similar as possible to DCTCP. However, unlike DCTCP, all processing is at dequeue, so we determine whether to mark a packet at the head of the queue by the byte-length of the queue _behind_ it. We plan to test whether using queuing time will work in all circumstances, and if we find that the step can cause oscillations, we will investigate replacing it with a steep random marking curve.

5. An EWMA is only one possible way to filter bursts; other more adaptive smoothing methods could be valid and it might be appropriate to decrease the EWMA faster than it increases.

6. In practice at line 10 the Classic queue would probably test for ECN capability on the packet to determine whether to drop or mark the packet. However, for brevity such detail is omitted. All packets classified into the L4S queue have to be ECN-capable, so no dropping logic is necessary at line 3. Nonetheless, L4S packets could be dropped by overload code (see Section 4.1).

7. In the integer variant of the pseudocode (Figure 9) real numbers are all represented as integers scaled up by $2^{32}$. In lines 3 & 9 the function maxrand() is arranged to return an integer in the range $0 \leq \text{maxrand()} < 2^{32}$. Queuing times are also scaled up by $2^{32}$, but in two stages: i) In lines 3 and 8 queuing times $\text{cq.ns()}$ and $\text{pkt.ns()}$ are returned in integer nanoseconds, making the values about $2^{30}$ times larger than when the units were seconds, ii) then in lines 3 and 9 an adjustment of $-2$ to the right bit-shift multiplies the result by $2^2$, to complete the scaling by $2^{32}$.

Appendix C. Guidance on Controlling Throughput Equivalence

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

Table 1: Value of $k'$ for which DCTCP throughput is roughly the same as Reno or Cubic, for some example RTT ratios

$k'$ is related to $k$ in Equation (1) (Section 2.1) by $k = 2^{k'}$. 

To determine the appropriate policy, the operator first has to judge whether it wants DCTCP flows to have roughly equal throughput with Reno or with Cubic (because, even in its Reno-compatibility mode, Cubic is about 1.4 times more aggressive than Reno). Then the operator needs to decide at what ratio of RTTs it wants DCTCP and Classic flows to have roughly equal throughput. For example choosing $k'=0$ (equivalent to $k=1$) will make DCTCP throughput roughly the same as Cubic, _if their RTTs are the same_.

However, even if the base RTTs are the same, the actual RTTs are unlikely to be the same, because Classic (Cubic or Reno) traffic needs a large queue to avoid under-utilization and excess drop, whereas L4S (DCTCP) does not. The operator might still choose this policy if it judges that DCTCP throughput should be rewarded for keeping its own queue short.

On the other hand, the operator will choose one of the higher values for $k'$, if it wants to slow DCTCP down to roughly the same throughput as Classic flows, to compensate for Classic flows slowing themselves down by causing themselves extra queuing delay.

The values for $k'$ in the table are derived from the formulae, which was developed in [DCttH15]:

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

For localized traffic from a particular ISP’s data centre, we used the measured RTTs to calculate that a value of $k'=3$ (equivalent to $k=8$) would achieve throughput equivalence, and our experiments verified the formula very closely.

For a typical mix of RTTs from local data centres and across the general Internet, a value of $k'=1$ (equivalent to $k=2$) is recommended as a good workable compromise.

**Appendix D. Open Issues**

Most of the following open issues are also tagged '{ToDo}' at the appropriate point in the document:

- Operational guidance to monitor L4S experiment
- PI2 appendix: scaling of alpha & beta, esp. dependence of beta_U on Tupdate
- Curvy RED appendix: complete the unfinished parts
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/

Olga Bondarenko
Simula Research Lab
Lysaker
Norway

Email: olgabnd@gmail.com
URI: https://www.simula.no/people/olgabo

Ing-jyh Tsang
Nokia
Antwerp
Belgium

Email: ing-jyh.tsang@nokia.com
Packetization Layer Path MTU Discovery for Datagram Transports
draft-ietf-tsvwg-datagram-plpmtud-05

Abstract

This document describes a robust method for Path MTU Discovery (PMTUD) for datagram Packetization Layers (PLs). The document 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, and no ICMP message is received). The method can also probe a network path with progressively larger packets to find 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."
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 5, 2019.

Copyright Notice

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

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

Table of Contents

1. Introduction ............................................. 3
   1.1. Classical Path MTU Discovery ........................ 4
   1.2. Packetization Layer Path MTU Discovery .............. 5
   1.3. Path MTU Discovery for Datagram Services .......... 6
2. Terminology ............................................. 7
3. Features Required to Provide Datagram PLPMTUD .......... 9
4. DPLPMTUD Mechanisms .................................... 11
   4.1. PLPMTU Probe Packets ................................ 11
   4.2. Confirmation of Probed Packet Size ................. 13
   4.3. Detection of Black Holes ........................... 13
   4.4. Response to PTB Messages ............................ 14
       4.4.1. Validation of PTB Messages ..................... 14
       4.4.2. Use of PTB Messages ............................ 15
5. Datagram Packetization Layer PMTUD .................... 16
   5.1. DPLPMTUD Components ............................... 17
       5.1.1. Timers ....................................... 17
       5.1.2. Constants .................................... 17
       5.1.3. Variables ................................... 18
   5.2. DPLPMTUD Phases .................................... 19
       5.2.1. Path Confirmation Phase ....................... 20
       5.2.2. Search Phase .................................. 21
           5.2.2.1. Resilience to inconsistent path information  21
       5.2.3. Search Complete Phase ......................... 21
       5.2.4. PROBE_BASE Phase ............................. 22
       5.2.5. ERROR Phase .................................. 22
           5.2.5.1. Robustness to inconsistent path .......... 23
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]). The term PTB message is applied to both IPv4 ICMP Unreachable messages (Type 3) that carry the error Fragmentation Needed (Type 3, Code 4) and ICMPv6 packet too big messages (Type 2). When a sender receives a PTB message, it reduces the effective MTU to the value reported in the PTB message (in this document called the PTB_SIZE). A method from time-to-time increases the packet size in attempt to discover an increase in the supported PMTU. The packets sent with a size larger than the current effective PMTU are known as probe packets.

Packets not intended as probe packets are either fragmented to the current effective PMTU, or an attempt to send a packet larger than current effective PMTU fails with an error code. Applications are sometimes provided with a primitive to let them read the maximum packet size, 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 silently discarded without the sender receiving ICMP PTB messages). This could arise when the PTB messages are not delivered back to the sender for some reason [RFC2923]). For example, ICMP messages are increasingly 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 endpoints behind this firewall. Other examples include cases where PTB messages are not correctly processed/generated by tunnel endpoints.

Another failure could result if a node that is not on the network path sends a PTB message that attempts to force the 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 the router issuing the ICMP message is acting on a tunneled packet, the ICMP message will be directed to the tunnel endpoint. 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. Failure to do appropriate processing therefore results in black-holing.

When a router issuing the ICMP message implements RFC 792 [RFC0792], it is only required to include (quote) the first 64 bits of the IP payload of the packet within the ICMP payload. This could be insufficient to perform the tunnel processing described in the previous bullet. Even if the decapsulated message is processed by the tunnel endpoint, there could be insufficient bytes remaining for the sender to interpret the quoted transport information. RFC 1812 [RFC1812] requires routers to return the full packet if possible. This can result in black-holing when used the path includes tunnels.

When a router issuing the ICMP message quotes a packet with an encrypted transport, it may lack sufficient context to determine the original transport header.

Even when the PTB message includes sufficient bytes of the quoted packet, the network layer could lack sufficient context to validate the ICMP message, because this 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).

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 Maximum Packet Size (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. The 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. This 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 PTB
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 of increased black-holing. For instance, the
information available at the PL, or higher layers, makes PTB
validation more straightforward.

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 utilise ICMP 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 utilises heartbeat
messages as probe packets, 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 Holed: Packets are Black holed when the sender is unaware that 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 silently discarded by the network, but is not made aware of a change to the path that resulted in a smaller PLPMTU by ICMP messages).

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.

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

**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 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 PLMPMTU. This is used to send a probe packet. In IPv4, a probe packet MUST be sent with the Don’t Fragment (DF) bit set in the IP header, and without network layer endpoint fragmentation. In IPv6, a probe packet is always sent without source fragmentation (as specified in section 5.4 of [RFC8201]).

4. Processing PTB messages: A DPLPMTUD sender MAY optionally utilize PTB messages received from the network layer to help identify when a network path does not support the current size of probe packet. Any received PTB message MUST be validated before it is used to update the PLPMTU discovery information [RFC8201]. This validation confirms that the PTB message was sent in response to a packet originating by the sender, and needs to be performed.
before the PLPMTU discovery method reacts to the PTB message. When the PTB_SIZE is indicated in the PTB message, this MAY be used by DPLPMTUD to reduce the probe size but MUST NOT be used to increase the PLPMTU ([RFC8201]). This validation SHOULD utilise information that can not be simply determined by an off-path attacker, for example, by checking the value of a protocol header field known only to the two PL endpoints. (Some datagram applications use well-known source and destination ports and therefore this check needs to rely on other information.)

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-side DPLPMTUD method.

6. Probing and congestion control: The isolated loss of a probe packet SHOULD NOT be treated as an indication of congestion and its loss SHOULD NOT directly trigger a congestion control reaction [RFC4821].

7. Probe loss recovery: If the data block carried by a probe packet needs to be sent reliably, the PL (or layers above) are REQUIRED to arrange any retransmission/repair of any resulting loss. This method is REQUIRED to be robust in the case where probe packets are lost due to other reasons (including link transmission error, congestion). The DPLPMTUD sender treats isolated loss of a probe packet (with or without an PTB message) as a potential indication of a PMTU limit for the path, but not as an indication of congestion, see Paragraph 6.

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 and make the required congestion control adjustments". Such methods MUST be robust to the wide variety of underlying network forwarding behaviours, 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 capacity has increased since it last measured the path. This determines when the path should again be probed.

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 utilise 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 utilise 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 ICMP 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 5 specifies this function for a set of IETF-specified protocols.

4.3. Detection of Black Holes

A PL sender needs to reduce the PLPMTU when it discovers the actual PMTU supported by a network path is less than the PLPMTU (i.e., to detect that traffic is being black holed). 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.

Black Hole detection is performed by periodically sending packet probes of size PLPMTU to verify that a network path still supports the last acknowledged PLPMTU size. There are two ways a DPLPMTUD sender detect that the current PLPMTU is not sustained by the path (i.e., to detect a black hole):

- A PL can rely upon a mechanisms implemented within the PL protocol 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 probing mechanism to send confirmation probe packets of the size of the current PLPMTU and a timer track whether acknowledgments are received (e.g., The number of probe packets sent without receiving an acknowledgement, PROBE_COUNT, becomes greater than the MAX_PROBES). These messages need to be
generated periodically (e.g., using the confirmation timer Section 5.1.1), and should be suppressed when the PL is not actively sending data. Successive loss of probes is an indication that the current path no longer supports the PLPMTU.

When the method detects the current PLPMTU is not supported (a black hole is found), DPLPMTUD sets a lower MPS. The PL then confirms that the updated PLPMTU can be successfully used across the path. This can need the PL 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 could instead utilise a control packet with padding.

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

A PL that receives a PTB message from a router or middlebox, MUST perform ICMP validation as specified in Section 5.2 of [RFC8085]. This needs the PL to check the protocol information in the quoted payload to validate the message originated from the sending node. This check includes determining the appropriate port and IP information - necessary for the PTB message to be passed to the PL. In addition, the PL SHOULD validate information from the ICMP payload to determine that the quoted packet was sent by the PL. These checks are intended to provide protection from packets that originate from a node that is not on the network path. PTB messages are discarded if they fail to pass these checks, or where there is insufficient ICMP payload to perform the checks.

PTB messages that have been validated can be utilised by the DPLPMTUD algorithm. A method that utilises these PTB messages can improve the speed at which the algorithm detects an appropriate PLPMTU, compared to one that relies solely on probing.
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 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 an informative summary of how PTB messages can be utilised.

Validating PTB Messages:

* A simple implementation is permitted to ignore received PTB messages and therefore the PLPMTU is not updated when a PTB message is received.
* An implementation that supports PTB messages MUST validate messages before they are processed.

MIN_PMTU < PTB_SIZE < BASE_MTU

* A robust PL MAY enter the PROBE_ERROR state for an IPv4 path when the PTB_SIZE reported in the PTB message >= 576B and when this is less than the BASE_MTU.
* A robust PL MAY enter the PROBE_ERROR state for an IPv6 path when the PTB_SIZE reported in the PTB message >= 1280B and when this is less than the BASE_MTU.

PTB_SIZE = PLPMTU

* Transition to SEARCH_COMPLETE.

PTB_SIZE > PROBED_SIZE

* The PTB_SIZE > PROBED_SIZE, inconsistent network signal. These PTB messages ought to be discarded without further processing (the PLPMTU not updated).
* The information could be utilised 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 PTB_SIZE reported in the PTB message to initialise 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.

5. Datagram Packetization Layer PMTUD

This section specifies Datagram PLPMTUD (DPLPMTUD). The method can be introduced at various points in the IP protocol stack to discover the PLPMTU so that an application can utilise an appropriate MPS for the current network path.

```
+----------------------+
|         APP*         |
+-------+----+---+---+
|       |    |   |
+---+--+ +--+--+ | +-+---+
| QUIC*| |UDPO*| | |SCTP*|
+---+--+ +--+--+ | ++--+-+
|       |    |  |  |
+-------+-+  |  |  |
|  |  |  |
++-+--++  |
| UDP  |
+-+--+
|     |
+--------------+-----+-+
|  Network Interface   |
+----------------------+
```

Figure 1: Examples where DPLPMTUD can be implemented

The central idea of DPLPMTUD is probing by a sender. Probe packets are sent to find the maximum size of user message that is completely transferred across the network path from the sender to the destination.

This section identifies the components needed for implementation, the phases of operation, the state machine and search algorithm.
5.1. DPLPMTUD Components

This section describes components of DPLPMTUD.

5.1.1. Timers

The method utilises 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 be larger 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 SHOULD inhibit sending probe packets when no application data has been sent since the previous probe packet.

CONFIRMATION_TIMER: 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 SHOULD inhibit sending probe packets when no application data has been sent since the previous probe packet.

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

5.1.2. Constants

The following constants are defined:

MAX_PROBES: MAX_PROBES is the maximum value of the PROBE_ERROR_COUNTER. The default value of MAX_PROBES is 10.
MIN_PMTU: The MIN_PMTU is smallest allowed probe packet size. For IPv6, this value is 1280 bytes, as specified in [RFC2460]. For IPv4, the minimum value is 68 bytes. (An IPv4 router is required to be able to forward a datagram of 68 octets without further fragmentation. This is the combined size of an IPv4 header and the minimum fragment size of 8 octets. In addition, receivers are required to be able to reassemble fragmented datagrams at least up to 576B, 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 utilises 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 initialised 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, in this case when the DPLPMTUD algorithm performs path probing to increase the size of the PLPMTU. The MPS is less than the PLPMTU. A probe packet has been sent of size PROBED_SIZE. When this is acknowledged, the PLPMTU will be raised to PROBED_SIZE allowing the PROBED_SIZE to be increased towards the actual PMTU.
5.2. DPLPMTUD Phases

The Datagram PLPMTUD algorithm moves through several phases of operation.

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).

Black hole detection, see Section 4.3 and PTB processing Section 4.4 proceed in parallel with these phases of operation.

---

Figure 3: DPLPMTUD Phases
Path Confirmation

* Connectivity is confirmed.
* DPLPMTUD confirms the BASE_PMTU is supported across the network path.
* DPLPMTUD then enters the search phase.

Search

* DPLPMTUD performs probing to increase the PLPMTU.
* DPLPMTUD then enters the search complete or an error phase.

Search Complete

* DPLPMTUD has found a suitable PLPMTU that is supported across the network path.
* Black hole detection will confirm this PLPMTU continues to be supported.
* On a longer time-frame, DPLPMTUD will re-enter the search phase to discover if the PLPMTU can be raised.

Error

* Inconsistent or invalid network signals cause DPLPMTUD to be unable to progress.
* This causes the algorithm to lower the MPS until the path is shown to support the BASE_PMTU, or to suspend DPLPMTUD.

5.2.1. Path Confirmation Phase

DPLPMTUD starts in the Path confirmation phase. Path confirmation is performed in two stages:

1. Connectivity to the remote peer is first confirmed. When a connection-oriented PL is used, this stage is implicit. It is performed as part of the normal PL connection handshake. In contrast, an connectionless PL MUST send an acknowledged probe packet to confirm that the remote peer is reachable.

2. In the second stage, the PL confirms it can successfully send a datagram of the BASE_PMTU size across the current path.
A PL that does not wish to support a network path with a PLPMTU less than BASE_PMTU can simplify the phase into a single step by performing connectivity checks with probes of the BASE_PMTU size.

A PL MAY respond to PTB messages while in this phase, see Section 4.4.

Once path confirmation has completed, DPLPMTUD can advertise an MPS to an upper layer.

If DPLPMTUD fails to complete these tests it enters the PROBE_DISABLED phase, see Section 5.2.6, and ceases using DPLPTMUD.

5.2.2. Search Phase

The search phase utilises a search algorithm in attempt to increase the PLPMTU (see Section 5.4.1). The PL sender increases the MPS each time a packet probe confirms a larger PLPMTU is supported by the path. The algorithm concludes by entering the SEARCH_COMPLETE phase, see Section 5.2.3.

A PL MAY respond to PTB messages while in this phase, using the PTB to advance or terminate the search, see Section 4.4. Similarly black hole detection can terminate the search by entering the PROBE_BASE phase, see Section 5.2.4.

5.2.2.1. Resilience to inconsistent path information

Sometimes a PL sender is able to detect inconsistent results from the sequence of PLPMTU probes that it sends or the sequence of PTB messages that it receives. This could be manifested as excessive fluctuation of the MPS.

When inconsistent path information is detected, a PL sender can enable an alternate search mode that clamps the offered MPS to a smaller value for a period of time. This avoids unnecessary black-holing of packets.

5.2.3. Search Complete Phase

On entry to the search complete phase, the DPLPMTUD sender starts the PMTU_RAISE_TIMER. In this phase, the PLPMTU remains at the value confirmed by the last successful probe packet.

In this phase, the PL MUST periodically confirm that the PLPMTU is still supported by the path. If the PL is designed in a way that is unable to confirm reachability to the destination endpoint after
If the DPLPMTUD sender is unable to confirm reachability for packets with a size of the current PLPMTU (e.g., if the CONFIRMATION_TIMER expires) or the PL signals a lack of reachability, the method exits the phase and enters the PROBE_BASE phase, see Section 5.2.4.

If the PMTU_RAISE_TIMER expires, the DPLPMTUD sender re-enters the Search phase, see Section 5.2.2, and resumes probing for a larger PLPMTU.

Back hole detection can be used in parallel to check that a network path continues to support a previously confirmed PLPMTU. If a black hole is detected the algorithm moves to the PROBE_BASE phase, see Section 5.2.4.

The phase can also exited when a validated PTB message is received (see Section 4.4.1).

5.2.4. PROBE_BASE Phase

This phase is entered when black hole detection or a PTB message indicates that the PLPMTU is not supported by the path.

On entry to this phase, the PLPMTU is set to the BASE_PMTU, and a corresponding reduced MPS is advertised.

PROBED_SIZE is then set to the PLPMTU (i.e., the BASE_PMTU), to confirm this size is supported across the path. If confirmed, DPLPMTUD enters the Search Phase to determine whether the PL sender can use a larger PLPMTU.

If the path cannot be confirmed to support the BASE_PMTU after sending MAX_PROBES, DPLPMTUD moves to the Error phase, see Section 5.2.5.

5.2.5. 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). In this phase, the MPS is set to a value less than the BASE_PMTU, but at least the size of the MIN_PMTU.

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.
Note: MIN_PMTU may be identical to BASE_PMTU, simplifying the actions in this phase.

If no acknowledgement is received for PROBE_COUNT probes of size MIN_PMTU, the method suspends DPLPMTUD, see Section 5.2.5.

5.2.5.1. Robustness to inconsistent path

Robustness to paths unable to sustain the BASE_PMTU. Some paths could be unable to sustain packets of the BASE_PMTU size. These paths could use an alternate algorithm to implement the PROBE_ERROR phase that allows fallback to a smaller than desired PLPMTU, rather than suffer connectivity failure.

This could also utilise methods such as endpoint IP fragmentation to enable the PL sender to communicate using packets smaller than the BASE_PMTU.

5.2.6. DISABLED Phase

This phase suspends operation of DPLPMTUD. It disables probing for the PLPMTU until action is taken by the PL or application using the PL.

5.3. State Machine

A state machine for DPLPMTUD is depicted in Figure 4. If multihoming is supported, a state machine is needed for each active path.
Figure 4: State machine for Datagram PLPMTUD. Note: Some state changes are not show to simplify the diagram.

The following states are defined:
PROBE_START: The PROBE_START state is the initial state before probing has started. The state confirms connectivity to the remote PL.

The PLPMTU is set to the BASE_PMTU size. Probing ought to start immediately after connection setup to prevent the loss of user data. PLPMTUD is not performed in this state. The state transitions to PROBE_SEARCH, when a network path has been confirmed, i.e., when a sent packet has been acknowledged on this network path and the BASE_PMTU is confirmed to be supported. If the network path cannot be confirmed this state transitions to PROBE_DISABLED.

PROBE_SEARCH: The PROBE_SEARCH 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 PTB message is validated; a probe of size PMTU_MAX is acknowledged or black hole detection is triggered.

SEARCH_COMPLETE: The SEARCH_COMPLETE state indicates a successful end to the PROBE_SEARCH state. DPLPMTUD remains in this state until either the PMTU_RAISE_TIMER expires; a received PTB message is validated; or black hole detection is triggered.

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 PROBE_BASE state. When used with an acknowledged PL (e.g., SCTP), DPLPMTUD SHOULD NOT continue to generate PLPMTU probes in this state.

PROBE_BASE: The PROBE_BASE state is used to confirm whether 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 PROBE_SIZE is set to the BASE_PMTU size and the PROBE_COUNT is set to zero.

Each time a probe packet is sent, and the PROBE_TIMER is started. The state is exited when the probe packet is acknowledged, and the PL sender enters the PROBE_SEARCH state.

The state is also left when the PROBE_COUNT reaches MAX_PROBES; a PTB message is validated. This causes the PL sender to enter the PROBE_ERROR state.

PROBE_ERROR: The PROBE_ERROR state represents the case where the network path is not known to support a PLPMTU of at least the BASE_PMTU size. It is entered when either a probe of size BASE_PMTU has not been acknowledged or a validated PTB message indicates a smaller PTB_SIZE smaller than the BASE_PMTU.

On entry, the PROBE_COUNT is set to zero and the PROBED_SIZE is set to the MIN_PMTU size, and the PLPMTU is reset to MIN_PMTU size. In this state, a probe packet is sent, and the PROBE_TIMER is started. The state transitions to the PROBE_SEARCH state when a probe packet is acknowledged of at least size BASE_PMTU. Robust implementations may validate the BASE_PMTU several times before transition to the PROBE_SEARCH.

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 PROBE_DISABLED.

PROBE_DISABLED: The PROBE_DISABLED state indicates that connectivity could not be established. DPLPMTUD MUST NOT probe in this state.

Appendix A contains an informative description of key events.

5.4. Search to Increase the PLPMTU

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

5.4.1. 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 PMTU_MAX. PMTU_MAX is the minimum of the local MTU and EMTU_R (learned from the remote endpoint). The PMTU_MAX MAY be reduced by an application that sets a maximum to the size of datagrams it will send.

The PROBE_COUNT is initialised 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 cancelled 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 reinitialised, 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.

5.4.2. 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 maximise 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.

xxx Author Note: A future version of this section will detail example methods for selecting probe size values, but does not plan to mandate a single method. xxx
5.4.3.  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 (this could happen when probe packets are lost due to other reasons, or some of the packets in a flow are forwarded along a portion of the path that supports a different actual PMTU).

Frequent path changes could occur due to unexpected "flapping" – where some packets from a flow pass along one path, but other packets follow a different path with different properties. DPLPMTUD can be made resilient to these anomalies by introducing hysteresis into the search decision to increase the MPS.


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 subsection 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 ICMP PTB messages).

In addition, it is desirable that PMTU discovery is not performed by multiple protocol layers. An application SHOULD avoid implementing DPLPMTUD when the underlying transport system provides this capability. Using a common method for managing the PLPMTU
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 initialised 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 normal 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. 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 with UDP Options

UDP Options [I-D.ietf-tsvwg-udp-options] can supply the additional functionality required to implement DPLPMTUD within the UDP transport service. Implementing DPLPMTU using UDP Options avoids the need for each application to implement the DPLPMTUD method.

Section 5.6 of [I-D.ietf-tsvwg-udp-options] defines the Maximum Segment Size (MSS) option, which allows the local sender to indicate the EMTU_R to the peer. The value received in this option can be used to initialise PMTU_MAX.

UDP Options enables padding to be added to UDP datagrams that are used as Probe Packets. Feedback confirming reception of each Probe Packet is provided by two new UDP Options:

- The Probe Request Option (Section 6.2.1) is set by a sending PL to solicit a response from a remote endpoint. A four-byte token identifies each request.
- The Probe Response Option (Section 6.2.2) is generated by the UDP Options receiver in response to reception of a previously received Probe Request Option. Each Probe Response Option echoes a previously received four-byte token.

The token value allows implementations to be distinguish between acknowledgements for initial probe packets and acknowledgements confirming receipt of subsequent probe packets (e.g., travelling along alternate paths with a larger RTT). Each probe packet needs to be uniquely identifiable by the UDP Options sender within the Maximum Segment Lifetime (MSL). The UDP Options sender therefore needs to not recycle token values until they have expired or have been acknowledged. A 4 byte value for the token field provides sufficient space for multiple unique probes to be made within the MSL.

The initial value of the four byte token field SHOULD be assigned to a randomised value, as described in section 5.1 of [RFC8085]) to enhance protection from off-path attacks.

Implementations ought to only send a probe packet with a Request Probe Option when required by their local state machine, i.e., when probing to grow the PLPMTU or to confirm the current PLPMTU. The procedure to handle the loss of a response packet is the responsibility of the sender of the request. Implementations are allowed to track multiple requests and respond to them with a single packet.
A PL needs to determine that the path can still support the size of datagram that the application is currently sending in the DPLPMTUD search_done state (i.e., to detect black-holing of data). One way to achieve this is to send probe packets of size PLPMTU or to utilise a higher-layer method that provides explicit feedback indicating any packet loss. Another possibility is to utilise data packets that carry a Timestamp Option. Reception of a valid timestamp that was echoed by the remote endpoint can be used to infer connectivity. This can provide useful feedback even over paths with asymmetric capacity and/or that carry UDP Option flows that have very asymmetric datagram rates, because an echo of the most recent timestamp still indicates reception of at least one packet of the transmitted size. This is sufficient to confirm there is no black hole.

In contrast, when sending a probe to increase the PLPMTU, a timestamp might be unable to unambiguously identify that a specific probe packet has been received. Timestamp mechanisms cannot be used to confirm the reception of individual probe messages and cannot be used to stimulate a response from the remote peer.

6.2.1. UDP Probe Request Option

The Probe Request Option allows a sending endpoint to solicit a response from a destination endpoint.

The Probe Request Option carries a four byte token set by the sender. This token can be set to a value that is likely to be known only to the sender (and is sent along the end-to-end path). The initial value of the token SHOULD be assigned to a randomised value, as described in section 5.1 of [RFC8085]) to enhance protection from off-path attacks.

The sender needs to then check the value returned in the UDP Probe Response Option. The value of the Token field, uniquely identifies a probe within the maximum segment lifetime.

```
+----------+--------+-----------------+
| Kind=9*  | Len=6  |     Token       |
+----------+--------+-----------------+
1 byte    1 byte       4 bytes
```

* To be confirmed by IANA.

Figure 5: UDP Probe REQ Option Format
6.2.2. UDP Probe Response Option

The Probe Response Option is generated in response to reception of a previously received Probe Request Option. This response is generated by the UDP Option processing.

The Probe Response Option carries a four byte token field. The Token field associates the response with the Token value carried in the most recently-received Echo Request. The rate of generation of UDP packets carrying a Probe Response Option is expected to be less than once per RTT and SHOULD be rate-limited (see Section 9).

+----------+--------+-----------------+
| Kind=10* | Len=6  |     Token       |
+----------+--------+-----------------+
 1 byte    1 byte       4 bytes

* To be confirmed by IANA.

Figure 6: UDP Probe RES Option Format

6.3. 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

6.3.1. SCTP/IPv4 and SCTP/IPv6

The base protocol is specified in [RFC4960]. This provides an acknowledged PL. A sender can therefore enter the PROBE_BASE state as soon as connectivity has been confirmed.

6.3.1.1. 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 handshake, before data is sent. Assuming normal behaviour (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.

6.3.1.2. 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.

6.3.1.3. 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.

6.3.2. DPLPMTUD for SCTP/UDP

The UDP encapsulation of SCTP is specified in [RFC6951].

6.3.2.1. Sending SCTP/UDP Probe Packets

Packet probing can be performed as specified in Section 6.3.1.1. The maximum payload is reduced by 8 bytes, which has to be considered when filling the PAD chunk.
6.3.2.2. 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.

6.3.2.3. Handling of PTB Messages by SCTP/UDP

Normal 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.

6.3.3. 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.

6.3.3.1. Sending SCTP/DTLS Probe Packets

Packet probing can be done as specified in Section 6.3.1.1.

6.3.3.2. 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.

6.3.3.3. Handling of PTB Messages by SCTP/DTLS

It is not possible to perform normal 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.

6.4. DPLPMTUD for QUIC

Quick UDP Internet Connection (QUIC) [I-D.ietf-quic-transport] is a UDP-based transport that provides reception feedback.

Section 9.2 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. This enables probing without affecting the transfer of other QUIC frames.
This provides an acknowledged PL. A sender can therefore enter the PROBE_BASE state as soon as connectivity has been confirmed.

6.4.1. Sending QUIC Probe Packets

A probe packet consists of a QUIC Header and a payload containing only PADDING Frames. 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 PROBE_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.

6.4.2. 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.

6.4.3. 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. Although QUIC does not currently specify a method for validating ICMP responses, it does provide some guidelines to make it harder for an off-path attacker to inject ICMP messages.

- Set the IPv4 Don’t Fragment (DF) bit on a small proportion of packets, so that most invalid ICMP messages arrive when there are no DF packets outstanding, and can therefore be identified as spurious.
- Store additional information from the IP or UDP headers from DF packets (for example, the IP ID or UDP checksum) to further authenticate incoming Datagram Too Big messages.
- Any reduction in PMTU due to a report contained in an ICMP packet is provisional until QUIC’s loss detection algorithm determines that the packet is actually lost.
XXX The above list was pulled whole from quic-transport - input is invited from QUIC contributors. XXX

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.

XXX If new UDP Options are specified in this document, a request to IANA will be included here. XXX

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-initialised 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 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 utilise 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).

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 PROBE_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 5.2.5.1 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]

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


10.2. Informative References


Appendix A. Event-driven state changes

This appendix contains an informative description of key events:

Path Setup: When a new path is initiated, the state is set to PROBE_START. This sends a probe packet with the size of the BASE_PMTU. As soon as the path is confirmed, the state changes to PROBE_SEARCH.

Arrival of an Acknowledgment: Depending on the probing state, the reaction differs according to Figure 7, which is a simplification of Figure 4 focusing on this event.
Condition 1: The maximum PMTU size has not yet been reached.
Condition 2: The maximum PMTU size has been reached. Condition 3:
Probe Timer expires and PROBE_COUNT = MAX_PROBES. Condition 4:
PROBE_ACK received. Condition 5: Black hole detected.

Figure 7: State changes at the arrival of an acknowledgment

Probing timeout: The PROBE_COUNT is initialised to zero each time
the value of PROBED_SIZE is changed and when a acknowledgment
confirming delivery of a probe packet. The PROBE_TIMER is started
each time a probe packet is sent. It is stopped when an
acknowledgment arrives that confirms delivery of a probe packet of
PROBED_SIZE. If the probe packet is not acknowledged before the
PROBE_TIMER expires, the PROBE_COUNT is incremented. When the
PROBE_COUNT equals the value MAX_PROBES, the state is changed,
only a new probe packet of the same size (PROBED_SIZE) is
resent. The state transitions are illustrated in Figure 8. This
shows a simplification of Figure 4 with a focus only on this
event.
Condition 1: The maximum number of probe packets has not been reached. Condition 2: The maximum number of probe packets has been reached. XXX This diagram has not been validated.

Figure 8: State changes at the expiration of the probe timer

PMTU raise timer timeout: DPLPMTUD periodically sends a probe packet to detect whether a larger PMTU is possible. This probe packet is generated by the PMTU_RAISE_TIMER.

Arrival of a PTB message: The active probing of the path can be supported by the arrival of a PTB message indicating the PTB_SIZE. Two examples are:

1. The PTB_SIZE is between the PLPMTU and the probe that triggered the PTB message.
2. The PTB_SIZE is smaller than the PLPMTU.

In first case, the PROBE_BASE state transitions to the PROBE_ERROR state. In the PROBE_SEARCH state, a new probe packet is sent with the size reported by the PTB message.

In second case, the probing starts again with a value of PROBE_BASE.
Appendix B. 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 superseding 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.

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
Identifying Modified Explicit Congestion Notification (ECN) Semantics
for Ultra-Low Queuing Delay (L4S)
draft-ietf-tsvwg-ecn-l4s-id-03

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, even during high
load. Examples of new active queue management (AQM) marking
algorithms and examples of new transports (whether TCP-like or real-
time) are specified separately. The new L4S identifier is the key
piece that enables them to interwork and distinguishes them from
‘Classic’ traffic. It gives an incremental migration path so that
existing ‘Classic’ TCP traffic will be no worse off, but it can be
prevented from degrading the ultra-low delay and loss of the new
scalable transports.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 3, 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
   1.1. Problem .................................................. 4
   1.2. Terminology .............................................. 5
   1.3. Scope ................................................... 6
2. L4S Packet Identifier ....................................... 7
   2.1. Consensus Choice of L4S Packet Identifier: Requirements ... 7
   2.2. L4S Packet Identification at Run-Time ...................... 8
   2.3. Interaction of the L4S Identifier with other Identifiers .. 8
   2.4. Pre-Requisite Transport Layer Behaviour .................. 10
      2.4.1. Pre-Requisite Congestion Response .................... 10
      2.4.2. Pre-Requisite Transport Feedback ..................... 11
   2.5. Exception for L4S Packet Identification by Network Nodes with Transport-Layer Awareness .................... 12
   2.6. The Meaning of L4S CE Relative to Drop ................... 12
3. L4S Experiments .............................................. 13
4. IANA Considerations .......................................... 13
5. Security Considerations ...................................... 13
6. Acknowledgements ............................................ 13
7. References ................................................. 13
   7.1. Normative References ..................................... 13
   7.2. Informative References .................................. 14
Appendix A. The ‘TCP Prague Requirements’ ....................... 19
   A.1. Requirements for Scalable Transport Protocols ............ 19
      A.1.1. Use of L4S Packet Identifier .......................... 19
      A.1.2. Accurate ECN Feedback ............................... 20
      A.1.3. Fall back to Reno/Cubic congestion control on packet

De Schepper & Briscoe Expires January 3, 2019 [Page 2]
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). ‘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 bit-rate 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, even during high load.

An example of an active queue management (AQM) marking algorithm that enables the L4S service is the DualQ Coupled AQM defined in a complementary specification [I-D.ietf-tsvwg-aqm-dualq-coupled]. An example of a scalable transport 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. However, AQMs like DualQ Coupled enable scalable transports like DCTCP to co-exist with...
existing traffic, each getting roughly the same flow rate when they compete under similar conditions. Note that DCTCP will still not be safe to deploy on the Internet until it satisfies the requirements listed in Section 2.4.

The new L4S identifier is the key piece that enables these two parts 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. The performance improvement is so great that it is hoped it will motivate 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. Web, voice, conversational video, gaming, finance apps, remote desktop and cloud-based applications. 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 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. 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 TCPs 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 TCP’s scalability problem also solves the latency problem, because the finer sawteeth cause very little queuing delay. 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 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).

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 2.3 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:

- updates the ECN proposed standard [RFC3168] (in certain specified cases including the present document) to relax the requirement that an ECN mark must be equivalent to a drop, both when applied by the network, and when responded to by the sender;
- changes the status of the experimental ECN nonce [RFC3540] to historic;
- makes consequent updates to the following 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.  L4S Packet Identifier

2.1.  Consensus Choice of L4S Packet Identifier: Requirements

This subsection briefly records the process that led to a consensus choice of L4S identifier, selected from all the alternatives in Appendix B.

Ideally, the identifier for packets using the Low Latency, Low Loss, Scalable throughput (L4S) service ought to meet the following requirements:

- it SHOULD survive end-to-end between source and destination applications: across the boundary between host and network, between interconnected networks, and through middleboxes;
- it SHOULD be common to IPv4 and IPv6 and transport agnostic;
- it SHOULD be incrementally deployable;
- it SHOULD enable an AQM to classify packets encapsulated by outer IP or lower-layer headers;
- it SHOULD consume minimal extra codepoints;
- it SHOULD not lead to some packets of a transport-layer flow being served by a different queue from 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.
2.2. L4S Packet Identification at Run-Time

The L4S treatment is an alternative packet marking treatment
[RFC4774] to the classic ECN treatment [RFC3168]. Like classic ECN,
it 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 behaviour.

For a packet to receive L4S treatment as it is forwarded, the sender
MUST set the ECN field in the IP header (v4 or v6) to the ECT(1)
codepoint.

A network node that implements the L4S service MUST classify arriving
ECT(1) packets for L4S treatment and it SHOULD classify arriving CE
packets for L4S treatment as well. Section 2.5 describes a possible
exception to this latter rule.

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 2.6. 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 [RFC7567] (Section 4.2.1), which follows
similar advice in RFC 3168 (Section 7).

For backward compatibility in uncontrolled environments, a network
node that implements the L4S treatment MUST also implement a classic
AQM treatment. It MUST classify arriving ECT(0) and Not-ECT packets
for treatment by the Classic AQM (see the discussion of the
classifier for the dual-queue coupled AQM in
[I-D.ietf-tsvwg-aqm-dualq-coupled]). Classic treatment means that
the AQM will mark ECT(0) packets under the same conditions as it
would drop Not-ECT packets [RFC3168].

2.3. Interaction of the L4S Identifier with other Identifiers

In a typical case for the public Internet with just the default Best
Efforts Per-Hop Behaviour (PHB), a network element that implements
L4S MAY classify packets into the L4S treatment if they carry certain
operator-defined non-ECN identifiers. Such non-ECN-based packet
types MUST be safe to mix with L4S traffic without harming the low
latency service. 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 DSCPs (see [I-D.briscoe-tsvwg-l4s-diffserv]).

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) and CE ECN codepoints indicate to a network element that a host is sending L4S packets - specifically that the host claims its behaviour satisfies the pre-requisite transport requirements in Section 2.4.

[I-D.briscoe-tsvwg-l4s-diffserv] gives detailed discussion on the interactions between L4S and Diffserv. In summary, Diffserv provides for differentiation of both bandwidth and low latency, but its control of latency depends on its control of bandwidth. In contrast, L4S concerns low latency, which it can provide for all traffic without differentiation and without affecting bandwidth allocation.

The way a network element would classify on the DSCP and ECN fields depends on which of the following cases applies:

- A common case is where a network element only offers Default Forwarding (DF a.k.a. Best Efforts). In this case, 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 matched any of the DSCPs in the bullet above, the network element could classify it for the L4S treatment irrespective of its ECN codepoint.

- On the other hand, a network operator might offer more Diffserv PHBs than just Default Forwarding. [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.
2.4. Pre-Requisite Transport Layer Behaviour

2.4.1. Pre-Requisite Congestion Response

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]) is an example of a scalable congestion control.

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) in the other.

In order to coexist safely with other Internet traffic, a scalable congestion control MUST NOT identify 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).
- 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 units of time, which is scalable, and MUST NOT count in units of packets (as in the 3 DupACK rule of traditional TCP), which is not scalable. 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 (see Appendix A.1.7 for rationale).

2.4.2. Pre-Requisite Transport Feedback

In general, a scalable congestion control needs feedback of the extent of CE marking on the forward path. Due to the history of TCP development, when ECN was added it 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: An ECN feedback protocol such as that specified in [I-D.stewart-tsvwg-sctpecn] would be a prerequisite for scalable congestion control. That draft would update the ECN feedback protocol sketched out in Appendix A of the standards track specification of SCTP [RFC4960] by adding a field to report the number of CE marks.

RTP over UDP: A prerequisite for scalable congestion control is for both (all) ends of one media-level hop to signal ECN support using the ecn-capable-rtp attribute [RFC6679]. Therefore, 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.

DCCP: The ACK vector in DCCP [RFC4340] is already sufficient to report the extent of CE marking as needed by a scalable congestion control.
2.5. 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 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 (ToDo: this advice will need to be verified experimentally). This is because a sender might have to switch from sending ECT(1) (L4S) packets to sending ECT(0) (Classic) packets, or back again, in the middle of a transport-layer flow. Such a switch-over is likely to be very rare, but it could be necessary if the path bottleneck moves from a network node that supports L4S to one that only supports Classic ECN. A host ought to be able to detect such a change from a change in RTT variation.

2.6. The Meaning of L4S CE Relative to Drop

The likelihood that an AQM drops a Not-ECT Classic packet \( (p_C) \) MUST be roughly proportional to the square of the likelihood that it would have marked it if it had been an L4S packet \( (p_L) \). That is

\[
p_C \approx (p_L / k)^2
\]

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, an AQM implementing the L4S and Classic treatments does not mark an ECT(1) packet under the same conditions that it would have dropped a Not-ECT packet. However, it does mark an ECT(0) packet under the same conditions that it would have dropped a Not-ECT packet.

3. L4S Experiments


4. IANA Considerations

This specification contains no IANA considerations.

5. Security Considerations

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

6. 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. Appendix A listing the TCP Prague 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.

7. References

7.1. Normative References

7.2. Informative References

[Alizadeh-stability]

[ARED01]

[DCttH15]

(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-00 (work in progress), March 2018.

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

[I-D.ietf-tcpm-rack]

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

[I-D.ietf-tsvwg-ecn-encap-guidelines]

[I-D.ietf-tsvwg-l4s-arch]

[I-D.sridharan-tcpm-ctcp]

[I-D.stewart-tsvwg-sctpecn]

[Mathis09]
[Paced-Chirping]


[QV]


[RFC2309]


[RFC2474]


[RFC2983]


[RFC3246]


[RFC3540]


[RFC3649]


[RFC4340]


Appendix A. The ‘TCP Prague Requirements’

This appendix is informative, not normative. It gives a list of modifications to current scalable transport protocols so that they can be deployed over the public Internet and coexist safely with existing traffic. The list complements the normative requirements in Section 2.4 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 TCP Prague 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.

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 transport protocols.

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: A scalable transport protocol 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/Cubic 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 transport 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/Cubic 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 the congestion window

Description: A scalable congestion control needs to remain responsive to congestion when RTTs are significantly smaller than in the current public Internet.

Motivation: As currently specified, the minimum required congestion window of TCP (and its derivatives) is set to 2 maximum segment sizes (MSS) (see equation (4) in [RFC5681]). Once the congestion window reaches this minimum, all current TCP algorithms become unresponsive to congestion signals. No matter how much drop or ECN marking, the congestion window no longer reduces. Instead, TCP forces the queue to grow, overriding any AQM and increasing queuing delay.

L4S mechanisms significantly reduce queueing delay so, over the same path, the RTT becomes lower. Then this problem becomes surprisingly common [TCP-sub-mss-w]. This is because, for the same link capacity, smaller RTT implies a smaller window. For instance, consider a residential setting with an upstream broadband Internet access of 8 Mb/s, assuming a max segment size of 1500 B. Two upstream flows will each have the minimum window of 2 MSS if the RTT is 6ms or less, which is quite common when accessing a nearby data centre. So, any more than two such parallel TCP flows will become unresponsive and increase queuing delay.

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

A.1.7. Measuring Reordering Tolerance in Time Units

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

Motivation: The 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 units of time before they deem there has been a loss, the
time over which the network has to keep packets in order stays constant.

Tolerance of reordering over a small duration will 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.

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 2.4.1) says that senders are not entitled to identify packets as L4S in the IP/ECN field unless they use the time-based approach. Then networks that identify L4S traffic separately (e.g. using [I-D.ietf-tsvwg-aqm-dualq-coupled]) can know for certain that all L4S traffic is using the scalable time-based approach.

This will allow networks to remove head-of-line blocking delay immediately, but only for L4S traffic. But 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 able to allow 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.

A.2. Scalable Transport Protocol Optimizations

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

Description: 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.

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.
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.

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 TCP Prague. 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 tranport-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 arriving early, which is a form of reordering. Reordering can cause the TCP sender to retransmit spuriously. However, one or two packets delivered early does not cause any spurious retransmissions because the subsequent packets continue to move the cumulative acknowledgement boundary forwards. Anyway, the risk of reordering would be low, because: i) it is quite unusual to experience more than one bottleneck queue on a path; ii) even then, reordering would only occur if there was simultaneous mixing of classic and L4S traffic, which would be much less likely in an access link, which is where most bottlenecks are located; iii) even then, spurious retransmissions would only occur if a contiguous sequence of three or more classic CE packets from one bottleneck arrived at the next, which should in itself happen very rarely with a good AQM. The risk would be completely eliminated in AQMs that were transport-aware (but they should not need to be);

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.

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
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>
</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
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.

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. So it is highly unlikely that ECT(1) will be needed for integrity protection in future.

C.2. Notification of Less Severe Congestion than CE

Various researchers have proposed to use ECT(1) as a less severe congestion notification than CE, particularly to enable flows to fill available capacity more quickly after an idle period, when another flow departs or when a flow starts, e.g. VCP [VCP], Queue View (QV) [QV].

Before assigning ECT(1) as an identifier for L4S, we must carefully consider whether it might be better to hold ECT(1) in reserve for future standardisation of rapid flow acceleration, which is an important and enduring problem [RFC6077].
Pre-Congestion Notification (PCN) is another scheme that assigns alternative semantics to the ECN field. It uses ECT(1) to signify a less severe level of pre-congestion notification than CE [RFC6660]. However, the ECN field only takes on the PCN semantics if packets carry a Diffserv codepoint defined to indicate PCN marking within a controlled environment. PCN is required to be applied solely to the outer header of a tunnel across the controlled region in order not to interfere with any end-to-end use of the ECN field. Therefore a PCN region on the path would not interfere with any of the L4S service identifiers proposed in Appendix B.

Authors’ Addresses

Koen De Schepper  
Nokia Bell Labs  
Antwerp  
Belgium

Email: koen.de_schepper@nokia.com  
URI: https://www.bell-labs.com/usr/koen.de_schepper

Bob Briscoe (editor)  
CableLabs  
UK

Email: ietf@bobbriscoe.net  
URI: http://bobbriscoe.net/
Forward Error Correction (FEC) Framework Extension to Sliding Window Codes
draft-ietf-tsvwg-fecframe-ext-06

Abstract

RFC 6363 describes a framework for using Forward Error Correction (FEC) codes to provide protection against packet loss. The framework supports applying FEC to arbitrary packet flows over unreliable transport and is primarily intended for real-time, or streaming, media. However FECFRAME as per RFC 6363 is restricted to block FEC codes. The present document updates FECFRAME 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 March 23, 2019.

Copyright Notice

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

Many applications need to transport a continuous stream of packetized data from a source (sender) to one or more destinations (receivers) over networks that do not provide guaranteed packet delivery. In particular packets may be lost, which is strictly the focus of this document: we assume that transmitted packets are either lost (e.g., because of a congested router, of a poor signal-to-noise ratio in a wireless network, or because the number of bit errors exceeds the correction capabilities of the physical-layer error correcting code)
or received by the transport protocol without any corruption (i.e.,
the bit-errors, if any, have been fixed by the physical-layer error
correcting code and therefore are hidden to the upper layers).

For these use-cases, Forward Error Correction (FEC) applied within
the transport or application layer, is an efficient technique to
improve packet transmission robustness in presence of packet losses
(or "erasures"), without going through packet retransmissions that
create a delay often incompatible with real-time constraints. The
FEC Building Block defined in [RFC5052] provides a framework for the
definition of Content Delivery Protocols (CDPs) that make use of
separately defined FEC schemes. Any CDP defined according to the
requirements of the FEC Building Block can then easily be used with
any FEC Scheme that is also defined according to the requirements of
the FEC Building Block.

Then FECFRAME [RFC6363] provides a framework to define Content
Delivery Protocols (CDPs) that provide FEC protection for arbitrary
packet flows over an unreliable datagram service transports 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. This approach
has major impacts on FEC encoding and decoding delays. The data
packets of continuous media flow(s) may be passed to the transport
layer immediately, without delay. But the block creation time, that
depends on the number of source symbols in this block, impacts both
the FEC encoding delay (since encoding requires that all source
symbols be known), and mechanically the packet loss recovery delay at
a receiver (since no repair symbol for the current block can be
generated and therefore received before that time). Therefore a good
value for the block size is necessarily a balance between the maximum
FEC decoding latency at the receivers (which must be in line with the
most stringent real-time requirement of the protected flow(s), hence
an incentive to reduce the block size), and the desired robustness
against long loss bursts (which increases with the block size, hence
an incentive to increase this size).

This document updates [RFC6363] in order to also support FEC codes
based on a sliding encoding window (A.K.A. convolutional codes)
[RFC8406]. This encoding window, either of fixed or variable size,
slides over the set of source symbols. FEC encoding is launched
whenever needed, from the set of source symbols present in the
sliding encoding window at that time. This approach significantly reduces FEC-related latency, since repair symbols can be generated and passed to the transport layer on-the-fly, at any time, and can be regularly received by receivers to quickly recover packet losses. Using sliding window FEC codes is therefore highly beneficial to real-time flows, one of the primary targets of FECFRAME. [RLC-ID] provides an example of such FEC Scheme for FECFRAME, built upon the simple sliding window Random Linear Codes (RLC).

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

- this extension 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;

- any receiver, for instance a legacy receiver that only supports block FEC schemes, can easily identify the FEC Scheme used in a FECFRAME session. This is made possible with the FEC Encoding ID that identifies the FEC Scheme used and which 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.

**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 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", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

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 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 dependant 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 RCTP 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 and the rank of the associated linear system permits it, then FEC decoding can be performed in order to recover missing source packets.
5. The FEC Scheme returns the received and decoded ADUs to the FEC Framework, along with indications of any ADUs that were missing and could not be decoded.

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

5.1.  General

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

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

5.3. FEC Scheme Requirements

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

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

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

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

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

- MUST define the relationships between ADUs and the associated source symbols (mapping);
- MUST define the management of the encoding window that slides over the set of ADUs. Appendix A provides non normative hints about what FEC Scheme designers need to consider;
- MUST define the management of the decoding window, consisting of 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.

- Litncensing: 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.
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 for their valuable feedbacks on this document. This document being an extension to [RFC6363], the authors would also like to thank Mark Watson as the main author this RFC.

14. References

14.1. Normative References


14.2. Informative References


Appendix A. About Sliding Encoding Window Management (non Normative)

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 non normative 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
IANA Assignment of DSCP Pool 3 (xxxx01) Values to require Publication of a Standards Track or Best Current Practice RFC
draft-ietf-tsvwg-iana-dscp-registry-08

Abstract

The Differentiated Services (Diffserv) architecture specifies use of a field in the IPv4 and IPv6 packet headers to carry Diffserv Codepoint (DSCP) values. The Internet Assigned Numbers Authority (IANA) maintains a registry of assigned DSCP values.

This update to RFC2474 changes the IANA assignment policy for Pool 3 of the registry (i.e., DSCP values of the form xxxx01) to Standards Action, i.e., values are assigned through a Standards Track or Best Current Practice RFC. The update also removes permission for experimental and Local Use of the Codepoints that form Pool 3 of the DSCP registry; Pool 2 Codepoints (i.e., DSCP values of the form xxxx11) remain available for these purposes.

Status of this Memo

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

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

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

This Internet-Draft will expire on December 07, 2018.
1. Introduction

The Differentiated Services (Diffserv) [RFC2475] architecture (updated by [RFC3260]) provides scalable service differentiation in the Internet. Diffserv uses the six most significant bits of the former IPv4 Type of Service (TOS) octet or the former IPv6 Traffic Class octet to convey the field, which is used to carry the Diffserv Codepoint (DSCP). This DSCP value is used to select a Diffserv Per hop Behaviour, PHB.

The six bit field is capable of conveying 64 distinct codepoints, and this codepoint space has been divided into three pools for the purpose of codepoint assignment and management (as shown in figure 1). Pool 1 comprises 32 codepoints [RFC2474]. These are assigned by Standards Action, as defined in [RFC8126]. Pool 2 comprises a pool of 16 codepoints reserved for experimental or Local Use (EXP/LU) as defined in [RFC2474], and Pool 3 comprises 16 codepoints, which were specified as "initially available for experimental or local use, but which should be preferentially utilized for standardized assignments if Pool 1 is ever exhausted" [RFC2474].
At the time of writing this document, 22 of the 32 Pool 1 codepoints have currently been assigned.

Although Pool 1 has not yet been completely exhausted, there is a need to assign codepoints for particular PHBs that are unable to use any of the unassigned values in Pool 1. This document changes the IANA registration policy of Pool 3 to assignment by Standards Action (Section 4.9 of [RFC8126] defines this as "assigned only through Standards Track or Best Current Practice RFCs in the IETF Stream").

An example is the need to assign a suitable recommended default codepoint for the Lower Effort (LE) per-hop behavior (PHB) [I-D.ietf-tsvwg-le-phb]. The LE PHB is designed 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 is allowed to preempt it). In deployed networks, there is continued use of bleaching (i.e. intentionally setting to zero) of the IP precedence field. (Setting the IP Precedence field to zero disables any class-based flow management by routers configured with TOS-based packet processing). This causes the first three bits of the former TOS byte (now the upper part of the DSCP field) to become zero. There is therefore a need to avoid this remapping of the DSCP for the LE PHB by assigning a codepoint that already has a zero value in the first three bits [I-D.ietf-tsvwg-le-phb].

Furthermore, if the LE PHB were to have been assigned one of the currently unused Pool 1 codepoints with a zero value in the first three bits, any bleaching of the IP precedence field would result in other (higher assurance) traffic being also remapped to the assigned DSCP. This remapping could then cause diffserv-marked traffic to receive an unintentional LE treatment for the remainder of the Internet path. It is therefore important to avoid the resulting priority inversion. The absence of unassigned codepoints in Pool 1 that exhibit these important properties motivates assigning a Pool 3 codepoint as the default that is recommended for use with this PHB.
To allow the IETF to utilise Pool 3 codepoints, this document requests IANA to manage Pool 3 assignments for DSCP values in Pool 3 via the Standards Action policy [RFC8126].

2. Terminology

This document assumes familiarity with the terminology used in [RFC2475] updated by [RFC3260].

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].

3. The update to RFC2474

This document updates section 6 of [RFC2474], in the following ways.

It updates the following text concerning the assignment policy:

OLD: which are initially available for experimental or local use, but which should be preferentially utilized for standardized assignments if Pool 1 is ever exhausted.

NEW: which are utilized for standardized assignments (replacing the previous availability for experimental or local use).

It removes the footnote in RFC2474 relating to Pool 3:

DELETE: "(*) may be utilized for future Standards Action allocations as necessary"

The new registry assignment policy is shown in Figure 2.

<table>
<thead>
<tr>
<th>Pool</th>
<th>Codepoint space</th>
<th>Assignment Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>xxxxx0</td>
<td>Standards Action</td>
</tr>
<tr>
<td>2</td>
<td>xxxx11</td>
<td>EXP/LU</td>
</tr>
<tr>
<td>3</td>
<td>xxxx01</td>
<td>Standards Action</td>
</tr>
</tbody>
</table>

Note for Pool 2: "Reserved for experimental or Local Use"

Figure 2: Updated Assignment Policy for the DSCP Registry

4. Security Considerations

Security considerations for the use of DSCP values are described in the RFCs that define their usage. This document does not present new
5. IANA Considerations

This section requests IANA to change the use of Pool 3 in the DSCP registry and to manage this pool using Standards Action, as defined as Section 4.9 of [RFC8126].

This requests IANA to make the following changes to the Differentiated Services field Codepoints (DSCP) Registry, made available at [Registry].

IANA is requested to reference RFC2474 and Section 4 of RFC3260 for the overall format of the DSCP registry.

IANA is requested to reference RFC2474 and Section 4 of RFC3260 for Pool 1.

This update does not modify the IANA registry text for Pool 2. This pool continues to preserve the note shown in Figure 2.

The previous registry text:

3 xxxx01 Experimental or Local Use May be utilized for future Standards Action allocations as necessary.

is replaced with the following registry text:

3 xxxx01 Standards Action.

To manage codepoints in Pool 3, IANA is requested to create and maintain a "Pool 3 Codepoints" subregistry. Pool 3 of the registry is to be created initially empty, with a format identical to that used for "Pool 1 Codepoints".

IANA is requested to reference RFC2474, Section 4 of RFC3260, and the current document for Pool 3.

The Registration Procedure for use of Pool 3 is Standards Action, as defined as Section 4.9 of [RFC8126]. IANA is expected to normally make assignments from Pool 1, until this Pool is exhausted, but MAY make assignments from Pool 3 where the format of the codepoint has properties that are needed for a specific PHB. The required characteristics for choosing a requested DSCP value MUST be explained in the IANA considerations of the document that requests any assignment from Pool 3.

6. Acknowledgments

G. Fairhurst received funding from the European Union’s Horizon 2020 research and innovation program 2014-2018 under grant agreement No. 644334 (NEAT).

7. References
7.1. Normative References


[Registry] IANA, "Differentiated Services Field Codepoints (DSCP), https://www.iana.org/assignments/dscp-registry/dscp-registry.xhtml".

7.2. Informative References


Appendix A. Revision Notes

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

Individual submission as draft -00.

o This is the initial version of the document.

o Advice in this rev. from Michelle Cotton on the IANA procedure.

o Thanks to Brian Carpenter for helpful inputs to this ID.

Individual submission as draft -01.
Thanks to Roland Bless for review comments.

Individual submission as draft -02 (author requests adoption as a TSVWG WG draft).

Thanks to David Black for review comments in preparing rev -02.

Working Group submission as draft -00

Adopted by the TSVWG working group.

Working Group submission as draft -01

Fixed exploded acronyms.

Working Group submission as draft -02

Corrections after WGLC.

Working Group submission as draft -03

Corrections after TSVWG Shepherd Review.

Working Group submission as draft -04

Added RFC 3260 as a necessary downref, with IANA asked to reference this.

Working Group submission as draft -05

Corrections following AD review.

Expansion of explanation about why the proposed change will help in assignment of a suitable DSCP for the LE PHB.

Working Group submission as draft -06

GenART feedback to changed assignment method to assignment policy.

Correction to the IANA reference documents.

Working Group submission as draft -07

Revised after IESG feedback - Assignment Policy changed final para text; Figure 2 reference changed; bleaching defined; definition of standards action aligned with actual IANA policy.

Working Group submission as draft -08

Revised after AD feedback - definition of standards action.

Author’s Address
Abstract

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

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

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

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

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

This Internet-Draft will expire on September 23, 2018.

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 queueing Latency, ultra-Low Loss and Scalable throughput (L4S) - for _all_ traffic. A service for all traffic will need none of the configuration or management baggage (traffic policing, traffic contracts) associated with favouring some packets over others. This document describes the L4S architecture for achieving that goal.

It must be said that queuing delay only degrades performance infrequently [Hohlfeld14]. It only occurs when a large enough
capacity-seeking (e.g. TCP) flow is running alongside the user’s traffic in the bottleneck link, which is typically in the access network. Or when the low latency application is itself a large capacity-seeking flow (e.g. interactive video). At these times, the performance improvement from L4S must be so remarkable that network operators will be motivated to deploy it.

Active Queue Management (AQM) is part of the solution to queuing under load. AQM improves performance for all traffic, but there is a limit to how much queuing delay can be reduced by solely changing the network; without addressing the root of the problem.

The root of the problem is the presence of standard TCP congestion control (Reno [RFC5681]) or compatible variants (e.g. TCP Cubic [RFC8312]). We shall call this family of congestion controls ‘Classic’ TCP. It has been demonstrated that if the sending host replaces Classic TCP with a ‘Scalable’ alternative, when a suitable AQM is deployed in the network the performance under load of all the above interactive applications can be stunningly improved. For instance, queuing delay under heavy load with the example DCTCP/DualQ solution cited below is roughly 1 millisecond (1 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 node should give nearly all the benefit. The L4S approach requires component mechanisms in different parts of an Internet path 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 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 compatible with standard TCP Reno [RFC5681]. 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 18 ms, 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 two separate meanings 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 a second 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.johansson-quic-ecn] 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-ecn-l4s-id], while the Classic queue uses either classic ECN [RFC3168] or drop, which are equivalent.

Before Classic ECN was standardized, there were various proposals to give an ECN mark a different meaning from drop. However, there was no particular reason to agree on any one of the alternative meanings, so 'the same as drop' was the only compromise that could be reached. RFC 3168 contains a statement that:

"An environment where all end nodes were ECN-Capable could allow new criteria to be developed for setting the CE codepoint, and new congestion control mechanisms for end-node reaction to CE packets. However, this is a research issue, and as such is not addressed in this document."

Latency isolation with coupled congestion notification (network): Using just two queues is not essential to L4S (more would be possible), but it is the simplest way to isolate all the L4S traffic that keeps latency low from all the legacy Classic traffic that does not.

Similarly, coupling the congestion notification between the queues is not necessarily essential, but it is a clever and simple way to allow senders to determine their rate, packet-by-packet, rather than be overridden by a network scheduler. Because otherwise a network scheduler would have to inspect at least transport layer headers, and it would have to continually assign a rate to each flow without any easy way to understand application intent.

L4S packet identifier (protocol): Once there are at least two separate treatments in the network, hosts need an identifier at the IP layer to distinguish which treatment they intend to use.

Scalable congestion notification (host): A scalable congestion control keeps the signalling frequency high so that rate variations can be small when signalling is stable, and rate can track variations in available capacity as rapidly as possible otherwise.

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 a 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 true 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 datacentre 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 datacentres.

6.3.2. Deployment Sequences

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

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

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

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

<table>
<thead>
<tr>
<th>Servers or proxies</th>
<th>Access link</th>
<th>Clients</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 DCTCP (existing)</td>
<td>DualQ AQM downstream</td>
<td>DCTCP (existing)</td>
</tr>
<tr>
<td>WORKS DOWNSTREAM FOR CONTROLLED DEPLOYMENTS/TRIALS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 TCP Prague</td>
<td>AccECN (already in progress:DCTCP/BBR)</td>
<td>FULLY WORKS DOWNSTREAM</td>
</tr>
<tr>
<td>3</td>
<td>DualQ AQM upstream</td>
<td>TCP Prague</td>
</tr>
<tr>
<td>FULLY WORKS UPSTREAM AND DOWNSTREAM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: Example L4S Deployment Sequences

Figure 3 illustrates some example sequences in which the parts of L4S might be deployed. It consists of the following stages:
1. Here, the immediate benefit of a single AQM deployment can be seen, but limited to a controlled trial or controlled deployment. In this example downstream deployment is first, but in other scenarios the upstream might be deployed first. If no AQM at all was previously deployed for the downstream access, the DualQ AQM greatly improves the Classic service (as well as adding the L4S service). If an AQM was already deployed, the Classic service will be unchanged (and L4S will still be added).

2. In this stage, the name ‘TCP Prague’ is used to represent a variant of DCTCP that is safe to use in a production environment. If the application is primarily unidirectional, ‘TCP Prague’ at one end will provide all the benefit needed. Accurate ECN feedback (AccECN) [I-D.ietf-tcpm-accurate-ecn] is needed at the other end, but it is a generic ECN feedback facility that is already planned to be deployed for other purposes, e.g. DCTCP, BBR [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 performance improvements between DCTCP and TCP Prague (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 (no reference yet);
- 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. 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 [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-00 (work in progress), March 2018.

[I-D.ietf-tcpm-alternativebackoff-ecn]
Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst,
"TCP Alternative Backoff with ECN (ABE)", draft-ietf-tcpm-
alternativebackoff-ecn-07 (work in progress), March 2018.

[I-D.ietf-tcpm-generalized-ecn]
Bagnulo, M. and B. Briscoe, "ECN++: Adding Explicit
Congestion Notification (ECN) to TCP Control Packets",
draft-ietf-tcpm-generalized-ecn-02 (work in progress),
October 2017.

[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-04
(work in progress), March 2018.

[I-D.ietf-tsvwg-ecn-encap-guidelines]
Briscoe, B., Kaippallimalil, J., and P. Thaler,
"Guidelines for Adding Congestion Notification to
Protocols that Encapsulate IP", draft-ietf-tsvwg-ecn-
encap-guidelines-10 (work in progress), March 2018.

[I-D.ietf-tsvwg-ecn-l4s-id]
Schepper, K. and B. Briscoe, "Identifying Modified
Congestion Notification (ECN) Semantics for
Ultra-Low Queuing Delay (L4S)", draft-ietf-tsvwg-ecn-
l4s-id-02 (work in progress), March 2018.

[I-D.johansson-quic-ecn]
Johansson, I., "ECN support in QUIC", draft-johansson-
quic-ecn-03 (work in progress), May 2017.

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

[I-D.stewart-tsvwg-sctpecn]
Stewart, R., Tuexen, M., and X. Dong, "ECN for Stream
Control Transmission Protocol (SCTP)", draft-stewart-
tsvwg-sctpecn-05 (work in progress), January 2014.


Appendix A. Standardization items

The following table includes all the items that will need to be standardized to provide a full L4S architecture.

Briscoe, et al. Expires September 23, 2018
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/iccrg" combination refers to the procedure typically used for congestion control changes, where tcpm owns the approval decision, but uses the iccrg 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, [RFC0257]</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>Y/a</td>
</tr>
<tr>
<td>2</td>
<td>tsvwg</td>
<td>n/a</td>
</tr>
<tr>
<td>3</td>
<td>tcpm</td>
<td>Y</td>
</tr>
<tr>
<td>4-1</td>
<td>tcpm</td>
<td>Y/a</td>
</tr>
<tr>
<td>4-2</td>
<td>tcpm/</td>
<td></td>
</tr>
<tr>
<td></td>
<td>iccrg?</td>
<td></td>
</tr>
<tr>
<td>4-3</td>
<td>tcpm/</td>
<td></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></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-1</td>
<td>tcpm</td>
<td>Y</td>
</tr>
<tr>
<td>5-2</td>
<td>tcpm/</td>
<td>Y/a</td>
</tr>
<tr>
<td></td>
<td>iccrg?</td>
<td></td>
</tr>
<tr>
<td>5-3</td>
<td>tcpm/</td>
<td></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/
Koen De Schepper
Nokia Bell Labs
Antwerp
Belgium

Email: koen.de_schepper@nokia.com
URI: https://www.bell-labs.com/usr/koen.de_schepper

Marcelo Bagnulo
Universidad Carlos III de Madrid
Av. Universidad 30
Leganes, Madrid 28911
Spain

Phone: 34 91 6249500
Email: marcelo@it.uc3m.es
URI: http://www.it.uc3m.es
A Lower Effort Per-Hop Behavior (LE PHB)
draft-ietf-tsvwg-le-phb-05

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 January 4, 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.

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 ....................................... 5
5. Traffic Conditioning Actions ............................ 6
6. Recommended DS Codepoint ............................... 6
7. Deployment Considerations ............................... 7
8. Remarking to other DSCPs/PHBs .......................... 8
9. The Update to RFC 4594 .................................. 8
10. The Update to RFC 8325 .................................. 10
11. The Update to draft-ietf-tsvwg-rtcweb-qos .................. 10
12. IANA Considerations ................................... 12
13. Security Considerations ................................ 12
14. References ............................................ 12
   14.1. Normative References ............................... 12
   14.2. Informative References ............................. 13
Appendix A. History of the LE PHB .......................... 15
Appendix B. Acknowledgments .............................. 15
Appendix C. Change History ................................. 15
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 traffic for which delivery is considered optional; that is, packets of this type of traffic ought to consume network resources only when no other traffic is present. 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 (Appendix A). Other commonly used names for LE PHB type services are "Lower-than-best-effort" or "Less-than-best-effort". Alternatively, the effect of this type of traffic on all other network traffic is strictly limited ("no harm" property). This is distinct from "best-effort" (BE) traffic since the network makes no commitment to deliver LE packets. In contrast, BE traffic receives an implied "good faith" commitment of at least some available network resources. This document proposes a Lower Effort Differentiated Services per-hop behavior (LE PHB) for handling this "optional" traffic in a differentiated services node.

2. Requirements Language

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

3. Applicability

A Lower Effort PHB is applicable for many applications that otherwise use best-effort delivery. More specifically, it is suitable for traffic and services that can tolerate strongly varying throughput for their data flows, especially periods of very low throughput or even starvation (i.e., long interruptions due to significant or even complete packet loss). Therefore, an application sending an LE
marked flow needs to be able to tolerate short or (even very) long interruptions due to the presence of severe congestion conditions during the transmission of the flow. Thus, there ought to be an expectation that packets of the LE PHB could be excessively delayed or dropped when any other traffic is present. 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 [RFC8085]). Since LE traffic could be starved completely for a longer period of time, transport protocols or applications (and their related congestion control mechanisms) SHOULD be able to detect and react to such a situation and ought to resume the transfer as soon as possible. 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.

Use of the LE PHB might assist a network operator in moving certain kinds of traffic or users to off-peak times. Alternatively, or in addition, 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 nor should packets 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 main 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.

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 SHOULD be forwarded only if no packet with any other PHB is awaiting transmission.
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 [RFC3168] SHOULD be used for LE packets, too.

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 on a regular basis without consent or knowledge of the user. 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
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 support the LE PHB should be aware that they violate the "no harm" property of LE. DS domains that do not offer support for the LE PHB support 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. 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 case 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 case 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-intercon [RFC8100].

9. The Update to RFC 4594

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

Changes:

- This update to RFC 4594 removes the following entry from figure 3:

  |---------------+---------+-------------+--------------------------|
  | Low-Priority  | CS1     | 001000      | Any flow that has no BW  |
  |     Data      |         |             | assurance                |
  |---------------+---------+-------------+--------------------------|

  and replaces this by the following entry:

  |---------------+---------+-------------+--------------------------|
  | Low-Priority  | LE      | 000001      | Any flow that has no BW  |
  |     Data      |         |             | assurance                |
  |---------------+---------+-------------+--------------------------|

- This update to RFC 4594 removes the following entry from figure 4:

  |---------------+-------+-------------------+---------+--------+----|
  | Low-Priority  | CS1   | Not applicable    | RFC3662 | Rate   | Yes|
  |     Data      |       |                   |         |        |    |
  |---------------+-------+-------------------+---------+--------+----|

  and replaces this by the following entry:

  |---------------+-------+-------------------+---------+--------+----|
  | Low-Priority  | LE    | Not applicable    | RFCXXXX | Rate   | Yes|
  |     Data      |       |                   |         |        |    |
  |---------------+-------+-------------------+---------+--------+----|

- 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.
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)."

10. The Update to RFC 8325

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 CS1 DSCP to UP 1"

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

+---------------+------+----------+-------------+--------------------+
| Low-Priority  | CS1  | RFC 3662 |     1       | AC_BK (Background) |
+---------------+------+----------+-------------+--------------------+

and replaces this by the following entry:

+---------------+----------+----------+-------------+--------------------+
| Low-Priority  | LE       | RFCXXXX  |     1       | AC_BK (Background) |
+---------------+----------+----------+-------------+--------------------+

11. 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</td>
<td>AF41, AF42</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(36, 38)</td>
<td>(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</td>
<td>AF31, AF32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(28, 30)</td>
<td>(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>

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.

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].
12. 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 [I-D.ietf-tsvwg-iana-dscp-registry] for the request to re-classify 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]

13. 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. However, this disclosed information is only useful if some form of identification happened at the same time, see [RFC6973] for further details on general privacy threats.

14. References

14.1. Normative References

14.2. Informative References


Bless Expires January 4, 2019 [Page 14]
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 borrowed from earlier Internet-Drafts and RFCs the co-authors of previous specifications are acknowledged here: Kathie Nichols and Klaus Wehrle. David Black, Gorry Fairhurst, and Ruediger Geib provided helpful comments and suggestions.

Appendix C.  Change History

This section briefly lists changes between Internet-Draft versions for convenience.

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 9 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.

Please replace the occurrences of RFCXXXX in Section 9 and Section 10 with the assigned RFC number for this document.

Delete Appendix C.

Delete this section.
Author’s Address

Roland Bless
Karlsruhe Institute of Technology (KIT)
Kaiserstr. 12
Karlsruhe  76131
Germany

Phone: +49 721 608 46413
Email: roland.bless@kit.edu
Abstract

The Stream Control Transmission Protocol (SCTP) provides a reliable communications channel between two end-hosts in many ways similar to the Transmission Control Protocol (TCP). With the widespread deployment of Network Address Translators (NAT), specialized code has been added to NAT for TCP that allows multiple hosts to reside behind a NAT and yet use only a single globally unique IPv4 address, even when two hosts (behind a NAT) choose the same port numbers for their connection. This additional code is sometimes classified as Network Address and Port Translation (NAPT).

This document describes the protocol extensions required for the SCTP endpoints and the mechanisms for NATs necessary to provide similar features of NAPT in the single-point and multi-point traversal scenario.
Copyright Notice

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

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

Table of Contents

1. Introduction ................................................. 3
2. Conventions ................................................. 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
   7.1. Single-homed Client to Single-homed Server ............ 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>No Support</td>
<td>No Support</td>
<td>Limited</td>
</tr>
<tr>
<td>Support</td>
<td>No Support</td>
<td>Support</td>
<td>None</td>
</tr>
<tr>
<td>No Support</td>
<td>Support</td>
<td>No Support</td>
<td>Limited</td>
</tr>
<tr>
<td>No Support</td>
<td>No Support</td>
<td>Support</td>
<td>None</td>
</tr>
<tr>
<td>No Support</td>
<td>No Support</td>
<td>No Support</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 1: Communication possibilities

From the table 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:

```
+---------+                   +---------+
|  SCTP   |         +-----+        /        \
|end point|=========| NAT |========= | Internet | =======|end point|
|    A    |         +-----+        \        /          |    B    |
+---------+                   +---------+
```

Single NAT scenario

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.

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

INIT[Initiate-Tag]
Priv-Addr:Int-Port ------> Ext-Addr:Ext-Port
Ext-VTag=0

Create(Initiate-Tag, Int-Port, 0, Ext-Port, Priv-Addr, RestartSupported)
Returns(NAT-State control block)

Translate To:

```
+--------+          +-----+           /        \\           +--------+
| Initiator | <------> | NAT | <------> | Destination | <------> | Initiator |
+--------+          +-----+           \         /          +--------+
\--/---/        \--/---/                \--/---/     
```

INI[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

The processing of incoming SCTP packets containing INIT-ACK chunks is described in the following figure. The Lookup() function getting as input the Internal-VTag, Internal-Port, External-VTag, and External-Port, returns the corresponding entry of the NAT table and updates the External-VTag by substituting it with the value of the Initiate-Tag of the INIT-ACK chunk. The wildcard character signifies that the parameter’s value is not considered in the Lookup() function or changed in the Update() function, respectively.
In the case Lookup fails, the SCTP packet is dropped. The Update routine inserts the External-VTag (the Initiate-Tag of the INIT-ACK chunk) in the NAT state control block.

The processing of incoming SCTP packets containing an ABORT or SHUTDOWN-COMPLETE chunk with the T-Bit set is described in the following figure.
For an incoming packet containing an INIT-chunk a table lookup is made only based on the addresses and port numbers. If an entry with an External-VTag of zero is found, it is considered a match and the External-VTag is updated.

This allows the handling of INIT-collision through NAT.

5. Data Formats

This section defines the formats used to support NAT traversal. Section 5.1 and Section 5.2 describe chunks and error causes sent by 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

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 9    | Reserved  |M|T|           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\ zero or more Error Causes /
```

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

[NOTE:
  ASSIGNMENT OF M-BIT TO BE CONFIRMED BY IANA.
]

5.2. New Error Causes

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

5.2.1. VTag and Port Number Collision Error Cause

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Cause Code = 0x00B0        |     Cause Length = Variable   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\ Chunk /
```

Stewart, et al. Expires January 3, 2019
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

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.
Assign a cause-code to be confirmed by IANA.

5.2.3. Port Number Collision Error Cause

```
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Cause Code = 0x00B2        |     Cause Length = Variable   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     Chunk
/ 
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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.

Assign a cause-code to be confirmed by IANA.

5.3. New Parameters

This section defines new parameters and their valid appearance defined by this document.
5.3.1. Disable Restart Parameter

This parameter is used to indicate that the RESTART procedure is requested to be disabled. Both endpoints of an association MUST include this parameter in the INIT chunk and INIT-ACK chunk when establishing an association and MUST include it in the ASCONF chunk when adding an address to successfully disable the restart procedure.

Parameter Type: 2 bytes (unsigned integer)
This field holds the IANA defined parameter type for the Disable Restart Parameter. 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.

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.

[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.
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:

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

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

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

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

INI T[Initiate-Tag = 1234]
192.0.2.1:1 ------------------------> 203.0.113.1:2
Ext-VTtag = 0
Host B receives the INIT and sends an INIT-ACK with the NAT’s external address as destination address.

\[\text{INIT-ACK[Initiate-Tag = 5678]}\]
\[192.0.2.1:1 \text{<-----------------------} 203.0.113.1:2\]
\[\text{Int-VTag = 1234}\]

The handshake finishes with a COOKIE-ECHO acknowledged by a COOKIE-ACK.
7.2. Single-homed Client to Multi-homed Server

The internal client is single-homed whereas the external server is multi-homed. The client (Host A) sends an INIT like in the single-homed case.
INIT[Initiate-Tag = 1234]
10.0.0.1:1 ----> 203.0.113.1:2
Ext-VTag = 0

NAT creates entry:

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

INIT[Initiate-Tag = 1234]
192.0.2.1:1 --------------------------> 203.0.113.1:2
Ext-VTag = 0

The server (Host B) includes its two addresses in the INIT-ACK chunk, which results in two NAT entries.
NAT does need to change the table for second address:

<table>
<thead>
<tr>
<th>NAT</th>
<th>Int VTag</th>
<th>Int Port</th>
<th>Ext VTag</th>
<th>Ext Port</th>
<th>Priv Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1234</td>
<td>1</td>
<td>5678</td>
<td>2</td>
<td>10.0.0.1</td>
</tr>
</tbody>
</table>

INIT-ACK[Initiate-Tag = 5678]
10.0.0.1:1 <---- 203.0.113.1:2
Int-VTag = 1234

The handshake finishes with a COOKIE-ECHO acknowledged by a COOKIE-ACK.
7.3. 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>

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:

Init-ACK[Initiate-Tag = 5678]
10.0.0.1:1 <-------- 203.0.113.1:2
Int-VTag = 1234

The handshake finishes with a COOKIE-ECHO acknowledged by a COOKIE-ACK.
Host A announces its second address in an ASCONF chunk. The address parameter contains an undefined address (0) to indicate that the source address should be added. The lookup address parameter within the ASCONF chunk will also contain the pair of VTags (external and internal) so that the NAT may populate its table completely with this single packet.

ASCONF [ADD-IP=0.0.0.0, INT-VTag=1234, Ext-VTag = 5678]  
10.1.0.1:1 --------> 203.0.113.129:2  
Ext-VTag = 5678

NAT 2 creates complete entry:  

Association is already established between Host A and Host B, when the NAT loses its state and obtains a new public address. Host A sends a DATA chunk to Host B.

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

The NAT box cannot find entry for the association. It sends ERROR message with the M-Bit set and the cause "NAT state missing".
On reception of the ERROR message, Host A sends an ASCONF chunk indicating that the former information has to be deleted and the source address of the actual packet added.

```
+---------+--------+----------+--------+-----------+
| NAT     | Int     | Ext      | Ext    | Priv      |
| VTag    | Port    | VTag     | Port   | 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:

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

INIT [Initiate-Tag = 1234]
192.0.2.1:1 --> 203.0.113.1:2
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.

```
+--------+     +-------+     /         \
| 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

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

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></td>
<td>1234</td>
<td>1</td>
<td>5678</td>
<td>2</td>
<td>10.0.0.1</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:

INIT-ACK[Initiate-Tag = 1234]
192.0.2.1:1 --> 10.1.0.1:2
Ext-VTag = 5678

The lookup for COOKIE-ECHO and COOKIE-ACK is successful.
8. Socket API Considerations

This section describes how the socket API defined in [RFC6458] is extended to provide a way for the application to control NAT friendliness.

Please note that this section is informational only.

A socket API implementation based on [RFC6458] is extended by supporting one new read/write socket option.
8.1. Get or Set the NAT Friendliness (SCTP_NAT_FRIENDLY)

This socket option uses the option_level IPPROTO_SCTP and the option_name SCTP_NAT_FRIENDLY. It can be used to enable/disable the NAT friendliness for future associations and retrieve the value for future and specific ones.

```c
struct sctp_assoc_value {
    sctp_assoc_t assoc_id;
    uint32_t assoc_value;
};
```

**assoc_id**: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets the application may fill in an association identifier or SCTP_FUTURE_ASSOC for this query. It is an error to use SCTP_{CURRENT|ALL}_ASSOC in assoc_id.

**assoc_value**: A non-zero value indicates a NAT-friendly mode.

9. IANA Considerations

[NOTE to RFC-Editor:

"RFCXXXX" is to be replaced by the RFC number you assign this document.
]

[NOTE to RFC-Editor:

The 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

| Value | Cause Code                          | Reference |
|-------+------------------------------------+-----------|
| 176   | VTag and Port Number Collision      | [RFCXXXX] |
| 177   | Missing State                       | [RFCXXXX] |
| 178   | Port Number Collision               | [RFCXXXX] |

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:

| ID Value | Chunk Parameter Type     | Reference |
|----------|--------------------------+-----------|
| 49159    | Disable Restart (0xC007)  | [RFCXXXX] |
| 49160    | VTags (0xC008)            | [RFCXXXX] |

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


12.2. Informative References


Authors’ Addresses

Randall R. Stewart
Netflix, Inc.
Chapin, SC  29036
US
Email: randall@lakerest.net

Michael Tuexen
Muenster University of Applied Sciences
Stegerwaldstrasse 39
48565 Steinfurt
DE
Email: tuexen@fh-muenster.de

Irene Ruengeler
Muenster University of Applied Sciences
Stegerwaldstrasse 39
48565 Steinfurt
DE
Email: i.ruengeler@fh-muenster.de
Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on January 17, 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 .... 21
   3.15. Transmittal in Fast Recovery ....................... 24
   3.16. Initial Value of ssthresh .......................... 24
   3.17. Automatically Confirmed Addresses ................. 25
   3.18. Only One Packet after Retransmission Timeout ....... 26
   3.19. INIT ACK Path for INIT in COOKIE-WAIT State ....... 27
   3.20. Zero Window Probing and Unreachable Primary Path ... 28
   3.21. Normative Language in Section 10 .................... 29
   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 ...... 38
   3.25. Processing of Chunks in an Incoming SCTP Packet .... 40
   3.26. CWND Increase in Congestion Avoidance Phase ...... 41
   3.27. Refresh of cwnd and ssthresh after Idle Period .... 43
   3.28. Window Updates After Receiver Window Opens Up .... 44
   3.29. Path of DATA and Reply Chunks ...................... 45
   3.30. Outstanding Data, Flightsize and Data In Flight Key Terms 47
   3.31. CWND Degradation due to Max.Burst .................. 48
   3.32. Reduction of RTO.Initial ........................... 49
   3.33. Ordering of Bundled SACK and ERROR Chunks .......... 51
   3.34. Undefined Parameter Returned by RECEIVE Primitive .. 51
   3.35. DSCP Changes ..................................... 52
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.

3.1.3. Solution Description

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

3.2. Upper Layer Protocol Shutdown Request Handling

3.2.1. Description of the Problem

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

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

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

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

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 type codes is done through an IETF Consensus action, as defined in [RFC8126]. Documentation of the chunk type MUST contain the following information:

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</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>6</td>
<td>Cookie Preservative</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>9</td>
<td>Reserved for ECN Capable (Note 2)</td>
</tr>
<tr>
<td>32768 (0x8000) Host Name Address (Note 3)</td>
<td>Optional</td>
<td>11</td>
<td>Supported Address Types (Note 4)</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

New text: (Section 3.3.2)

<table>
<thead>
<tr>
<th>Variable Parameters</th>
<th>Status</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv4 Address (Note 1)</td>
<td>Optional</td>
<td>5</td>
<td></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
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).

3.6.3. Solution Description

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

3.7. Data Transmission Rules

3.7.1. Description of the Problem

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

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

3.7.2. Text Changes to the Document
The sender MUST also have an algorithm for sending new DATA chunks to avoid silly window syndrome (SWS) as described in [RFC0813]. The algorithm can be similar to the one described in Section 4.2.3.4 of [RFC1122]. However, regardless of the value of rwnd (including if it is 0), the data sender can always have one DATA chunk in flight to the receiver if allowed by cwnd (see rule B below). This rule allows the sender to probe for a change in rwnd that the sender missed due to the SACK having been lost in transit from the data receiver to the data sender.

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
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.

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.
Old text: (Section 6.7, Figure 9)

Endpoint A                                    Endpoint Z {App sends 3 messages; strm 0} DATA [TSN=6,Strm=0,Seq=2]  (Start T3-rtx timer) 

----> (ack delayed)

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]  (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)
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.

o) Receive Unacknowledged Message

Format: RECEIVE_UNACKED(data retrieval id, buffer address, buffer size [,stream id] [, stream sequence number] [,partial flag] [,payload protocol-id])
Old text: (Section 10.1)

M) Set Protocol Parameters

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

New text: (Section 10.1)

M) Set Protocol Parameters

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

Old text: (Appendix C)

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

New text: (Appendix C)

ICMP2) An implementation MAY ignore all ICMPv6 messages where the
  type field is not "Destination Unreachable", "Parameter
  Problem", or "Packet Too Big".
Internet-Draft         RFC 4960 Errata and Issues              July 2018

---------
Old text: (Appendix C)
---------

ICMP7) If the ICMP message is either a v6 "Packet Too Big" or a v4 "Fragmentation Needed", an implementation MAY process this information as defined for PATH MTU discovery.

---------
New text: (Appendix C)
---------

ICMP7) If the ICMP message is either a v6 "Packet Too Big" or a v4 "Fragmentation Needed", an implementation MAY process this information as defined for PMTU discovery.

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

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

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

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.
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).

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.

3.10.3. Solution Description

Text moved to the appropriate location.

3.11. partial_bytes_acked after T3-rtx Expiration

3.11.1. Description of the Problem

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

3.11.2. Text Changes to the Document

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

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

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

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

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

\[ \text{ssthresh} = \max(\text{cwnd}/2, 4\times\text{MTU}) \]
\[ \text{cwnd} = 1\times\text{MTU} \]
\[ \text{partial_bytes_acked} = 0 \]

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)
---------

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

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

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

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.

---------

Old text: (Section 8.3)
---------

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

---------

New text: (Section 8.3)
---------

Upon the receipt of the HEARTBEAT ACK, the sender of the HEARTBEAT MUST clear the error counter of the destination transport address to which the HEARTBEAT was sent, and mark the destination transport address as active if it is not so marked. The endpoint MAY optionally report to the upper layer when an inactive destination address is marked as active due to the reception of the latest HEARTBEAT ACK. The receiver of the HEARTBEAT ACK SHOULD also clear the association overall error counter (as defined in Section 8.1).
3.13.3. Solution Description

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

3.14. Path for Fast Retransmission

3.14.1. Description of the Problem

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

3.14.2. Text Changes to the Document

---------
Old text: (Section 6.4)
---------

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

---------
New text: (Section 6.4)
---------

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

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

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.

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

3.16. Initial Value of ssthresh
3.16.1. Description of the Problem

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

3.16.2. Text Changes to the Document

--------
Old text: (Section 7.2.1)
--------

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

--------
New text: (Section 7.2.1)
--------

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

3.16.3. Solution Description

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

3.17. Automatically Confirmed Addresses

3.17.1. Description of the Problem

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

3.17.2. Text Changes to the Document
Old text: (Section 5.4)

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

New text: (Section 5.4)

1) Any addresses passed to the sender of the INIT by its upper layer in the request to initialize an association are automatically considered to be CONFIRMED.

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.

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 following rules MUST be applied:

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

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

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

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.

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) Send

Format: SEND(association id, buffer address, byte count [,context]

-> result

...

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

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

E) Send


-> result

...

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

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

G) Receive

Format: RECEIVE(association id, buffer address, buffer size [,stream id])


...

o Stream Sequence Number - the Stream Sequence Number assigned by the sending SCTP peer.

o partial flag - if this returned flag is set to 1, then this Receive contains a partial delivery of the whole message. When this flag is set, the stream id and Stream Sequence Number MUST accompany this receive. When this flag is set to 0, it indicates that no more deliveries will be received for this Stream Sequence
New text: (Section 10.1)

G) Receive

Format: RECEIVED(association id, buffer address, buffer size 
[,stream id])
-> byte count [,transport address] [,stream id] [,stream sequence 
number] [,partial flag] [,delivery number] [,payload protocol-id]
...

- 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.

Old text: (Section 10.1)

N) Receive Unsent Message

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

- Stream Sequence Number - this value is returned indicating the
  Stream Sequence Number that was associated with the message.

- partial flag - if this returned flag is set to 1, then this
  message is a partial delivery of the whole message. When this
  flag is set, the stream id and Stream Sequence Number MUST
  accompany this receive. When this flag is set to 0, it indicates
  that no more deliveries will be received for this Stream Sequence
  Number.
New text: (Section 10.1)

---------

N) Receive Unsent Message

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

...

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.

---------

Old text: (Section 10.1)

---------

O) Receive Unacknowledged Message

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

...

o Stream Sequence Number - this value is returned indicating the Stream Sequence Number that was associated with the message.

o partial flag - if this returned flag is set to 1, then this message is a partial delivery of the whole message. When this flag is set, the stream id and Stream Sequence Number MUST accompany this receive. When this flag is set to 0, it indicates that no more deliveries will be received for this Stream Sequence Number.

---------

New text: (Section 10.1)

---------

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

... 

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.

3.21.3. Solution Description

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

3.22. Increase of partial_bytes_acked in Congestion Avoidance

3.22.1. Description of the Problem

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

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

o Duplicate TSNs are not considered in partial_bytes_acked although they confirm that the DATA chunks were successfully received by the peer.

3.22.2. Text Changes to the Document
Old text: (Section 7.2.2)

- 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)

- 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.

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

The following change is based on the change described in Section 3.6.
An endpoint shall 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 shall 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 shall 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 MAY report the failure to the upper layer and optionally report back all outstanding user data remaining in its outbound queue. The association is automatically closed when the peer endpoint becomes unreachable.

The following changes are based on [RFC4960].
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.

When an outstanding TSN is acknowledged or a HEARTBEAT sent to that address is acknowledged with a HEARTBEAT ACK, the endpoint SHOULD clear the error counter of the destination transport address to which the DATA chunk was last sent (or HEARTBEAT was sent), and SHOULD also report to the upper layer when an inactive destination address is marked as active. When the peer endpoint is multi-homed and the last chunk sent to it was a retransmission to an alternate address, there exists an ambiguity as to whether or not the acknowledgement could be credited to the address of the last chunk sent. However, this ambiguity does not seem to bear any significant consequence to SCTP behavior. If this ambiguity is undesirable, the transmitter MAY choose not to clear the error counter if the last chunk sent was a retransmission.

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.
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.

Old text: (Section 8.3)

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

New text: (Section 8.3)

Upon the receipt of the HEARTBEAT ACK, the sender of the HEARTBEAT SHOULD clear the error counter of the destination transport address to which the HEARTBEAT was sent, and mark the destination transport address as active if it is not so marked. The endpoint SHOULD report to the upper layer when an inactive destination address is marked as active due to the reception of the latest HEARTBEAT ACK. The receiver of the HEARTBEAT ACK SHOULD also clear the association overall error counter (as defined in Section 8.1).

Old text: (Section 9.2)

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

New text: (Section 9.2)
An endpoint SHOULD limit the number of retransmissions of the SHUTDOWN chunk to the protocol parameter ‘Association.Max.Retrans’. If this threshold is exceeded, the endpoint SHOULD destroy the TCB and SHOULD report the peer endpoint unreachable to the upper layer (and thus the association enters the CLOSED state).

Old text: (Section 9.2)

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

New text: (Section 9.2)

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

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.

--------
Old text: (Section 15)
--------

The following protocol parameters are RECOMMENDED:

\[\begin{align*}
\text{RTO.Initial} & \text{ - 3 seconds} \\
\text{RTO.Min} & \text{ - 1 second} \\
\text{RTO.Max} & \text{ - 60 seconds} \\
\text{Max.Burst} & \text{ - 4} \\
\text{RTO.Alpha} & \text{ - 1/8} \\
\text{RTO.Beta} & \text{ - 1/4} \\
\text{Valid.Cookie.Life} & \text{ - 60 seconds} \\
\text{Association.Max.Retrans} & \text{ - 10 attempts} \\
\text{Path.Max.Retrans} & \text{ - 5 attempts (per destination address)} \\
\text{Max.Init.Retransmits} & \text{ - 8 attempts} \\
\text{HB.interval} & \text{ - 30 seconds} \\
\text{HB.Max.Burst} & \text{ - 1}
\end{align*}\]

--------
New text: (Section 15)
--------

The following protocol parameters are RECOMMENDED:

\[\begin{align*}
\text{RTO.Initial} & \text{ - 3 seconds} \\
\text{RTO.Min} & \text{ - 1 second} \\
\text{RTO.Max} & \text{ - 60 seconds} \\
\text{Max.Burst} & \text{ - 4}
\end{align*}\]
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

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

--------
Old text: (Section 11.3)
--------

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

--------
New text: (Section 11.3)
--------

It is helpful for some firewalls if they can inspect just the first fragment of a fragmented SCTP packet and unambiguously determine whether it corresponds to an INIT chunk (for further information, please refer to [RFC1858]). Accordingly, we stress the requirements, stated in Section 3.1, that (1) an INIT chunk MUST NOT be bundled with any other chunk in a packet, and (2) a packet containing an INIT chunk MUST have a zero Verification Tag. The receiver of an INIT chunk MUST silently discard the INIT chunk and all further chunks if the INIT chunk is bundled with other chunks or the packet has a non-zero verification tag.

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.

-------
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 (partial_bytes_acked - 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 (partial_bytes_acked - cwnd). Next, cwnd is increased by 1*MTU.

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 \( \min(4*\text{MTU}, \max(2*\text{MTU}, 4380 \text{ bytes})) \).

---------
New text: (Section 7.2.1)
---------

o The initial cwnd before DATA transmission MUST be set to \( \min(4*\text{MTU}, \max(2*\text{MTU}, 4380 \text{ bytes})) \).

---------
Old text: (Section 7.2.1)
---------

o When the endpoint does not transmit data on a given transport address, the cwnd of the transport address should be adjusted to \( \max(\text{cwnd}/2, 4*\text{MTU}) \) per RTO.

---------
New text: (Section 7.2.1)
---------

o While the endpoint does not transmit data on a given transport address, the cwnd of the transport address SHOULD be adjusted to \( \max(\text{cwnd}/2, 4*\text{MTU}) \) once per RTO. Before the first cwnd adjustment, the ssthresh of the transport address SHOULD be set to the cwnd.

3.27.3. Solution Description

A rule about cwnd adjustment after a sufficiently long idle period is removed.

The text is updated to describe the ssthresh handling. When the idle period is detected, the cwnd value is stored to the ssthresh value.

3.28. Window Updates After Receiver Window Opens Up

3.28.1. Description of the Problem

The sending of SACK chunks for window updates is only indirectly referenced in [RFC4960], Section 6.2, where it is stated that an SCTP
receiver must not generate more than one SACK for every incoming packet, other than to update the offered window.

However, the sending of window updates when the receiver window opens up is necessary to avoid performance problems.

3.28.2. Text Changes to the Document

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

An SCTP receiver MUST NOT generate more than one SACK for every incoming packet, other than to update the offered window as the receiving application consumes new data.

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

An SCTP receiver MUST NOT generate more than one SACK for every incoming packet, other than to update the offered window as the receiving application consumes new data. When the window opens up, an SCTP receiver SHOULD send additional SACK chunks to update the window even if no new data is received. The receiver MUST avoid sending a large number of window updates, in particular large bursts of them. One way to achieve this is to send a window update only if the window can be increased by at least a quarter of the receive buffer size.

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.
o Initially it states that an endpoint should always transmit DATA chunks to the primary path. Then it states that the rule for transmittal of reply chunks should also be followed if the endpoint is bundling DATA chunks together with the reply chunk which contradicts with the first statement to always transmit DATA chunks to the primary path. Some implementations were having problems with it and sent DATA chunks bundled with reply chunks to a different destination address than the primary path that caused many gaps.

3.29.2. Text Changes to the Document

--------
Old text: (Section 6.4)
--------

An endpoint SHOULD transmit reply chunks (e.g., SACK, HEARTBEAT ACK, etc.) to the same destination transport address from which it received the DATA or control chunk to which it is replying. This rule should also be followed if the endpoint is bundling DATA chunks together with the reply chunk.

However, when acknowledging multiple DATA chunks received in packets from different source addresses in a single SACK, the SACK chunk may be transmitted to one of the destination transport addresses from which the DATA or control chunks being acknowledged were received.

--------
New text: (Section 6.4)
--------

An endpoint SHOULD transmit reply chunks (e.g., INIT ACK, COOKIE ACK, HEARTBEAT ACK, etc.) in response to control chunks to the same destination transport address from which it received the control chunk to which it is replying.

The selection of the destination transport address for packets containing SACK chunks is implementation dependent. However, an endpoint SHOULD NOT vary the destination transport address of a SACK when it receives DATA chunks coming from the same source address.

When acknowledging multiple DATA chunks received in packets from different source addresses in a single SACK, the SACK chunk MAY be transmitted to one of the destination transport addresses from which the DATA or control chunks being acknowledged were received.
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.

---

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).

3.30.3. Solution Description

Now Section 1.3, Key Terms, includes explanations of outstanding data, data in flight and flightsize key terms. Section 6.1 is corrected to properly use the outstanding data term.

3.31. CWND Degradation due to Max.Burst

3.31.1. Description of the Problem

Some implementations were experiencing a degradation of cwnd because of the Max.Burst limit. This was due to misinterpretation of the suggestion in [RFC4960], Section 6.1, on how to use the Max.Burst parameter when calculating the number of packets to transmit.

3.31.2. Text Changes to the Document
Old text: (Section 6.1)

D) When the time comes for the sender to transmit new DATA chunks, the protocol parameter Max.Burst SHOULD be used to limit the number of packets sent. The limit MAY be applied by adjusting cwnd as follows:

\[
\text{if } ((\text{flightsize} + \text{Max.Burst} \times \text{MTU}) < \text{cwnd}) \quad \text{cwnd} = \text{flightsize} + \text{Max.Burst} \times \text{MTU}
\]

Or it MAY be applied by strictly limiting the number of packets emitted by the output routine.

New text: (Section 6.1)

D) When the time comes for the sender to transmit new DATA chunks, the protocol parameter Max.Burst SHOULD be used to limit the number of packets sent. The limit MAY be applied by adjusting cwnd temporarily as follows:

\[
\text{if } ((\text{flightsize} + \text{Max.Burst} \times \text{MTU}) < \text{cwnd}) \quad \text{cwnd} = \text{flightsize} + \text{Max.Burst} \times \text{MTU}
\]

Or it MAY be applied by strictly limiting the number of packets emitted by the output routine. When calculating the number of packets to transmit and particularly using the formula above, cwnd SHOULD NOT be changed permanently.

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
- SACK.Delay - 200 milliseconds

---------

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

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.

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) Receive

Format: RECEIVE(association id, buffer address, buffer size [,stream id])

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

G) Receive

Format: RECEIVE(association id, buffer address, buffer size [,stream id])

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.

---------

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

M) Set Protocol Parameters

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

This primitive allows the local SCTP to customize the protocol parameters.

Mandatory attributes:

- association id - local handle to the SCTP association.
- protocol parameter list - the specific names and values of the protocol parameters (e.g., Association.Max.Retrans; see Section 15) that the SCTP user wishes to customize.

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

M) Set Protocol Parameters

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

This primitive allows the local SCTP to customize the protocol parameters.
Mandatory attributes:

- association id - local handle to the SCTP association.

- protocol parameter list - the specific names and values of the
  protocol parameters (e.g., Association.Max.Retrans; see Section
  15, or other parameters like the DSCP) that the SCTP user wishes
  to customize.

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
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".

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.

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 ICMP type 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.

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
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.

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.

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.

3.38.3. Solution Description

Text was added to clarify how the cwnd limit should be handled.

3.39. Zero Window Probing

3.39.1. Description of the Problem

The text describing zero window probing was not clearly handling the case where the window was not zero, but too small for the next DATA chunk to be transmitted. Even in this case, zero window probing has to be performed to avoid deadlocks.
3.39.2. Text Changes to the Document

-------

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

A) At any given time, the data sender MUST NOT transmit new data to any destination transport address if its peer’s rwnd indicates that the peer has no buffer space (i.e., rwnd is 0; see Section 6.2.1). However, regardless of the value of rwnd (including if it is 0), the data sender can always have one DATA chunk in flight to the receiver if allowed by cwnd (see rule B, below). This rule allows the sender to probe for a change in rwnd that the sender missed due to the SACK’s having been lost in transit from the data receiver to the data sender.

When the receiver’s advertised window is zero, this probe is called a zero window probe. Note that a zero window probe SHOULD only be sent when all outstanding DATA chunks have been cumulatively acknowledged and no DATA chunks are in flight. Zero window probing MUST be supported.

-------

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

A) At any given time, the data sender MUST NOT transmit new data to any destination transport address if its peer’s rwnd indicates that the peer has no buffer space (i.e., rwnd is smaller than the size of the next DATA chunk; see Section 6.2.1).

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

When the receiver has no buffer space, this probe is called a zero window probe. Note that a zero window probe SHOULD only be sent when all outstanding DATA chunks have been cumulatively acknowledged and no DATA chunks are in flight. Zero window probing MUST be supported.

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.

--------
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.

--------
Old text: (Appendix A)
--------
[RFC3168] details negotiation of ECN during the SYN and SYN-ACK stages of a TCP connection.

--------
New text: (Appendix A)
--------

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.

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.

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

--------
Old text: (Section 3.3.2)
--------

Note 3: An INIT chunk MUST NOT contain more than one Host Name Address parameter. Moreover, the sender of the INIT MUST NOT combine any other address types with the Host Name Address in the INIT. The receiver of INIT MUST ignore any other address types if the Host Name Address parameter is present in the received INIT chunk.

--------
New text: (Section 3.3.2)
--------

Note 3: An INIT chunk MUST NOT contain the Host Name Address parameter. The receiver of an INIT chunk containing a Host Name Address parameter MUST send an ABORT and MAY include an Error Cause indicating an Unresolvable Address.

--------
Old text: (Section 3.3.2.1)
--------

The sender of INIT uses this parameter to pass its Host Name (in place of its IP addresses) to its peer. The peer is responsible for resolving the name. Using this parameter might make it more likely for the association to work across a NAT box.

--------
New text: (Section 3.3.2.1)
--------

The sender of an INIT chunk MUST NOT include this parameter. The usage of the Host Name Address parameter is deprecated.

--------
Old text: (Section 3.3.2.1)
--------
Internet-Draft         RFC 4960 Errata and Issues              July 2018

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.

---------

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.

---------

Old text: (Section 5.1.2)
---------

B) If there is a Host Name parameter present in the received INIT or INIT ACK chunk, the endpoint shall resolve that host name to a list of IP address(es) and derive the transport address(es) of this peer by combining the resolved IP address(es) with the SCTP source port.

The endpoint MUST ignore any other IP Address parameters if they are also present in the received INIT or INIT ACK chunk.

The time at which the receiver of an INIT resolves the host name has potential security implications to SCTP. If the receiver of
an INIT resolves the host name upon the reception of the chunk, and the mechanism the receiver uses to resolve the host name involves potential long delay (e.g., DNS query), the receiver may open itself up to resource attacks for the period of time while it is waiting for the name resolution results before it can build the State Cookie and release local resources.

Therefore, in cases where the name translation involves potential long delay, the receiver of the INIT MUST postpone the name resolution till the reception of the COOKIE ECHO chunk from the peer. In such a case, the receiver of the INIT SHOULD build the State Cookie using the received Host Name (instead of destination transport addresses) and send the INIT ACK to the source IP address from which the INIT was received.

The receiver of an INIT ACK shall always immediately attempt to resolve the name upon the reception of the chunk.

The receiver of the INIT or INIT ACK MUST NOT send user data (piggy-backed or stand-alone) to its peer until the host name is successfully resolved.

If the name resolution is not successful, the endpoint MUST immediately send an ABORT with "Unresolvable Address" error cause to its peer. The ABORT SHALL be sent to the source IP address from which the last peer packet was received.

----------

New text: (Section 5.1.2)

----------

B) If there is a Host Name parameter present in the received INIT or INIT ACK chunk, the endpoint MUST immediately send an ABORT and MAY include an Error Cause indicating an Unresolvable Address to its peer. The ABORT SHALL be sent to the source IP address from which the last peer packet was received.

----------

Old text: (Section 11.2.4.1)

----------

The use of the host name feature in the INIT chunk could be used to flood a target DNS server. A large backlog of DNS queries, resolving the host name received in the INIT chunk to IP addresses, could be accomplished by sending INITs to multiple hosts in a given domain. In addition, an attacker could use the host name feature in an indirect attack on a third party by sending large numbers of INITs to random hosts containing the host name of the target. In addition to
the strain on DNS resources, this could also result in large numbers
of INIT ACKs being sent to the target. One method to protect against
this type of attack is to verify that the IP addresses received from
DNS include the source IP address of the original INIT. If the list
of IP addresses received from DNS does not include the source IP
address of the INIT, the endpoint MAY silently discard the INIT.
This last option will not protect against the attack against the DNS.

New text: (Section 11.2.4.1)

The support of the Host Name Address parameter has been removed from
the protocol. Endpoints receiving INIT or INIT ACK chunks containing
the Host Name Address parameter MUST send an ABORT chunk in response
and MAY include an Error Cause indicating an Unresolvable Address.

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.

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.

--------
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.

Please note that Sections 14.2, 14.3, 14.4, and 14.5 need to be renumbered.

3.43.3. Solution Description

[RFC6096] was integrated and the reference updated to [RFC8126].

3.44. Integration of RFC 6335

3.44.1. Description of the Problem

[RFC6335] updates [RFC4960] by updating Procedures for the Port Numbers Registry. This should be integrated into the base specification. While there, update the reference to the RFC giving guidelines for writing IANA sections to [RFC8126].

3.44.2. Text Changes to the Document

---------
Old text: (Section 14.5)
---------

SCTP services may use contact port numbers to provide service to unknown callers, as in TCP and UDP. IANA is therefore requested to open the existing Port Numbers registry for SCTP using the following rules, which we intend to mesh well with existing Port Numbers registration procedures. An IESG-appointed Expert Reviewer supports IANA in evaluating SCTP port allocation requests, according to the procedure defined in [RFC2434].

Port numbers are divided into three ranges. The Well Known Ports are those from 0 through 1023, the Registered Ports are those from 1024 through 49151, and the Dynamic and/or Private Ports are those from
49152 through 65535. Well Known and Registered Ports are intended for use by server applications that desire a default contact point on a system. On most systems, Well Known Ports can only be used by system (or root) processes or by programs executed by privileged users, while Registered Ports can be used by ordinary user processes or programs executed by ordinary users. Dynamic and/or Private Ports are intended for temporary use, including client-side ports, out-of-band negotiated ports, and application testing prior to registration of a dedicated port; they MUST NOT be registered.

The Port Numbers registry should accept registrations for SCTP ports in the Well Known Ports and Registered Ports ranges. Well Known and Registered Ports SHOULD NOT be used without registration. Although in some cases -- such as porting an application from TCP to SCTP -- it may seem natural to use an SCTP port before registration completes, we emphasize that IANA will not guarantee registration of particular Well Known and Registered Ports. Registrations should be requested as early as possible.

Each port registration SHALL include the following information:

- 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.
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].

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

The following format MUST be used for the DATA chunk:

```
+-------------------------------+-------------------------------+
| TSN                           | Payload Protocol Identifier   |
+-------------------------------+-------------------------------+
| Stream Sequence Number n      | User Data (seq n of Stream S) |
+-------------------------------+-------------------------------+
| Stream Identifier S           |                               |
+-------------------------------+-------------------------------+
| Reserved|U|B|E| Length                        |
+-------------------------------+-------------------------------+
| Type = 0                       |                               |
+-------------------------------+-------------------------------+
```
Reserved: 5 bits

Should be set to all ‘0’s and ignored by the receiver.

--------

New text: (Section 3.3.1)
--------

0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Type = 0    |  Res  |I|U|B|E|    Length                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              TSN                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Stream Identifier S      |   Stream Sequence Number n    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Payload Protocol Identifier                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \n/                 User Data (seq n of Stream S)                 /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Res: 4 bits

SHOULD be set to all ‘0’s and ignored by the receiver.

I bit: 1 bit

The (I)mmediate Bit MAY be set by the sender, whenever the sender of a DATA chunk can benefit from the corresponding SACK chunk being sent back without delay. See [RFC7053] for a discussion about

--------

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).
o The sender is in the SHUTDOWN-PENDING state.
o The sending of a DATA chunk fills the congestion or receiver window.

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

Note: The SHUTDOWN chunk does not contain Gap Ack Block fields. Therefore, the endpoint should use a SACK instead of the SHUTDOWN chunk to acknowledge DATA chunks received out of order.

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

Note: The SHUTDOWN chunk does not contain Gap Ack Block fields. Therefore, the endpoint SHOULD use a SACK instead of the SHUTDOWN chunk to acknowledge DATA chunks received out of order.

Upon receipt of an SCTP packet containing a DATA chunk with the I bit set, the receiver SHOULD NOT delay the sending of the corresponding SACK chunk, i.e., the receiver SHOULD immediately respond with the corresponding SACK chunk.

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

E) Send

-> result

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

E) Send

Format: SEND(association id, buffer address, byte count [,context] [,stream id] [,life time] [,destination transport address] etc.)

---------
New text: (Append optional parameter in Subsection E of Section 10.1)

---

- sack immediately - set the I bit on the last DATA chunk used for sending buffer.

Please note that the change in Section 6.2 is only about adding a paragraph.

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 B)

---

#ifndef __crc32cr_table_h__
#define __crc32cr_table_h__

#define CRC32C_POLY 0x1EDC6F41

---

/* Example of the crc table file */
#ifndef __crc32cr_table_h__
#define __crc32cr_table_h__

#define CRC32C_POLY 0x1EDC6F41

---

/* 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_xoror=0x00000000 */

------------------------------------------
*/
#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])

unsigned long  crc_c[256] =
{
  0x00000000L, 0xF26B8303L, 0xE13B70F7L, 0x1350F3F4L,
  0xC79A971FL, 0x35F1141CL, 0x26A1E7E8L, 0xD4CA64EBL,
  0x8AD958CFL, 0x78B2DBCCL, 0x6BE22838L, 0x9989AB3BL,
  0x4D43CFD0L, 0xBF284CD3L, 0xAC78BF27L, 0x5E133C24L,
  0x105EC76FL, 0xE235446CL, 0xF165B798L, 0x030E349BL,
  0xD7C45070L, 0x25AFD373L, 0x36FF2087L, 0xC494A384L,
  0x9A879FA0L, 0x68EC1CA3L, 0x7BBCEF57L, 0x89D76C54L,
  0x5D1D08BFL, 0xAF768BBCL, 0xBC267848L, 0x4E4DFB48L,
  0x20BD8EDEL, 0xD2D60DDDL, 0xC186FE29L, 0x33ED7D2AL,
  0xE72719C1L, 0x154C9AC2L, 0x061C6936L, 0xF477EA35L,
  0xAA64D611L, 0x580F5512L, 0x4B5FA6E6L, 0xB93425E5L,
  0x6DFE410EL, 0x9F95C20DL, 0x8CC531F9L, 0x7EAEB2FAL,
  0x30E349B1L, 0xC288CAB2L, 0xD1D83946L, 0x23B3BA45L,
  0x7F79DEAEL, 0x05125DADL, 0x1642AE59L, 0xEA492D5AL,
  0xB5A317EL, 0x4851927DL, 0x5B016189L, 0xA96AE28AL,
  0x77A08661L, 0x8F6CB0562L, 0x9C9BF696L, 0x6EEF0759SL,
  0xA47B19C8L, 0x3B3109EBFL, 0xA0406D4BL, 0x522BEE48L,
  0x86E18AA3L, 0x748A09A0L, 0x67DAFA54L, 0x95B17957L,
  0xC0B64573L, 0x39C9C670L, 0x2A993584L, 0xD8F2B687L,
  0x0C38D26CL, 0xFE53451FL, 0xED03A29BL, 0x1F682198L,
  0x5125DA3DL, 0xa345E9D0L, 0xB01EAA24L, 0x42752927L,
  0x96BF4DCCL, 0x64D4CECFL, 0x77843D3BL, 0x85EFBE8BL,
  0xDBFC821CL, 0x2997011FL, 0x3AC7AC1E8L, 0x2C8AC71E8L,
  0x1C661503L, 0xE0D96000L, 0xF5D65F4FL, 0x0F36E6F7L,
  0x61C69362L, 0x93A13106L, 0x80FDE395L, 0x72966096L,
  0xA65C047DL, 0x75A7877EL, 0x4767748AL, 0xB50CF789L,
  0xEB1FBCADL, 0x197448AE1L, 0x0A24BB5AL, 0xBF84359FL,
  0x2C85C82BL, 0x7DEEEFB1L, 0xCDBE2C45L, 0x3FD5AF4DL,
  0x7198540DL, 0x83F3D70EL, 0x90A324FAL, 0x62C8A79FL,
  0xB602C312L, 0x44690411L, 0x5739B3E5L, 0xA5523060L,
  0xPB410CFC2L, 0x092A8FC1L, 0x1A7A7C35L, 0xE811FF36L,
  0x3CD9B9DDL, 0xCEB018DEL, 0xEDDE0EB2AL, 0x2F8B6829L,
  0x86F3B78BL, 0x709DB87BL, 0x63CD488FL, 0x91A6C8C8L,
  0x456CA6C7L, 0xB7072E64L, 0xA57DC90L, 0x563C5F93L,
  0x082F63B7L, 0xEA440EB4L, 0xE9143400L, 0x1B7F9043L,
  0xCFB5F4A8L, 0x3EDE77ABAL, 0x2E88E845FL, 0x0DCE5075CL,
  0x92A8FC1L, 0x60C3F14L, 0x73938CE0L, 0x81F80E43L,
  0x55326B08L, 0xA759E800L, 0xB4091BFCL, 0x466298FCL,
  0x1871A480L, 0xEA1A27DBL, 0xF94AD42FL, 0x0BB1572CL,
  0xDFB33CC7L, 0x2D80B0C4L, 0x3ED04330L, 0xCCBBC033L,
  0xA24BB5A6L, 0x502036A5L, 0x4370C551L, 0xBB1B4652L,
  0x65D122B9L, 0x979AA1BAL, 0x84EA524EL, 0x7681D14DL,
  0x2892ED69L, 0xDAF966EAL, 0xC9A99D9EL, 0x3BC21E9DL,

New text: (Appendix B)

---

# 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 */
#ifdef __crc32cr_h_
#define __crc32cr_h_

#define CRC32C_POLY 0x1EDC6F41UL
#define crc32c(c,d) ((c=(c>>8)^crc32c[(c^(d))&0xFF]))

uint32_t crc32c[256] = {

/* Example of table build routine */

#include <stdio.h>
#include <stdlib.h>

#define OUTPUT_FILE   "crc32cr.h"
#define CRC32C_POLY    0x1EDC6F41L

FILE *tf;
unsigned long
reflect_32 (unsigned long b) {
    int i;
    unsigned long rw = 0L;

    for (i = 0; i < 32; i++){
        if (b & 1)
            rw |= 1 << (31 - i);
        b >>= 1;
    }
    return (rw);
}

unsigned long
build_crc_table (int index)
{  
  int i;
  unsigned long rb;

  rb = reflect_32(index);

  for (i = 0; i < 8; i++) {
    if (rb & 0x80000000L)
      rb = (rb << 1) ^ CRC32C_POLY;
    else
      rb <<= 1;
  }

  return (reflect_32(rb));
}

main ()
{
  int i;

  printf("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"");
  fprintf(tf, "#define CRC32C_POLY 0x%08lX
", CRC32C_POLY);
  fprintf(tf,
"#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])\n");
  fprintf(tf, "unsigned long  crc_c[256] =\n{\n");
  for (i = 0; i < 256; i++){
    fprintf(tf, "0x%08lXL, ", build_crc_table(i));
    if ((i & 3) == 3)
      fprintf(tf, "\n"");
  }
  fprintf(tf, "};\n\n#endif\n");

  if (fclose(tf) != 0)
    printf("Unable to close <%s>.
", OUTPUT_FILE);
  else
    printf("The CRC-32c table has been written to <%s>.
", OUTPUT_FILE);
}

--------

New text: (Appendix B)
/* Example of table build routine */

#include <stdio.h>
#include <stdlib.h>

#define OUTPUT_FILE   "crc32cr.h"
#define CRC32C_POLY   0x1EDC6F41UL

static FILE *tf;

static uint32_t
reflect_32(uint32_t b)
{
    int i;
    uint32_t rw = 0UL;

    for (i = 0; i < 32; i++) {
        if (b & 1)
            rw |= 1 << (31 - i);
        b >>= 1;
    }
    return (rw);
}

static uint32_t
build_crc_table(int index)
{
    int i;
    uint32_t rb;

    rb = reflect_32(index);

    for (i = 0; i < 8; i++) {
        if (rb & 0x80000000UL)
            rb = (rb << 1) ^ (uint32_t)CRC32C_POLY;
        else
            rb <<= 1;
    }
    return (reflect_32(rb));
}

int
main (void)
{
    int i;
printf("\nGenerating CRC-32c table file <%s>\n",
OUTPUT_FILE);
if ((tf = fopen(OUTPUT_FILE, "w")) == NULL) {
    printf ("Unable to open %s\n", OUTPUT_FILE);
    exit (1);
} 
fprintf(tf, "#ifndef __crc32cr_h__\n");
fprintf(tf, "#define __crc32cr_h__\n"");
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, "# endif \n");
} 
fprintf(tf, "#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])\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("The CRC-32c table has been written to <%s>.\n", OUTPUT_FILE);
}
-------
Old text: (Appendix B)
-------

/* 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]);
    }

    return result;
}
result = ~crc32;

/* result now holds the negated polynomial remainder;
 * since the table and algorithm is "reflected" [williams95].
 * That is, result has the same value as if we mapped the message
 * to a polynomial, computed the host-bit-order polynomial
 * remainder, performed final negation, then did an end-for-end
 * bit-reversal.
 * Note that a 32-bit bit-reversal is identical to four inplace
 * 8-bit reversals followed by an end-for-end byteswap.
 * In other words, the bytes of each bit are in the right order,
 * but the bytes have been byteswapped. So we now do an explicit
 * byteswap. On a little-endian machine, this byteswap and
 * the final ntohl cancel out and could be elided.
 */

byte0 = result & 0xff;
byte1 = (result>>8) & 0xff;
byte2 = (result>>16) & 0xff;
byte3 = (result>>24) & 0xff;
crc32 = ((byte0 << 24) |
          (byte1 << 16) |
          (byte2 << 8) |
          byte3);
return (crc32);
}

int insert_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    unsigned long crc32;
    message = (SCTP_message *) buffer;
    message->common_header.checksum = 0L;
    crc32 = generate_crc32c(buffer,length);
    /* and insert it into the message */
    message->common_header.checksum = htonl(crc32);
    return 1;
}

int validate_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    unsigned int i;
    unsigned long original_crc32;
    unsigned long crc32 = ~0L;
/* save and zero checksum */
message = (SCTP_message *) buffer;
original_crc32 = ntohl(message->common_header.checksum);
message->common_header.checksum = 0L;
crc32 = generate_crc32c(buffer,length);
return ((original_crc32 == crc32)? 1 : -1);
}

New text: (Appendix B)

/*********
/* Example of crc insertion */

#include "crc32cr.h"

uint32_t
generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    uint32_t crc32 = 0xffffffffUL;
    uint32_t result;
    uint8_t byte0, byte1, byte2, byte3;
    for (i = 0; i < length; i++) {
        CRC32C(crc32, buffer[i]);
    }

    result = ˜crc32;

    /* result now holds the negated polynomial remainder;
    * since the table and algorithm is "reflected" [williams95].
    * That is, result has the same value as if we mapped the message
    * to a polynomial, computed the host-bit-order polynomial
    * remainder, performed final negation, then did an end-for-end
    * bit-reversal.
    * Note that a 32-bit bit-reversal is identical to four inplace
    * 8-bit reversals followed by an end-for-end byteswap.
    * In other words, the bits of each byte are in the right order,
    * but the bytes have been byteswapped. So we now do an explicit
    * byteswap. On a little-endian machine, this byteswap and
    * the final ntohl cancel out and could be elided.
    */

    byte0 = result & 0xff;
    byte1 = (result>>8) & 0xff;
    byte2 = (result>>16) & 0xff;
    byte3 = (result>>24) & 0xff;
\[
\begin{align*}
crc32 &= ((byte0 \ll 24) \mid \\
&\quad (byte1 \ll 16) \mid \\
&\quad (byte2 \ll 8) \mid \\
&\quad byte3); \\
\end{align*}
\]

return (crc32);
}

int
insert_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    uint32_t crc32;
    message = (SCTP_message *) buffer;
    message->common_header.checksum = 0UL;
    crc32 = generate_crc32c(buffer,length);
    /* and insert it into the message */
    message->common_header.checksum = htonl(crc32);
    return 1;
}

int
validate_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    unsigned int i;
    uint32_t original_crc32;
    uint32_t crc32;

    /* save and zero checksum */
    message = (SCTP_message *)buffer;
    original_crc32 = ntohl(message->common_header.checksum);
    message->common_header.checksum = 0L;
    crc32 = generate_crc32c(buffer, length);
    return ((original_crc32 == crc32) ? 1 : -1);
}

<CODE ENDS>

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.

---------
Old text: (Section 3.3.4)
---------

Gap Ack Blocks:

These fields contain the Gap Ack Blocks. They are repeated for each Gap Ack Block up to the number of Gap Ack Blocks defined in the Number of Gap Ack Blocks field. All DATA chunks with TSNs greater than or equal to (Cumulative TSN Ack + Gap Ack Block Start) and less than or equal to (Cumulative TSN Ack + Gap Ack Block End) of each Gap Ack Block are assumed to have been received correctly.

---------
New text: (Section 3.3.4)
Gap Ack Blocks:

These fields contain the Gap Ack Blocks. They are repeated for each Gap Ack Block up to the number of Gap Ack Blocks defined in the Number of Gap Ack Blocks field. All DATA chunks with TSNs greater than or equal to (Cumulative TSN Ack + Gap Ack Block Start) and less than or equal to (Cumulative TSN Ack + Gap Ack Block End) of each Gap Ack Block are assumed to have been received correctly. Gap Ack Blocks SHOULD be isolated. That means that the DATA chunks with TSN equal to (Cumulative TSN Ack + Gap Ack Block Start - 1) and (Cumulative TSN Ack + Gap Ack Block End + 1) have not been received.

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.
3.48.3. Solution Description

The data sender is required to limit the number of ordered user messages within the current send window.

3.49. Update RFC 2119 Boilerplate

3.49.1. Description of the Problem

The text to be used to refer to the [RFC2119] terms has been updated by [RFC8174].

3.49.2. Text Changes to the Document

--------- Old text: (Section 2) ---------

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

--------- New text: (Section 2) ---------

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

3.49.3. Solution Description

The text has been updated to the one specified in [RFC8174].

3.50. Missed Text Removal

3.50.1. Description of the Problem

When integrating the changes to Section 7.2.4 of [RFC2960] as described in Section 2.8.2 of [RFC4460] some text was not removed and is therefore still in [RFC4960].
3.50.2. Text Changes to the Document

--------
Old text: (Section 7.2.4)
--------

A straightforward implementation of the above keeps a counter for each TSN hole reported by a SACK. The counter increments for each consecutive SACK reporting the TSN hole. After reaching 3 and starting the Fast-Retransmit procedure, the counter resets to 0. Because cwnd in SCTP indirectly bounds the number of outstanding TSN’s, the effect of TCP Fast Recovery is achieved automatically with no adjustment to the congestion control window size.

--------
New text: (Section 7.2.4)
--------

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
Propagating Explicit Congestion Notification Across IP Tunnel Headers Separated by a Shim

draft-ietf-tsvwg-rfc6040update-shim-06

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.

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 19, 2018.

Copyright Notice

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

RFC 6040 on "Tunnelling of Explicit Congestion Notification" [RFC6040] made the rules for propagation of Explicit Congestion Notification (ECN [RFC3168]) consistent for all forms of IP in IP tunnel.

A common pattern for many tunnelling protocols is to encapsulate an inner IP header (v4 or v6) with shim header(s) then an outer IP header (v4 or v6). Some of these shim headers are designed as generic encapsulations, so they do not necessarily directly encapsulate an inner IP header. Instead they can encapsulate headers such as link-layer (L2) protocols that in turn often encapsulate IP.
To clear up confusion, this specification clarifies that the scope of RFC 6040 includes any IP-in-IP tunnel, including those with shim header(s) and other encapsulations between the IP headers. Where necessary, it updates the specifications of the relevant encapsulation protocols with the specific text necessary to comply with RFC 6040.

This specification also updates RFC 6040 to state how operators ought to configure a legacy tunnel ingress to avoid unsafe system configurations.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119] when, and only when, they appear in all capitals, as shown here.

This specification uses the terminology defined in RFC 6040 [RFC6040].

3. Scope of RFC 6040

In section 1.1 of RFC 6040, its scope is defined as:

"...ECN field processing at encapsulation and decapsulation for any IP-in-IP tunnelling, whether IPsec or non-IPsec tunnels. It applies irrespective of whether IPv4 or IPv6 is used for either the inner or outer headers. ..."

This was intended to include cases where shim header(s) sit between the IP headers. Many tunnelling implementers have interpreted the scope of RFC 6040 as it was intended, but it is ambiguous. Therefore, this specification updates RFC 6040 by adding the following scoping text after the sentences quoted above:

It applies in cases where an outer IP header encapsulates an inner IP header either directly or indirectly by encapsulating other headers that in turn encapsulate (or might encapsulate) an inner IP header.

There is another problem with the scope of RFC 6040. Like many IETF specifications, RFC 6040 is written as a specification that implementations can choose to claim compliance with. This means it does not cover two important cases:
1. those cases where it is infeasible for an implementation to access an inner IP header when adding or removing an outer IP header;

2. those implementations that choose not to propagate ECN between IP headers.

However, the ECN field is a non-optional part of the IP header (v4 and v6). So any implementation that creates an outer IP header has to give the ECN field some value. There is only one safe value a tunnel ingress can use if it does not know whether the egress supports propagation of the ECN field; it has to clear the ECN field in any outer IP header to 0b00.

However, an RFC has no jurisdiction over implementations that choose not to comply with it or cannot comply with it, including all those implementations that pre-dated the RFC. Therefore it would have been unreasonable to add such a requirement to RFC 6040. Nonetheless, to ensure safe propagation of the ECN field over tunnels, it is reasonable to add requirements on operators, to ensure they configure their tunnels safely (where possible). Before stating these configuration requirements in Section 4, the factors that determine whether propagating ECN is feasible or desirable will be briefly introduced.

3.1. Feasibility of ECN Propagation between Tunnel Headers

In many cases shim header(s) and an outer IP header are always added to (or removed from) an inner IP packet as part of the same procedure. We call this a tightly coupled shim header. Processing the shim and outer together is often necessary because the shim(s) are not sufficient for packet forwarding in their own right; not unless complemented by an outer header. In these cases it will often be feasible for an implementation to propagate the ECN field between the IP headers.

In some cases a tunnel adds an outer IP header and a tightly coupled shim header to an inner header that is not an IP header, but that in turn encapsulates an IP header (or might encapsulate an IP header). For instance an inner Ethernet (or other link layer) header might encapsulate an inner IP header as its payload. We call this a tightly coupled shim over an encapsulating header.

Digging to arbitrary depths to find an inner IP header within an encapsulation is strictly a layering violation so it cannot be a required behaviour. Nonetheless, some tunnel endpoints already look within a L2 header for an IP header, for instance to map the Diffserv codepoint between an encapsulated IP header and an outer IP header.
In such cases at least, it should be feasible to also (independently) propagate the ECN field between the same IP headers. Thus, access to the ECN field within an encapsulating header can be a useful and benign optimization. The guidelines in section 6 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 just before the last paragraph of section 4.3:

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.

In order that the network operator can comply with the above safety rules, even if a tunnel ingress does not claim to support RFC 6040, RFC 4301 or the full functionality mode of RFC 3168, the implementation of the tunnel ingress:

- MUST make propagation of the ECN field between inner and outer IP headers independent of any configuration of Diffserv codepoint propagation;

- SHOULD zero the outer ECN field in its default configuration.

There might be concern that the above "MUST" makes compliant equipment non-compliant at a stroke. However, any equipment that is still treating the ToS octet (IPv4) or the 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 codepoint (DSCP) [RFC2474]. 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.eastlake-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 SCCRP (Start-Control-Connection-Request and Start-Control-Connection-Reply). This AVP MAY be hidden (the H-bit set to 0 or 1) and is optional (M-bit not set). The length (before hiding) of this AVP MUST be 8 octets. The Vendor ID is the IETF Vendor ID of 0.

Bit 15 of the Value field of the LCCE Capability AVP is defined as the ECN Capability flag (E). When the ECN Capability flag is set to 1, it indicates that the sender supports ECN propagation. When the ECN Capability flag is cleared to zero, or when no LCCE Capability AVP is present, it indicates that the sender does not support ECN propagation. All the other bits are reserved. They MUST be cleared to zero when sent and ignored when received or forwarded.

An LCCE initiating a control connection will send a Start-Control-Connection-Request (SCCRQ) containing an LCCE Capability AVP with the ECN Capability flag set to 1. If the tunnel terminator supports ECN, it will return a Start-Control-Connection-Reply (SCCRP) that also includes an LCCE Capability AVP with the ECN Capability flag set to 1. Then, for any sessions created by that control connection, both ends of the tunnel can use the normal mode of RFC 6040, i.e. it can copy the IP ECN field from inner to outer when encapsulating data packets.

If, on the other hand, the tunnel terminator does not support ECN it will ignore the ECN flag in the LCCE Capability AVP and send an SCCRP to the tunnel initiator without a Capability AVP (or with a Capability AVP but with the ECN Capability flag cleared to zero). The tunnel initiator interprets the absence of the ECN Capability flag in the SCCRP as an indication that the tunnel terminator is incapable of supporting ECN. When encapsulating data packets for any sessions created by that control connection, the tunnel initiator will then use the compatibility mode of RFC 6040 to clear the ECN field of the outer IP header to 0b00.
If the tunnel terminator does not support this ECN extension, the network operator is still expected to configure it to comply with the safety provisions set out in Section 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 (MIPv4) [RFC5944], Mobile IPv6 (MIPv6) [RFC6275], Proxy Mobile IP (PMIPv) [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 predecessors (RFC 4301 or the full functionality mode of RFC 3168) MUST zero the ECN field in the outer IPv6 header."

Briscoe                Expires September 19, 2018              [Page 13]
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 tunnelled over AMT.

To comply with RFC 6040, an AMT relay and gateway will follow the rules for propagation of the ECN field at ingress and egress respectively, as described in Section 4 of RFC 6040 [RFC6040].

Before encapsulating any data packets, RFC 6040 requires an ingress AMT relay to check that the egress AMT gateway supports ECN propagation as defined in RFC 6040 or one of its compatible predecessors (RFC 4301 or the full functionality mode of RFC 3168). If the egress gateway supports ECN, the ingress relay can use the normal mode of encapsulation (copying the IP ECN field from inner to outer). Otherwise, the ingress relay has to use compatibility mode, which means it has to clear the outer ECN field to zero [RFC6040].

An AMT tunnel is created dynamically (not manually), so the relay will need to determine the remote gateway’s support for ECN using the ECN capability declaration defined in Section 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:

+----------------+----------------+-----------+
| Attribute Type | Description    | Reference |
|----------------+----------------+-----------|
| ZZ             | ECN Capability | RFCXXXX   |
+----------------+----------------+-----------+

[TO BE REMOVED: This registration should take place at the following location: https://www.iana.org/assignments/l2tp-parameters/l2tp-parameters.xhtml]

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]


10.2. Informative References


Author’s Address

Bob Briscoe
CableLabs
UK

EMail: ietf@bobbriscoe.net
URI: http://bobbriscoe.net/
Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC) Schemes for FECFRAME
draft-ietf-tsvwg-rlc-fec-scheme-09

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, as defined in [fecframe-ext]. 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 March 23, 2019.

Copyright Notice

Copyright (c) 2018 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 ........... 3
   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. Procedures ............................................... 7
   3.1. Possible Parameter Derivations ....................... 7
       3.1.1. Case of a CBR Real-Time Flow ................... 8
       3.1.2. Other Types of Real-Time Flow .................. 10
       3.1.3. Case of a Non Real-Time Flow .................. 11
   3.2. ADU, ADUI and Source Symbols Mappings ................ 11
   3.3. Encoding Window Management .......................... 13
   3.4. Pseudo-Random Number Generator (PRNG) ............... 13
   3.5. Coding Coefficients Generation Function ............. 15
   3.6. Finite Fields Operations ............................ 17
       3.6.1. Finite Field Definitions ....................... 17
       3.6.2. Linear Combination of Source Symbols Computation .. 17
4. Sliding Window RLC FEC Scheme over GF(2^8) for Arbitrary
   Packet Flows ............................................. 18
   4.1. Formats and Codes ................................. 18
       4.1.1. FEC Framework Configuration Information .......... 18
       4.1.2. Explicit Source FEC Payload ID ................ 19
       4.1.3. Repair FEC Payload ID .......................... 20
       4.1.4. Additional Procedures .......................... 21
5. Sliding Window RLC FEC Scheme over GF(2) for Arbitrary Packet
   Flows .................................................... 21
   5.1. Formats and Codes ................................... 21
       5.1.1. FEC Framework Configuration Information .......... 22
       5.1.2. Explicit Source FEC Payload ID ................ 22
       5.1.3. Repair FEC Payload ID .......................... 22
       5.1.4. Additional Procedures .......................... 22
6. FEC Code Specification .................................... 22
   6.1. Encoding Side ....................................... 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 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 a end-host/server (source) or a middlebox) and a single FEC decoding point (either a 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, 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 front 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 follow a totally different 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 3.5). 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 7). Similarly, each FEC
Source Packet features a fixed 32-bit long trailer, called Explicit Source FEC Payload ID (Figure 5), 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 signalling information associated to each source or repair symbol. It also defines the FEC Framework Configuration Information (FFCI) carrying signalling 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 (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)
PRNG: pseudo-random number generator
tinymt32_rand(maxv): PRNG defined in Section 3.4 and used in this specification, that returns a new random integer in \([0; maxv-1]\)
DT: coding coefficients density threshold, an integer between 0 and 15 (inclusive) the controls the fraction of coefficients that are non zero

3. Procedures

This section introduces the procedures that are used by these FEC Schemes.

3.1. Possible Parameter Derivations

The Sliding Window RLC FEC Scheme relies on several parameters:

Maximum FEC-related latency budget, max_lat (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 B 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 as explained below;

Encoding window current (resp. maximum) size, ew_size (resp. ew_max_size) (in symbols):

at a FECFRAME sender, during FEC encoding, a repair symbol is computed as a linear combination of the ew_size source symbols present in the encoding window. The ew_max_size is the maximum size of this window, while ew_size is the current size. For instance, at session start, upon receiving new source ADUs, the ew_size progressively increases until it reaches its maximum value, ew_max_size. We have:
Decoding window maximum size, \textit{dw\_max\_size} (in symbols): at a FECFRAME receiver, \textit{dw\_max\_size} is the maximum number of received or lost source symbols that are still within their latency budget;

Linear system maximum size, \textit{ls\_max\_size} (in symbols): at a FECFRAME receiver, the linear system maximum size, \textit{ls\_max\_size}, is the maximum number of received or lost source symbols in the linear system (i.e., the variables). It SHOULD NOT be smaller than \textit{dw\_max\_size} since it would mean that, even after receiving a sufficient number of FEC Repair Packets, a lost ADU may not be recovered just because the associated source symbols have been prematurely removed from the linear system, which is usually counter-productive. On the opposite, the linear system MAY grow beyond the \textit{dw\_max\_size} (Appendix B);

Symbol size, \textit{E} (in bytes): the \textit{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 SHOULD fix the \textit{E} parameter and communicate it as part of the FEC Scheme-Specific Information (Section 4.1.1.2).

Code rate, \textit{cr}: The code rate parameter determines the amount of redundancy added to the flow. More precisely the \textit{cr} is the ratio between the total number of source symbols and the total number of source plus repair symbols and by definition: \(0 < \text{cr} \leq 1\). This is an input parameter that enables a FECFRAME sender to derive other internal parameters, as explained below. However there is no need to communicate the \textit{cr} parameter per see (it’s not required to process a repair symbol at a receiver). This code rate parameter can be dynamic. 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.

The FEC Schemes can be used in various manners. They can be used to protect a source ADU flow having real-time constraints, or a non-realtime source ADU flow. The source ADU flow may be a Constant Bitrate (CBR) or Variable BitRate (VBR) flow. The flow’s minimum/maximum bitrate might or might not be known. The FEC Schemes can also be used over the Internet or over a CBR communication path. It follows that the FEC Scheme parameters can be derived in different ways, as described in the following sections.

3.1.1. Case of a CBR Real-Time Flow

In the following, we consider a real-time flow with \textit{max\_lat} latency budget. The encoding symbol size, \textit{E}, is constant. The code rate, \textit{cr}, is also constant, its value depending on the expected
communication loss model (this choice is out of scope of this document).

In a first configuration, the source ADU flow bitrate at the input of the FECFRAME sender is fixed and equal to \( br_{in} \) (in bits/s), and this value is known by the FECFRAME sender. It follows that the transmission bitrate at the output of the FECFRAME sender will be higher, depending on the added repair flow overhead. In order to comply with the maximum FEC-related latency budget, we have:

\[
dw_{\text{max size}} = \frac{(\text{max lat} \times br_{in})}{(8 \times E)}
\]

In a second configuration, the FECFRAME sender generates a fixed bitrate flow, equal to the CBR communication path bitrate equal to \( br_{out} \) (in bits/s), and this value is known by the FECFRAME sender, as in [Roca17]. The maximum source flow bitrate needs to be such that, with the added repair flow overhead, the total transmission bitrate remains inferior or equal to \( br_{out} \). We have:

\[
dw_{\text{max size}} = \frac{(\text{max lat} \times br_{out} \times cr)}{(8 \times E)}
\]

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 exact value having no impact on the the FEC-related latency budget. For the FEC Schemes specified in this document, in line with [Roca17], the \( ew_{\text{max size}} \) SHOULD be computed with:

\[
w_{\text{max size}} = dw_{\text{max size}} \times 0.75
\]

The \( ew_{\text{max size}} \) is the main parameter at a FECFRAME sender. It is RECOMMENDED to check that the \( ew_{\text{max size}} \) value stays within reasonable bounds in order to avoid hazardous behaviours.

The \( dw_{\text{max size}} \) is computed by a FECFRAME sender but not explicitly communicated to a FECFRAME receiver. However a FECFRAME receiver can easily evaluate the \( ew_{\text{max size}} \) by observing the maximum Number of Source Symbols (NSS) value contained in the Repair FEC Payload ID of received FEC Repair Packets (Section 4.1.3). A receiver can then easily compute \( dw_{\text{max size}} \):

\[
dw_{\text{max size}} = \frac{\text{max NSS observed}}{0.75}
\]

A receiver can then chose an appropriate linear system maximum size:

\[
l_{\text{max size}} \geq dw_{\text{max size}}
\]
It is good practice to use a larger value for ls_max_size as explained in Appendix B, which does not impact maximum latency nor interoperability. However, the linear system size should not be too large for practical reasons (e.g., in order to limit computation complexity). It is RECOMMENDED to check that the ls_max_size value stays within reasonable bounds in order to avoid hazardous behaviours.

The particular case of session start needs to be managed appropriately. Here ew_size increases each time a new source ADU is received by the FECFRAME sender, until it reaches the ew_max_size value. 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 needed.

3.1.2. Other Types of Real-Time Flow

In other configurations, a real-time source ADU flow, with a max_lat latency budget, features a variable bitrate (VBR). 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 Section 3.1.1. Considering the smallest bitrate means that the encoding window and decoding window maximum sizes estimation are pessimistic; these windows have the smallest size required to enable a decoding on-time at a FECFRAME receiver. If the instantaneous bitrate is higher than this smallest bitrate, this approach leads to an encoding window that is unnecessarily small, which reduces robustness in front of long erasure bursts.

Another approach consists in using ADU timing information (e.g., using the timestamp field of an RTP packet header, or registering the time upon receiving a new ADU). From the global FEC-related latency budget the FECFRAME sender can derive a practical maximum latency budget for encoding operations, max_lat_for_encoding. For the FEC Schemes specified in this document, this latency budget SHOULD be computed with:

\[
\text{max\_lat\_for\_encoding} = \text{max\_lat} \times 0.75
\]

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 (Section 3.1.3).
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 Section 3.1.1.

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. However it is usually preferable to use a ls_max_size "too large" (which can increase computation complexity and memory requirements) than the opposite (which can reduce recovery performance).

Beyond these general guidelines, the details of how to manage these situations at a FECFRAME sender and receiver can depend on additional considerations that are out of scope of this document.

3.1.3. 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.

Beyond these general guidelines, the details of how to manage these situations at a FECFRAME sender and receiver can depend on additional considerations that are out of scope of this document.

3.2. ADU, ADUI and Source Symbols Mappings

At a sender, an ADU coming from the application cannot directly be 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). At a sender, this identifier is prepended to each ADU before FEC encoding. This way, FEC decoding at a receiver also recovers this Flow ID and a recovered
ADU can be assigned to the right source flow (note that transport port numbers and IP addresses cannot be used to that purpose as they are not recovered during FEC decoding).

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.

```
+---------+---------------------------------------------+-------------+
|F| L| ADU                               |     Pad     |
+---------+---------------------------------------------+-------------+
< ------------------ >< ------------------ >< ------------------ >
```

Figure 1: ADUI Creation example (here 3 source symbols are created for this ADUI).

Note that neither the initial 3 bytes nor the optional padding are sent over the network. However, they are considered during FEC encoding, and a receiver who lost a certain FEC Source Packet (e.g., the UDP datagram containing this FEC Source Packet when UDP is used...
as the transport protocol) will be able to recover the ADUI if FEC decoding succeeds. Thanks to the initial 3 bytes, this receiver will get rid of the padding (if any) and identify the corresponding ADU flow.

3.3. Encoding Window Management

Source symbols and the corresponding ADUs are removed from the encoding window:

- 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. Pseudo-Random Number Generator (PRNG)

The RLC FEC Schemes defined in this document rely on the TinyMT32 PRNG, a small-sized variant of the Mersenne Twister PRNG, as defined in the reference implementation version 1.1 (2015/04/24) by Mutsuo Saito (Hiroshima University) and Makoto Matsumoto (The University of Tokyo).

- Official github site and reference implementation: [https://github.com/MersenneTwister-Lab/TinyMT](https://github.com/MersenneTwister-Lab/TinyMT)

For the RLC FEC Schemes defined in this document, the tinymt32 32-bit version (rather than the 64-bit version) MUST be used. This PRNG
requires a parameter set that needs to be pre-calculated. For the
RLC FEC Schemes defined in this document, the following parameter set
MUST be used:

- mat1 = 0x8f7011ee = 2406486510;
- mat2 = 0xfc78ff1f = 4235788063;
- tmat = 0x3793fdff = 932445695.

This parameter set is the first entry of the precalculated parameter
sets in file tinymt32dc.0.1048576.txt, by Kenji Rikitake, and
available at:

- <https://github.com/jj1bdx/tinymt32dc-
  longbatch/blob/master/tinymt32dc/tinymt32dc.0.1048576.txt>.

This is also the parameter set used in [KR12].

The PRNG reference implementation is distributed under a BSD license
and excerpts of it are reproduced in Appendix A. In order to
validate an implementation of this PRNG, using seed 1, the 10,000th
value returned by: tinymt32_rand(s, 0xffff) MUST be equal to 0x7c37.

This PRNG MUST first be initialized with a 32-bit unsigned integer,
used as a seed. The following function is used to this purpose:

```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 equal to 0), since this is the
Repair_Key 16-bit field value of the Repair FEC Payload ID
(Section 4.1.3). 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 maxv-1
inclusive is needed, the following function is used:

```c
uint32_t tinymt32_rand (tinymt32_t * s, uint32_t maxv);
```

This function takes as parameter both a pointer to the same
tinymt32_t structure (that needs to be left unchanged between
successive calls to the function) and the maxv value.
3.5. 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. When DT equals 15, the maximum value, the function guarantees that all coefficients are non zero (i.e., maximum density). When DT is between 0 (minimum value) and strictly inferior to 15, the average probability of having a non zero coefficient equals \((\text{DT} + 1) / 16\).

These considerations apply both the RLC over GF(2) and RLC over GF(2\(^8\)), the only difference being the value of the \(m\) parameter. With the RLC over GF(2) FEC Scheme (Section 5), \(m\) MUST be equal to 1. With RLC over GF(2\(^8\)) FEC Scheme (Section 4), \(m\) MUST be equal to 8.

```c
/*
 * Fills in the table of coding coefficients (of the right size)
 * provided with the appropriate number of coding coefficients to
 * use for the repair symbol key provided.
 *
 * (in) repair_key    key associated to this repair symbol. This
 *                    parameter is ignored (useless) if m=2 and dt=15
 * (in) 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 table. This value is
 *                    equal to the current encoding window size.
 * (in) dt            integer between 0 and 15 (inclusive) that
 *                    controls the density. With value 15, all
 *                    coefficients are guaranteed to be non zero
 *                    (i.e. equal to 1 with GF(2) and equal to a
 *                    value in \{1,... 255\} with GF(2\(^8\)), otherwise
 *                    a fraction of them will be 0.
 * (in) m             Finite Field GF(2\(^m\)) parameter. In this
 *                    document only values 1 and 8 are considered.
 * (out)              returns an error code
 */
int generate_coding_coefficients (uint16_t  repair_key,
                                 uint8_t   cc_tab[],
                                 uint16_t  cc_nb,
                                 uint16_t  dt,
                                 uint8_t   m)
{
    uint32_t  i;
```
tinymt32_t s; /* PRNG internal state */

if (dt > 15) {
    return SOMETHING_WENT_WRONG; /* bad dt parameter */
} else {
    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++) {
                if (tinymt32_rand(&s, 16) <= dt) {
                    cc_tab[i] = (uint8_t) 1;
                } else {
                    cc_tab[i] = (uint8_t) 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_rand(&s, 256);
                } while (cc_tab[i] == 0);
            }
        } else {
            /* here a certain fraction of coefficients should be 0 */
            for (i = 0; i < cc_nb; i++) {
                if (tinymt32_rand(&s, 16) <= dt) {
                    do {
                        cc_tab[i] = (uint8_t) tinymt32_rand(&s, 256);
                    } while (cc_tab[i] == 0);
                } else {
                    cc_tab[i] = 0;
                }
            }
        }
        break;
    default:
    }
}
3.6. Finite Fields Operations

3.6.1. 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 MUST be used for GF(2^8):

\[ x^8 + x^4 + x^3 + x^2 + 1 \]

With GF(2), multiplication corresponds to a logical AND operation.

3.6.2. 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:

\[ \text{repair}[i] = \text{cc_tab}[0] \times \text{src}_0[i] + \text{cc_tab}[1] \times \text{src}_1[i] + \ldots + \text{cc_tab}[\text{ew_size} - 1] \times \text{src}_\text{ew_size}_1[i] \]

where \( \times \) is the multiplication over GF(2^8) and + is an XOR operation. 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;
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

If another mechanism requires the FSSI to be carried as an opaque octet string (for instance, after a Base64 encoding), the encoding format consists of the following 2 octets:

Encoding symbol length (E): 16-bit field.

```
0 1
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-----------------------------------
|   Encoding Symbol Length (E)   |
+-----------------------------------
```

Figure 3: 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 4.

```
+-----------------------------------+
|           IP Header              |
+-----------------------------------+
|   Transport Header               |
+-----------------------------------+
|              ADU                  |
|   Explicit Source FEC Payload ID |
+-----------------------------------+
```

Figure 4: 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 (Figure 5):

Encoding Symbol ID (ESI) (32-bit field): this unsigned integer identifies the first source symbol of the ADUI corresponding to this FEC Source Packet. The ESI is incremented for each new
source symbol, and after reaching the maximum value (2^32-1), wrapping to zero occurs.

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Encoding Symbol ID (ESI)                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Figure 5: 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 6.

+--------------------------------+  
|           IP Header            |  
+--------------------------------+  
|        Transport Header        |  
+--------------------------------+  
|     Repair FEC Payload ID      |  
+--------------------------------+  
|         Repair Symbol          |  
+--------------------------------+  

Figure 6: 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 (Figure 7):

Repair_Key (16-bit field): this unsigned integer is used as a seed by the coefficient generation function (Section 3.5) in order to generate the desired number of coding coefficients. 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 3.5. More
precisely, it controls the probability of having a non zero coding
coefficient, which equals (DT+1) / 16. When a FEC Repair Packet
contains several repair symbols, the DT value applies to all of
them;
Number of Source Symbols in the encoding window, NSS (12-bit field):
this unsigned integer indicates the number of source symbols in
the encoding window when this repair symbol was generated. When a
FEC Repair Packet contains several repair symbols, this NSS value
applies to all of them;
ESI of First Source Symbol in the encoding window, FSS_ESI (32-bit
field):
this unsigned integer indicates the ESI of the first source symbol
in the encoding window when this repair symbol was generated.
When a FEC Repair Packet contains several repair symbols, this
FSS_ESI value applies to all of them;

0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Repair_Key              |  DT   |NSS (# src symb in ew) |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            FSS_ESI                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

Figure 7: Repair FEC Payload ID Encoding Format

4.1.4. Additional Procedures

The following procedure applies:

- 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 32-bit value.

5. Sliding Window RLC FEC Scheme over GF(2) for Arbitrary Packet Flows

This fully-specified FEC Scheme defines the Sliding Window Random
Linear Codes (RLC) over GF(2) (binary case).

5.1. Formats and Codes
5.1.1. FEC Framework Configuration Information

5.1.1.1. FEC Encoding ID

o FEC Encoding ID: the value assigned to this fully specified FEC Scheme MUST be YYYY, as assigned by IANA (Section 10).

When SDP is used to communicate the FFCI, this FEC Encoding ID is carried in the 'encoding-id' parameter.

5.1.1.2. FEC Scheme-Specific Information

All the considerations of Section 4.1.1.2 apply here.

5.1.2. Explicit Source FEC Payload ID

All the considerations of Section 4.1.1.2 apply here.

5.1.3. Repair FEC Payload ID

All the considerations of Section 4.1.1.2 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 it is RECOMMENDED that the sender use value 0 for the Repair_Key field, but a receiver SHALL ignore this field.

5.1.4. Additional Procedures

All the considerations of Section 4.1.1.2 apply here.

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 3.5) 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.
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 $ew_{\text{size}}$ 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.

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 $ew_{\text{size}}$ coding coefficients that are computed by the same coefficient generation function (Section 3.5), 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. 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 B 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 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);

- **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;

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 Section 3.1.1. 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.

Section 3.1.2 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 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]). 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 compute the XOR sum of all the source symbols in the encoding window. In that case: (1) a single repair symbol can be produced for any encoding window, and (2) the repair_key parameter becomes useless (the coding coefficients generation function does not rely on the PRNG).

10. IANA Considerations

This document registers two values in the "FEC Framework (FECFRAME) FEC Encoding IDs" registry [RFC6363] as follows:

- YYYY refers to the Sliding Window Random Linear Codes (RLC) over GF(2) FEC Scheme for Arbitrary Packet Flows, as defined in Section 5 of this document.
- XXXX refers to the Sliding Window Random Linear Codes (RLC) over GF(2^{^8}) FEC Scheme for Arbitrary Packet Flows, as defined in Section 4 of this document.

11. Acknowledgments

The authors would like to thank Russ Housley, Alan DeKok, Spencer Dawkins, Gorry Fairhurst, Jonathan Detchart, Emmanuel Lochin, and Marie-Jose Montpetit for their valuable feedbacks on this document.

12. References

12.1. Normative References
12.2. Informative References


[Roca16] Roca, V., Teibi, B., Burdinat, C., Tran, T., and C. Thienot, "Block or Convolutional AL-FEC Codes? A Performance Comparison for Robust Low-Latency Communications", HAL open-archive document, hal-01395937 https://hal.inria.fr/hal-01395937/en/, November 2016, <https://hal.inria.fr/hal-01395937/en/>.

Appendix A. TinyMT32 Pseudo-Random Number Generator

The TinyMT32 PRNG reference implementation is distributed under a BSD license by the authors and excerpts of it are reproduced in Figure 8. The differences with respect to the original source code are:

- the unused parts of the original source code have been removed;
- the appropriate parameter set has been added to the initialization function;
- the tinymt32_rand() function has been added;
- the function order has been changed;
- certain internal variables have been renamed for compactness purposes.

<CODE BEGINS>
/**
 * Tiny Mersenne Twister only 127 bit internal state
 *
 * Authors : Mutsuo Saito (Hiroshima University)
 *           Makoto Matsumoto (University of Tokyo)
 *
 * Copyright (c) 2011, 2013 Mutsuo Saito, Makoto Matsumoto, Hiroshima University and The University of Tokyo.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
 * - Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
 * - Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
 * - Neither the name of the Hiroshima University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
* ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
* TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF
* THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*/

#include <stdint.h>

/*
  tinymt32 internal state vector and parameters
*/
typedef struct {
    uint32_t status[4];
    uint32_t mat1;
    uint32_t mat2;
    uint32_t tmat;
} tinymt32_t;

static void tinymt32_next_state (tinymt32_t * s);
static uint32_t tinymt32_temper (tinymt32_t * s);
static double tinymt32_generate_32double (tinymt32_t * s);

/**
 * Parameter set to use for the IETF RLC FEC Schemes specification.
 * Do not change.
 * This parameter set is the first entry of the precalculated parameter
 * sets in file tinymt32dc.0.1048576.txt, by Kenji Rikitake, available
 * at: https://github.com/jj1bdx/tinymtdc-longbatch/blob/master/
 * tinymt32dc/tinymt32dc.0.1048576.txt
 * It is also the parameter set used:
 * Rikitake, K., "TinyMT Pseudo Random Number Generator for
 * Erlang", ACM 11th SIGPLAN Erlang Workshop (Erlang’12),
 * September, 2012.
 */
#define TINYMT32_MAT1_PARAM     0x8f7011ee
#define TINYMT32_MAT2_PARAM     0xfc78ff1f
#define TINYMT32_TMAT_PARAM     0x3793fdff

/**
 * This function initializes the internal state array with a 32-bit
 * unsigned integer seed.
 * @param s    pointer to tinymt internal state.
 * @param seed  a 32-bit unsigned integer used as a seed.
 */
void tinymt32_init (tinymt32_t * s, uint32_t seed)
{
#define MIN_LOOP 8
#define PRE_LOOP 8
s->status[0] = seed;
s->status[1] = s->mat1 = TINYMT32_MAT1_PARAM;
s->status[2] = s->mat2 = TINYMT32_MAT2_PARAM;
for (int i = 1; i < MIN_LOOP; i++) {
    s->status[i & 3] ^= i + UINT32_C(1812433253)
        * (s->status[(i - 1) & 3]
            ^ (s->status[(i - 1) & 3] >> 30));
}
for (int i = 0; i < PRE_LOOP; i++) {
tinymt32_next_state(s);
}

/**
 * This function outputs an integer in the [0 .. maxv-1] range.
 * @param s     pointer to tinymt internal state.
 * @return      32-bit unsigned integer between 0 and maxv-1 inclusive.
 */
uint32_t tinymt32_rand (tinymt32_t * s, uint32_t maxv)
{
    return (uint32_t)(tinymt32_generate_32double(s) * (double)maxv);
}

/**
 * Internal tinymt32 constants and functions.
 * Users should not call these functions directly.
 */
#define TINYMT32_MEXP 127
#define TINYMT32_SH0 1
#define TINYMT32_SH1 10
#define TINYMT32_SH8 8
#define TINYMT32_MASK UINT32_C(0xffffffff)
#define TINYMT32_MUL (1.0f / 16777216.0f)

/**
 * This function changes internal state of tinymt32.
 * @param s     pointer to tinymt internal state.
 */
static void tinymt32_next_state (tinymt32_t * s)
{
    uint32_t x;
    uint32_t y;
    y = s->status[3];
x = (s->status[0] & TINYMT32_MASK)
    ^ s->status[1]
    ^ s->status[2];
x ^= (x << TINYMT32_SH0);
y ^= (y >> TINYMT32_SH0) ^ x;
s->status[0] = s->status[1];
s->status[1] = s->status[2];
s->status[2] = x ^ (y << TINYMT32_SH1);
s->status[3] = y;
s->status[1] ^= -((int32_t)(y & 1)) & s->mat1;
s->status[2] ^= -((int32_t)(y & 1)) & s->mat2;
}

/**
 * This function outputs 32-bit unsigned integer from internal state.
 * @param s     pointer to tinymt internal state.
 * @return      32-bit unsigned pseudos number
 */
static uint32_t tinymt32_temper (tinymt32_t * s)
{
    uint32_t t0, t1;
    t0 = s->status[3];
    t1 = s->status[0] + (s->status[2] >> TINYMT32_SH8);
    t0 ^= t1;
    t0 ^= -((int32_t)(t1 & 1)) & s->tmat;
    return t0;
}

/**
 * This function outputs double precision floating point number from
 * internal state. The returned value has 32-bit precision.
 * In other words, this function makes one double precision floating
 * point number from one 32-bit unsigned integer.
 * @param s     pointer to tinymt internal state.
 * @return      floating point number r (0.0 <= r < 1.0)
 */
static double tinymt32_generate_32double (tinymt32_t * s)
{
    tinymt32_next_state(s);
    return (double)tinymt32_temper(s) * (1.0 / 4294967296.0);
}

Figure 8: TinyMT32 pseudo-code
Appendix B. Decoding Beyond Maximum Latency Optimization

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:

\[
\text{ls\_max\_size} > \text{dw\_max\_size}
\]

Usually the following choice is a good trade-off between decoding performance and extra CPU overhead:

\[
\text{ls\_max\_size} = 2 \times \text{dw\_max\_size}
\]

When the \text{dw\_max\_size} is very small, it may be preferable to keep a minimum \text{ls\_max\_size} value (e.g., \text{LS\_MIN\_SIZE\_DEFAULT} = 40 symbols). Going below this threshold will not save a significant amount of memory nor CPU cycles. Therefore:

\[
\text{ls\_max\_size} = \max(2 \times \text{dw\_max\_size}, \text{LS\_MIN\_SIZE\_DEFAULT})
\]

Finally, it is worth noting that a good receiver, i.e., a receiver that benefits from an FEC protection significantly higher than what is required to recover from packet losses, can choose to reduce the \text{ls\_max\_size}. In that case lost ADUs will be recovered without relying on this optimization.

\[
\text{ls\_max\_size}
\]

\[
\text{late\ source\ symbols} \quad \text{(pot. decoded but not delivered)} \quad \text{dw\_max\_size}
\]

\[
\text{src0} \quad \text{src1} \quad \text{src2} \quad \text{src3} \quad \text{src4} \quad \text{src5} \quad \text{src6} \quad \text{src7} \quad \text{src8} \quad \text{src9} \quad \text{src10} \quad \text{src11} \quad \text{src12}
\]

Figure 9: 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 9). 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

Belkacem Teibi
INRIA
Univ. Grenoble Alpes
France

EMail: belkacem.teibi@inria.fr
Tunnel Congestion Feedback
draft-ietf-tsvwg-tunnel-congestion-feedback-06

Abstract

This document describes a method to measure congestion on a tunnel segment based on recommendations from RFC 6040, "Tunneling of Explicit Congestion Notification", and to use IPFIX to communicate the congestion measurements from the tunnel’s egress to a controller which can respond by modifying the traffic control policies at the tunnel’s ingress.

Status of this Memo

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

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

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

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

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

Copyright and License Notice
Copyright (c) 2017 IETF Trust and the persons identified as the
document authors. All rights reserved.

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

Table of Contents

1. Introduction .................................................. 3
2. Conventions And Terminologies ................................. 3
3. Congestion Information Feedback Models ...................... 4
4. Congestion Level Measurement ................................ 5
5. Congestion Information Delivery ............................... 7
5.1 IPFIX Extensions ............................................. 8
5.1.1 tunnelEcnCeCeByteTotalCount ......................... 8
5.1.2 tunnelEcnEct0NectByteTotalCount ...................... 8
5.1.3 tunnelEcnEct1NectByteTotalCount ...................... 9
5.1.4 tunnelEcnCeNectByteTotalCount ......................... 9
5.1.5 tunnelEcnCeEct0ByteTotalCount ......................... 9
5.1.6 tunnelEcnCeEct1ByteTotalCount ......................... 10
5.1.7 tunnelEcnEct0Ect0ByteTotalCount ..................... 10
5.1.8 tunnelEcnEct1Ect1PacketTotalCount ................... 10
5.1.9 tunnelEcnCEMarkedRatio ................................. 11
6. Congestion Management ......................................... 11
6.1 Example ...................................................... 11
7. Security Considerations ....................................... 14
8. IANA Considerations ........................................... 15
9. References ..................................................... 17
9.1 Normative References ....................................... 17
9.2 Informative References ..................................... 18
10. Acknowledgements ............................................. 18
Authors’ Addresses .............................................. 18
1. Introduction

In IP networks, persistent congestion [RFC2914] lowers transport throughput, leading to waste of network resource. Appropriate congestion control mechanisms are therefore critical to prevent the network from falling into the persistent congestion state. Currently, transport protocols such as TCP [RFC793], SCTP [RFC4960], DCCP [RFC4340], have their built-in congestion control mechanisms, and even for certain single transport protocol like TCP there can be a couple of different congestion control mechanisms to choose from. All these congestion control mechanisms are implemented on host side, and there are reasons that only host side congestion control is not sufficient for the whole network to keep away from persistent congestion. For example, (1) some protocol’s congestion control scheme may have internal design flaws; (2) improper software implementation of protocol; (3) some transport protocols, e.g. RTP [RFC3550] do not even provide congestion control at all; (4) a heavy load from a much larger than expected number of responsive flows could also lead to persistent congestion.

Tunnels are widely deployed in various networks including public Internet, data center network, and enterprise network etc. A tunnel consists of ingress, egress and a set of intermediate routers. For the tunnel scenario, a tunnel-based mechanism is introduced for network traffic control to keep the network from persistent congestion. Here, tunnel ingress will implement congestion management function to control the traffic entering the tunnel.

This document provides a mechanism of feeding back inner tunnel congestion level to the ingress. Using this mechanism the egress can feed the tunnel congestion level information it collects back to the ingress. After receiving this information the ingress will be able to perform congestion management according to network management policy.

The following subjects are out of scope of current document: it gives no advice on how to select which tunnel endpoints should be used in order to manage traffic over a network criss-crossed by multiple tunnels; if a congested node is part of multiple tunnels, and it causes congestion feedback to multiple traffic management functions at the inresses of all the tunnels, the draft gives no advice on how all the traffic management functions should respond.

2. Conventions And Terminologies

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]
DP: Decision Point, an logical entity that makes congestion management decision based on the received congestion feedback information.

EP: Enforcement Point, an logical entity that implements congestion management action according to the decision made by Decision Point.

ECT: ECN-Capable Transport code point defined in RFC3168.

3. Congestion Information Feedback Models

The feedback model mainly consists of tunnel egress and tunnel ingress. The tunnel egress composes of meter function and exporter function; tunnel ingress composes EP (Enforcement Point) function, collector function and DP (Decision Point) function.

The Meter function collects network congestion level information, and conveys the information to Exporter which feeds back the information to the collector function.

The feedback message contains CE-marked packet ratio, the traffic volumes of all kinds of ECN marking packets.

The collector collects congestion level information from exporter, after that congestion management Decision Point (DP) function will make congestion management decision based on the information from collector.

The Enforcement Point controls the traffic entering tunnel, and it implements traffic control decision of DP.
4. Congestion Level Measurement

The congestion level measurement is based on ECN (Explicit Congestion Notification) [RFC3168] and packet drop. The network congestion level could be indicated through the ratio of CE-marked packet and the volumes of packet drop, the relationship between these two kinds of indicator is complementary. If the congestion level in tunnel is not high enough, the packets would be marked as CE instead of being dropped, and then it is easy to calculate congestion level according to the ratio of CE-marked packets. If the congestion level is so high that ECT packet will be dropped, then the packet loss ratio could be calculated by comparing total packets entering ingress and total packets arriving at egress over the same span of packets, if packet loss is detected, it could be assumed that severe congestion has occurred in the tunnel.

Egress calculates CE-marked packet ratio by counting different kinds of ECN-marked packet, the CE-marked packet ratio will be used as an indication of tunnel load level. It’s assumed that routers in the tunnel will not drop packets biased towards certain ECN codepoint, so calculating of CE-marked packet ratio is not affect by packet drop.

The calculation of volumes of packet drop is by comparing the traffic volumes between ingress and egress.

Faked ECN-capable transport (ECT) is used at ingress to defer...
packet loss to egress. The basic idea of faked ECT is that, when
encapsulating packets, ingress first marks tunnel outer header
according to RFC6040, and then remarks outer header of Not-ECT
packet as ECT, there will be three kinds of combination of outer
header ECN field and inner header ECN field: CE|CE, ECT|N-ECT,
ECT|ECT (in the form of outer ECN| inner ECN); when decapsulating
packets at egress, RFC6040 defined decapsulation behavior is used,
and according to RFC6040, the packets marked as CE|N-ECT will be
dropped by egress. Faked-ECT is used to shift some drops to the
egress in order to calculate CE-marked packet ratio more precisely
by egress.

To calculate congestion level, for the same span of packets, the
ratio of CE-marked packets will be calculated by egress, and the
total bytes count of packets at ingress and egress will be
compared to detect the traffic volume loss in tunnel.

The basic procedure of packets loss measurement is as follows:

```
+-------+                 +------+
|Ingress|                 |Egress|
+-------+                 +------+
   +----------------+                |
   |cumulative count|                |
   +----------------+                |
       <node id-i, ECN counts>             |
           ---------------------------------
       <node id-e, ECN counts>             |
                                     <------------------------|
```

Figure 2: Procedure of Packet Loss Measurement

Ingress encapsulates packets and marks outer header according to
faked ECT as described above. Ingress cumulatively counts packet
bytes for three types of ECN combination (CE|CE, ECT|N-ECT, ECT|ECT)
and then the ingress regularly sends cumulative bytes counts message
of each type of ECN combination to the egress.

When each message arrives at egress, (1) egress calculates the ratio
of CE-marked packet; (2) the egress cumulatively counts packet bytes
coming from the ingress and adds its own bytes counts of each type of
ECN combination (CE|CE, ECT|N-ECT, CE|N-ECT, CE|ECT, ECT|ECT) to the
message for ingress to calculate packet loss. Egress feeds back CE-marked packet ratio and bytes counts information to the ingress for evaluating congestion level in the tunnel.

The counting of bytes can be at the granularity of the all traffic from the ingress to the egress to learn about the overall congestion status of the path between the ingress and the egress. The counting can also be at the granularity of individual customer’s traffic or a specific set of flows to learn about their congestion contribution.

5. Congestion Information Delivery

As described above, the tunnel ingress needs to convey a message containing cumulative bytes counts of packets of each type of ECN combination to tunnel egress, and the tunnel egress also needs to feed back the message of cumulative bytes counts of packets of each type of ECN combination and CE-marked packet ratio to the ingress. This section describes how the messages should be conveyed.

The message travels along the same path with network data traffic, referred as in-band signal. Because the message is transmitted in band, so the message packet may get lost in case of network congestion. To cope with the situation that the message packet gets lost, the bytes counts values are sent as cumulative counters. Then if a message is lost the next message will recover the missing information. Even though the missing information could be recovered, the message should be transmitted in a much higher priority than users’ traffic flows.

IPFIX [RFC7011] is selected as a candidate information feedback protocol. IPFIX uses preferably SCTP as transport. SCTP allows partially reliable delivery [RFC3758], which ensures the feedback message will not be blocked in case of packet loss due to network congestion.

Ingress can do congestion management at different granularity which means both the overall aggregated inner tunnel congestion level and congestion level contributed by certain traffic(s) could be measured for different congestion management purpose. For example, if the ingress only wants to limit congestion volume caused by certain traffic(s), e.g. UDP-based traffic, then congestion volume for the traffic will be fed back; or if the ingress do overall congestion management, the aggregated congestion volume will be fed back.

When sending message from ingress to egress, the ingress acts as IPFIX exporter and egress acts as IPFIX collector; When feedback congestion level information from egress to ingress, then the egress acts as IPFIX exporter and ingress acts as IPFIX collector.
The combination of congestion level measurement and congestion information delivery procedure should be as following:

# The ingress determines IPFIX template record to be used. The template record can be pre-configured or determined at runtime, the content of template record will be determined according to the granularity of congestion management, if the ingress wants to limit congestion volume contributed by specific traffic flow then the elements such as source IP address, destination IP address, flow id and CE-marked packet volume of the flow etc will be included in the template record.

# Meter on ingress measures traffic volume according to template record chosen and then the measurement records are sent to egress in band.

# Meter on egress measures congestion level information according to template record, the content of template record should be the same as template record of ingress.

# Exporter of egress sends measurement record together with the measurement record of ingress back to the ingress.

5.1 IPFIX Extensions

This sub-section defines a list of new IPFIX Information Elements according to RFC7013 [RFC7013].

5.1.1 tunnelEcnCeCeByteTotalCount

Description: The total number of bytes of incoming packets with CE|CE ECN marking combination at the Observation Point since the Metering Process (re-)initialization for this Observation Point.

Abstract Data Type: unsigned64

Data Type Semantics: totalCounter

ElementId: TBD1

Statues: current

Units: bytes

5.1.2 tunnelEcnEct0NectByteTotalCount

Description: The total number of bytes of incoming packets with ECT(0)|N-ECT ECN marking combination at the Observation Point since
the Metering Process (re-)initialization for this Observation Point.

Abstract Data Type: unsigned64
Data Type Semantics: totalCounter
ElementId: TBD2
Statues: current
Units: bytes

5.1.3 tunnelEcnEct1NectByteTotalCount

Description: The total number of bytes of incoming packets with ECT(1)|N-ECT ECN marking combination at the Observation Point since the Metering Process (re-)initialization for this Observation Point.

Abstract Data Type: unsigned64
Data Type Semantics: totalCounter
ElementId: TBD3
Statues: current
Units: bytes

5.1.4 tunnelEcnCeNectByteTotalCount

Description: The total number of bytes of incoming packets with CE|N-ECT ECN marking combination at the Observation Point since the Metering Process (re-)initialization for this Observation Point.

Abstract Data Type: unsigned64
Data Type Semantics: totalCounter
ElementId: TBD4
Statues: current
Units: bytes

5.1.5 tunnelEcnCeEct0ByteTotalCount

Description: The total number of bytes of incoming packets with CE|ECT(0) ECN marking combination at the Observation Point since the
Metering Process (re-)initialization for this Observation Point.

Abstract Data Type: unsigned64
Data Type Semantics: totalCounter
ElementId: TBD5
Statues: current
Units: bytes

5.1.6 tunnelEcnCeEct1ByteTotalCount

Description: The total number of bytes of incoming packets with CE|ECT(1) ECN marking combination at the Observation Point since the Metering Process (re-)initialization for this Observation Point.

Abstract Data Type: unsigned64
Data Type Semantics: totalCounter
ElementId: TBD6
Statues: current
Units: bytes

5.1.7 tunnelEcnEct0Ect0ByteTotalCount

Description: The total number of bytes of incoming packets with ECT(0)|ECT(0) ECN marking combination at the Observation Point since the Metering Process (re-)initialization for this Observation Point.

Abstract Data Type: unsigned64
Data Type Semantics: totalCounter
ElementId: TBD7
Statues: current
Units: bytes

5.1.8 tunnelEcnEct1Ect1PacketTotalCount

Description: The total number of bytes of incoming packets with ECT(1)|ECT(1) ECN marking combination at the Observation Point since
the Metering Process (re-)initialization for this Observation Point.

Abstract Data Type: unsigned64
Data Type Semantics: totalCounter
ElementId: TBD8
Statues: current
Units: bytes

5.1.9 tunnelEcnCEMarkedRatio

Description: The ratio of CE-marked Packet at the Observation Point.
Abstract Data Type: float32
ElementId: TBD8
Statues: current

6. Congestion Management

After tunnel ingress receives congestion level information, then congestion management actions could be taken based on the information, e.g. if the congestion level is higher than a predefined threshold, then action could be taken to reduce the congestion level.

The design of network side congestion management SHOULD take host side e2e congestion control mechanism into consideration, which means the congestion management needs to avoid the impacts on e2e congestion control. For instance, congestion management action must be delayed by more than a worst-case global RTT (e.g. 100ms), otherwise tunnel traffic management will not give normal e2e congestion control enough time to do its job, and the system could go unstable.

The detailed description of congestion management is out of scope of this document, as examples, congestion management such as circuit breaker [RFC8084] could be applied. Circuit breaker is an automatic mechanism to estimate congestion, and to terminate flow(s) when persistent congestion is detected to prevent network congestion collapse.

6.1 Example
This subsection provides an example of how the solution described in this document could work.

First of all, IPFIX template records are exchanged between ingress and egress to negotiate the format of data record, the example here is to measure the congestion level for the overall tunnel (caused by all the traffic in tunnel). After the negotiation is finished, ingress sends in-band message to egress, the message contains the number of each kind of ECN-marked packets (i.e. CE|CE, ECT|N-ECT and ECT|ECT) received until the sending of message.

After egress receives the message, the egress calculates CE-marked packet ratio and counts number of different kinds of ECN-marking packets received until receiving the message, then the egress sends a feedback message containing the counts together with the information in ingress’s message to ingress.

Figure 3 to Figure 6 below show the example procedure between ingress and egress.

```
+---------------------------------+----------------------+
<p>| Set ID=2                              Length=40         |
|---------------------------------|----------------------|
| Template ID=256                       Field Count =8    |
|---------------------------------|----------------------|
| tunnelEcnCeCeByteTotalCount         Field Length=8      |
|---------------------------------|----------------------|
| tunnelEcnEctNectByteTotalCount      Field Length=8      |
|---------------------------------|----------------------|
| tunnelEcnEctEctByteTotalCount       Field Length=8      |
|---------------------------------|----------------------|
| tunnelEcnCeCeByteTotalCount         Field Length=8      |
|---------------------------------|----------------------|
| tunnelEcnEctNectByteTotalCount      Field Length=8      |
|---------------------------------|----------------------|
| tunnelEcnEctEctByteTotalCount       Field Length=8      |
|---------------------------------|----------------------|
| tunnelEcnEcnNectByteTotalCount       Field Length=8      |
|---------------------------------|----------------------|
| tunnelEcnEctByteTotalCount           Field Length=8    |
|---------------------------------|----------------------|</p>
<table>
<thead>
<tr>
<th>tunnelEcnCEMarkedRatio</th>
<th>Field Length=4</th>
</tr>
</thead>
</table>
```

Figure 3: Template Record Sent From Egress to Ingress
Figure 4: Template Record Sent From Ingress to Egress

Figure 5 Traffic flow Between Ingress and Egress

|M| : Message Packet
+++

|P| : User Packet
+++
The following provides an example of how tunnel congestion level could be calculated:

Congestion Level could be divided into two categories: (1) slight congestion (no packets dropped); (2) serious congestion (packet dropping happen).

For slight congestion, the congestion level is indicated as the ratio of CE-marked packet:

\[ ce_{marked} = R; \]

For serious congestion, the congestion level is indicated as the number of volume loss:

\[ total_{ingress} = (A1 + B1 + C1) \]
\[ total_{egress} = (A2 + B2 + C2 + D + E) \]
\[ volume_{loss} = (total_{ingress} - total_{egress}) \]
This document describes the tunnel congestion calculation and feedback.

The tunnel endpoints are assumed to be deployed in the same administrative domain, so the ingress and egress will trust each other, the signaling traffic between ingress and egress will be protected utilizing security mechanism provided IPFIX (see section 11 in RFC7011).

From the consideration of privacy point of view, in case of fine grained congestion management, ingress is aware of the amount of traffic for specific application flows inside the tunnel which seems to be an invasion of privacy. But in any way, the ingress could The solution doesn’t introduce more privacy problem.

8. IANA Considerations

This document defines a set of new IPFIX Information Elements (IE), which need to be registered at IANA IPFIX Information Element Registry.

ElementID: TBD1
Name: tunnelEcnCeCePacketTotalCount
Data Type: unsigned64
Data Type Semantics: totalCounter
Status: current
Description: The total number of bytes of incoming packets with CE|CE ECN marking combination at the Observation Point since the Metering Process (re-)initialization for this Observation Point.
Units: octets

ElementID: TBD2
Name: tunnelEcnEct0NectPacketTotalCount
Data Type: unsigned64
Data Type Semantics: totalCounter
Status: current
Description: The total number of bytes of incoming packets with ECT(0)|N-ECT ECN marking combination at the Observation Point since the Metering Process (re-)initialization for this Observation Point.
Units: octets

ElementID: TBD3
Name: tunnelEcnEct1NectPacketTotalCount
Data Type: unsigned64
Data Type Semantics: totalCounter
Status: current
Description: The total number of bytes of incoming packets with
ECT(1)|N-ECT ECN marking combination at the Observation Point since
the Metering Process (re-)initialization for this Observation Point.
Units: octets

ElementID: TBD4
Name:tunnelEcnCeNectPacketTotalCount
Data Type: unsigned64
Data Type Semantics: totalCounter
Status: current
Description:The total number of bytes of incoming packets with CE|N-
ECT ECN marking combination at the Observation Point since the
Metering Process (re-)initialization for this Observation Point.
Units: octets

ElementID: TBD5
Name:tunnelEcnCeEct0PacketTotalCount
Data Type: unsigned64
Data Type Semantics: totalCounter
Status: current
Description:The total number of bytes of incoming packets with
CE|ECT(0) ECN marking combination at the Observation Point since the
Metering Process (re-)initialization for this Observation Point.
Units: octets

ElementID: TBD6
Name:tunnelEcnCeEct1PacketTotalCount
Data Type: unsigned64
Data Type Semantics: totalCounter
Status: current
Description:The total number of bytes of incoming packets with
CE|ECT(1) ECN marking combination at the Observation Point since the
Metering Process (re-)initialization for this Observation Point.
Units: octets

ElementID: TBD7
Name:tunnelEcnEct0Ect0PacketTotalCount
Data Type: unsigned64
Data Type Semantics: totalCounter
Status: current
Description:The total number of bytes of incoming packets with
ECT(0)|ECT(0) ECN marking combination at the Observation Point since
the Metering Process (re-)initialization for this Observation Point.
Units: octets

ElementID: TBD8
Name:tunnelEcnEct1Ect1PacketTotalCount
Data Type: unsigned64
Data Type Semantics: totalCounter
Status: current
Description: The total number of bytes of incoming packets with ECT(1)|ECT(1)ECN marking combination at the Observation Point since the Metering Process (re-)initialization for this Observation Point.
Units: octets
ElementID: TBD9
Name: tunnelEcnCEMarkedRatio
Data Type: float32
Status: current
Description: The ratio of CE-marked Packet at the Observation Point.

[TO BE REMOVED: This registration should take place at the following location: http://www.iana.org/assignments/ipfix/ipfix.xhtml#ipfix-information-elements]

9. References

9.1 Normative References


[RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion
9.2 Informative References


10. Acknowledgements

Thanks Bob Briscoe for his insightful suggestions on the basic mechanisms of congestion information collection and many other useful comments. Thanks David Black for his useful technical suggestions. Also, thanks Anthony Chan, Jake Holland, John Kaippallimalil and Vincent Roca for their careful reviews.

Authors' Addresses

Xinpeng Wei
Beiqing Rd. Z-park No.156, Haidian District,
Beijing, 100095, P. R. China
E-mail: weixinpeng@huawei.com

Zhu Lei
Beiqing Rd. Z-park No.156, Haidian District,
Beijing, 100095, P. R. China
E-mail: lei.zhu@huawei.com
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 January 19, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (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) ................................. 10
   5.5. Lite (LITE) ............................................. 11
   5.6. Maximum Segment Size (MSS) ............................ 13
   5.7. Fragmentation (FRAG) .................................... 14
      5.7.1. Coupling FRAG with LITE ........................... 16
   5.8. Timestamps (TIME) ....................................... 17
   5.9. Authentication and Encryption (AE) ....................... 17
   5.10. Experimental (EXP) ..................................... 18
6. UDP API Extensions ............................................. 19
7. Whose options are these? ...................................... 20
8. UDP options LITE option vs. UDP-Lite ......................... 20
9. Interactions with Legacy Devices .............................. 21
10. Options in a Stateless, Unreliable Transport Protocol ....... 22
11. UDP Option State Caching .................................... 22
12. Updates to RFC 768 ........................................... 22
13. Multicast Considerations ..................................... 23
14. Security Considerations ...................................... 23
15. IANA Considerations .......................................... 23
16. References ................................................... 24
   16.1. Normative References ................................. 24
   16.2. Informative References ............................... 24
17. Acknowledgments .............................................. 26
Appendix A. Implementation Information ........................... 28
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 length field is typically redundant with the size of the maximum space available as a transport protocol payload (see also discussion in Section 9).

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:

\[
\text{UDP}_\text{Length} = \text{IPv4}_\text{Total}_\text{Length} - \text{IPv4}_\text{IHL} \times 4
\]

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:

\[
\text{UDP}_\text{Length} = \text{IPv6}_\text{Payload}_\text{Length} - \text{sum(extension header lengths)}
\]

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 9). UDP-Lite used the difference in these pointers to indicate the partial coverage of the UDP Checksum, such that the UDP user data, UDP header, and UDP pseudoheader (a subset of the IP header) are covered by the UDP checksum but additional user data in the surplus area is not covered [RFC3828]. This document uses the surplus area for UDP transport options.

The UDP option area is thus defined as the location between the end of the UDP payload and the end of the IP datagram as a trailing options area. This area can occur at any valid byte offset, i.e., it need not be 16-bit or 32-bit aligned. In effect, this document redefines the UDP "Length" field as a "trailer offset".

UDP options are defined using a TLV (type, length, and optional value) syntax similar to that of TCP [RFC793]. They are typically a minimum of two bytes in length as shown in Figure 4, excepting only the one byte options "No Operation" (NOP) and "End of Options List" (EOL) described below.

```
+--------+--------+
|  Kind  | Length |
+--------+--------+
```

Figure 4 UDP option default format

>> UDP options MAY occur at any UDP length offset.

>> The UDP length MUST be at least as large as the UDP header (8) and no larger than the IP transport payload. Values outside this range MUST be silently discarded as invalid and logged where rate-limiting permits.

Others have considered using values of the UDP Length that is larger than the IP transport payload as an additional type of signal. Using
a value smaller than the IP transport payload is expected to be backward compatible with existing UDP implementations, i.e., to deliver the UDP Length of user data to the application and silently ignore the additional surplus area data. Using a value larger than the IP transport payload would either be considered malformed (and be silently dropped) or could cause buffer overruns, and so is not considered silently and safely backward compatible. Its use is thus out of scope for the extension described in this document.

>> UDP options MUST be interpreted in the order in which they occur in the UDP option area.

5. UDP Options

The following UDP options are currently defined:

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0*</td>
<td>-</td>
<td>End of Options List (EOL)</td>
</tr>
<tr>
<td>1*</td>
<td>-</td>
<td>No operation (NOP)</td>
</tr>
<tr>
<td>2*</td>
<td>2</td>
<td>Option checksum (OCS)</td>
</tr>
<tr>
<td>3*</td>
<td>4</td>
<td>Alternate checksum (ACS)</td>
</tr>
<tr>
<td>4*</td>
<td>4</td>
<td>Lite (LITE)</td>
</tr>
<tr>
<td>5*</td>
<td>4</td>
<td>Maximum segment size (MSS)</td>
</tr>
<tr>
<td>6*</td>
<td>8/10</td>
<td>Fragmentation (FRAG)</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>Timestamps (TIME)</td>
</tr>
<tr>
<td>8</td>
<td>(varies)</td>
<td>Authentication and Encryption (AE)</td>
</tr>
<tr>
<td>9-126</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.

>> 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 14.

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 an 8-bit ones-complement sum (Ones8) that covers all of the UDP options. OCS is 8-bits to allow the entire option to occupy a total of 16 bits. The primary purpose of OCS is to detect non-standard (i.e., non-option) uses of the surplus area.

OCS can be calculated by computing the 16-bit ones-complement sum and "folding over" the result (using carry wraparound). Note that OCS is direct, i.e., it is not negated or adjusted if zero (unlike the Internet checksum as used in IPv4, TCP, and UDP headers). OCS protects the option area from errors in a similar way that the UDP checksum protects the UDP user data.
> When present, the option checksum SHOULD occur as early as possible, preferably preceded by only NOP options for alignment and the LITE option if present.

OCS covers the entire UDP option area, including the Lite option as formatted before swapping (or relocation) for transmission (or, equivalently, after the swap/relocation after reception).

> 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, unless the endpoints have negotiated otherwise for this segment’s socket pair.

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 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.

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. 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  |
+---------+--------------+------------------+
      |            |
  +--------------+
```

*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.

```
+--------+--------+--------+--------+
| Kind=6 | Len=10 | Frag. Offset |
+--------+--------+--------+--------+
|          Identification           |
+--------+--------+--------+--------+
|    Checksum     |
+--------+--------+
```

Figure 15   UDP terminal FRAG option format

>> 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).

>> 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
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 5.7.1).

>> 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.

5.7.1. 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-payload packets, 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.

```
+---------+--------------+---------+
| UDP Hdr |   LiteFrag   |LITE|FRAG|
+---------+--------------+----+----+
<--------> ^^^^           ^^^^                      
Zero UDP Length |              |
+--------------+

Figure 16 Preparing FRAG as Lite data
```

```
+---------+--------------+----+
| UDP Hdr |LITE|LiteFrag |FRAG|
+---------+--------------+----+
<--------> |             ^
Zero UDP Length |             |
+-------------+

Figure 17 Lite option before transmission
```

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.
5.8. 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

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.

>> UDP 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.

>> UDP SHOULD make this RTT estimate available to the user application.

5.9. 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. UDP-AO supports NAT traversal in a similar manner as TCP-AO [RFC6978]. UDP-AO can also be extended to provide a similar encryption capability as TCP-AO-ENC, in a similar manner [To18ao]. For these reasons, the option is known as UDP-AE.
Like TCP-AO, UDP-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 UDP-AE’s Key Derivation Function (KDF) uses zeroes as the value for both ISNs. This means that the UDP-AE reuses keys when socket pairs are reused, unlike TCP-AO.

UDP-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 UDP-AE hash areas are zeroed before computing the UDP-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, UDP-AE can be configured to support NAT traversal, excluding one or both of the UDP ports [RFC6978].

5.10. 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].
6. 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.
7. 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. 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, ACS, or AE).

8. 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.

9. 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)
10. 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.

11. 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][To18cb]. 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 To18cb]

12. 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.
13. 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.

14. 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 [RFC5246] (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 5.9). 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.

15. 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 by IESG Approval or Standards Action [RFC8126].

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].

16. References

16.1. Normative References


16.2. Informative References


17. 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 is 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_ao</td>
<td>0</td>
<td>Default include AE</td>
</tr>
</tbody>
</table>

Socket options (sockopt), cached for outgoing datagrams:

<table>
<thead>
<tr>
<th>Name</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP_OPT</td>
<td>Enable UDP options (at all)</td>
</tr>
<tr>
<td>UDP_OPT_OCS</td>
<td>Enable UDP OCS option</td>
</tr>
<tr>
<td>UDP_OPT_ACS</td>
<td>Enable UDP ACS option</td>
</tr>
<tr>
<td>UDP_OPT_LITE</td>
<td>Enable UDP LITE option</td>
</tr>
<tr>
<td>UDP_OPT_MSS</td>
<td>Enable UDP MSS option</td>
</tr>
<tr>
<td>UDP_OPT_TIME</td>
<td>Enable UDP TIME option</td>
</tr>
<tr>
<td>UDP_OPT_FRAG</td>
<td>Enable UDP FRAG option</td>
</tr>
<tr>
<td>UDP_OPT_AE</td>
<td>Enable UDP AE option</td>
</tr>
</tbody>
</table>

Send/sendto parameters:

(TBD - currently using cached parameters)

Connection parameters (per-socketpair cached state, part UCB):

<table>
<thead>
<tr>
<th>Name</th>
<th>Initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>opts_enabled</td>
<td>net.ipv4.udp_opt</td>
</tr>
<tr>
<td>ocs_enabled</td>
<td>net.ipv4.udp_opt_ocs</td>
</tr>
</tbody>
</table>

The following option is included for debugging purposes, and MUST NOT be enabled otherwise.

System variables
net.ipv4.udp_opt_junk  0

System-level variables (sysctl):

<table>
<thead>
<tr>
<th>Name</th>
<th>default</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>net.ipv4.udp_opt_junk</td>
<td>0</td>
<td>Default use of junk</td>
</tr>
</tbody>
</table>

Socket options (sockopt):

<table>
<thead>
<tr>
<th>Name</th>
<th>params</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP_JUNK</td>
<td></td>
<td>Enable UDP junk option</td>
</tr>
<tr>
<td>UDP_JUNK_VAL</td>
<td>fillval</td>
<td>Value to use as junk fill</td>
</tr>
<tr>
<td>UDP_JUNK_LEN</td>
<td>length</td>
<td>Length of junk payload in bytes</td>
</tr>
</tbody>
</table>

Connection parameters (per-socketpair cached state, part UCB):

<table>
<thead>
<tr>
<th>Name</th>
<th>Initial value</th>
</tr>
</thead>
<tbody>
<tr>
<td>junk_enabled</td>
<td>net.ipv4.udp_opt_junk</td>
</tr>
<tr>
<td>junk_value</td>
<td>0xABCD</td>
</tr>
<tr>
<td>junk_len</td>
<td>4</td>
</tr>
</tbody>
</table>
Abstract

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 February 18, 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
1. Introduction

Versions 4 and 5 [RFC1928] of the SOCKS protocol were developed two decades ago and are in widespread use for circuit level gateways or as circumvention tools, and enjoy wide support and usage from various software, such as web browsers, SSH clients, and proxifiers. However, their design needs an update in order to take advantage of
the new features of transport protocols, such as TCP Fast Open [RFC7413], or to better assist newer transport protocols, such as MPTCP [RFC6824].

One of the main issues faced by SOCKS version 5 is that, when taking into account the TCP handshake, method negotiation, authentication, connection request and grant, it may take up to 5 RTTs for a data exchange to take place at the application layer. This is especially costly in networks with a large delay at the access layer, such as 3G, 4G, or satellite.

The desire to reduce the number of RTTs manifests itself in the design of newer security protocols. TLS version 1.3 [I-D.ietf-tls-tls13] defines a zero round trip (0-RTT) handshake mode for connections if the client and server had previously communicated.

TCP Fast Open [RFC7413] is a TCP option that allows TCP to send data in the SYN and receive a response in the first ACK, and aims at obtaining a data response in one RTT. The SOCKS protocol needs to concern itself with at least two TFO deployment scenarios: First, when TFO is available end-to-end (at the client, at the proxy, and at the server); second, when TFO is active between the client and the proxy, but not at the server.

This document describes the SOCKS protocol version 6. The key improvements over SOCKS version 5 are:

- The client sends as much information upfront as possible, and does not wait for the authentication process to conclude before requesting the creation of a socket.
- The connection request also mimics the semantics of TCP Fast Open [RFC7413]. As part of the connection request, the client can supply the potential payload for the initial SYN that is sent out to the server.
- The protocol can be extended via options without breaking backward-compatibility.
- The protocol can leverage the aforementioned options to support 0-RTT authentication schemes.

1.1. Revision log

Typos and minor clarifications are not listed.

draft-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.

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.
  o Option range for vendor-specific options.
  o Socket options.
  o Password authentication.
  o 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 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.

<table>
<thead>
<tr>
<th>Version Major</th>
<th>Minor</th>
<th>Command Code</th>
<th>Initial Data Size</th>
<th>Port</th>
<th>Address Type</th>
<th>Address</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>Variable</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2: SOCKS 6 Request

- Version: The major byte MUST be set to 0x06, and the minor byte MUST be set to 0x00.

- Command Code:
  * 0x00 NOOP: authenticate the client and do nothing.
  * 0x01 CONNECT: requests the establishment of a TCP connection.
  * 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.
- Number of Options: the number of SOCKS options that appear in the Options field.
- Options: see Section 8.
- Initial Data Size: A two-byte number in network byte order. In case of NOOP, BIND or UDP ASSOCIATE, this field MUST be set to 0. In case of CONNECT, this is the number of bytes of initial data that are supplied in the following field.
- Initial Data: The first octets of the data stream.

Clients can advertise their supported authentication methods by including an Authentication Method option (see Section 8.2).

The server MAY truncate the initial data to an arbitrary size and disregard the rest. This is will be communicated later to the client, should the authentication process be successful (see Section 7). As such, server implementations do not have to buffer the initial data while waiting for the (potentially malicious) client to authenticate.

5. Version Mismatch Replies

Upon receipt of a request starting with a version number other than 6.0, the proxy sends the following response:
Figure 3: SOCKS 6 Version Mismatch Reply

- **Version**: The major byte MUST be set to 0x06, and the minor byte MUST be set to 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 | Number of | Options  |
| Major | Minor |      | Options  |          |
+-------+-------+------+--------+-----------+----------+
|   1   |   1   |  1   |  1      |     1    | Variable |
+-------+-------+------+--------+-----------+----------+
```

Figure 4: SOCKS 6 Authentication Reply

- **Version**: The major byte MUST be set to 0x06, and the minor byte MUST be set to 0x00.

- **Type**:
  - 0x00: authentication successful.
  - 0x01: further authentication needed.

- **Method**: The chosen authentication method.

- **Number of Options**: the number of SOCKS options that appear in the Options field.

- **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. The client and proxy SHOULD NOT negotiate the encryption of the application data. Descriptions of such negotiations are beyond the scope of this memo. When the negotiation is complete (either successfully or unsuccessfully), the proxy sends another Authentication Reply.

7. Operation Replies

After the authentication negotiations are complete, the server sends an Operation Reply:

```
+-------+--------------+------+---------+----------+
| Reply | Initial Data | Bind | Address |   Bind   |
| Code  |    Offset    | Port |  Type   | Address  |
+-------+--------------+------+---------+----------+
|   1   |      2       |  2   |    1    | Variable |
+-------+--------------+------+---------+----------+

+-----------------------------------+
| Number of                        |
| Options  | Options       |
+-----------------------------------+
|     1     | 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

- **Initial Data Offset**: A two-byte number in network byte order. In case of BIND or UDP ASSOCIATE, this field MUST be set to 0. In case of CONNECT, it represents the offset in the plain data stream from which the client is expected to continue sending data.

- **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

- **Number of Options**: the number of SOCKS options that appear in the Options field.

- **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. The client MUST resume the data stream at the offset indicated by the Initial Data Offset field.

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

The relay of UDP packets is handled exactly as in SOCKS 5 [RFC1928].

8. SOCKS Options

SOCKS options have the following format:

+---------------+-------------+
| Kind | Length | Option Data |
+------+--------+-------------+
|  1   |   1    |   Variable  |
+------------------------+

Figure 6: SOCKS 6 Option

- Kind: MUST be allocated by IANA. (See Section 12.)
- Length: The 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 server (i.e. IP, TCP, UDP). A Stack option can affect either the proxy’s protocol on the...
client-proxy leg or on the proxy-server 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.

Stack options that are part of the same message MUST NOT contradict one another.

```
+---------------+--------+--------+------+----------+
| Kind | Length | Leg    | Level | Code | Data     |
+------+--------+--------+--------+------+----------+
|  1   |   1    | 2 bits | 6 bits |  1   | Variable |
+------+--------+--------+--------+------+----------+
```

Figure 7: Stack Option

- Kind: 0x01 (Stack option)
- Length: The length of the option.
- Leg:
  * 0x1: Client-Proxy Leg
  * 0x2: Proxy-Server Leg
  * 0x3: Both Legs
- Level:
  * 0x01: IP
  * 0x02: IPv4
  * 0x03: IPv6
  * 0x04: TCP
  * 0x05: UDP
- Code: Option code
- Data: Option-specific data
8.1.1. TFO options

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Leg</th>
<th>Level</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2 bits</td>
<td>6 bits</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 8: TFO Option

- Kind: 0x01 (Stack option)
- Length: MUST be 4.
- Leg: MUST be 0x2 (Proxy-Server Leg).
- Level: 0x04 (TCP).
- Code: 0x01

If a SOCKS Request contains a TFO option, the proxy SHOULD attempt to use TFO in case of a CONNECT command, or accept TFO in case of a BIND command. Otherwise, the proxy MUST NOT attempt to use TFO in case of a CONNECT command, or accept TFO in case of a BIND command.

In case of a CONNECT command, the proxy MAY include a TFO option in the Operation reply if TFO was attempted, the operation succeeded and the remote server supports TFO. In case of a BIND command, the proxy MAY include a TFO option in the first Operation reply to signal that it will accept an incoming TFO connection.

8.1.2. 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.

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Leg</th>
<th>Level</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2 bits</td>
<td>6 bits</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 9: Multipath TCP Option

- Kind: 0x01 (Stack option)
o Length: 4.

o Leg: MUST be 0x2 (Proxy-Server Leg).

o Level: 0x04 (TCP).

o Code: 0x02

8.1.3. 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.

+---------------+--------+--------+------+-----------+
| Kind | Length | Leg    | Level | Code | Scheduler |
+------+--------+--------+--------+------+-----------+
|  1   |   1    | 2 bits | 6 bits |  1   |     1     |
+------+--------+--------+--------+------+-----------+

Figure 10: MPTCP Scheduler Option

o Kind: 0x01 (Stack option)

o Length: MUST be 5.

o Leg: Either 0x01, 0x02, or 0x03 (Client-Proxy, Proxy-Client or Both legs).

o Level: 0x04 (TCP).

o Code: 0x03

o Scheduler:
  * 0x01: Default
  * 0x02: Round-Robin
  * 0x03: Redundant
8.2. Authentication Method options

Authentication Method options are used by clients to advertise supported authentication methods. They can be part of SOCKS Requests.

+-----------------+--------+---------------------+
| Kind | Length | Methods |                |
+------+--------+----------+---------------------+
|  1   |   1    | Variable |                |
+-----------------+--------+---------------------+

Figure 11: Authentication Method Option

- Kind: 0x02 (Authentication Method option)
- Length: The length of the option.
- Methods: One byte per advertised method. Method numbers are assigned by IANA.

Clients MUST support the "No authentication required" method. Clients SHOULD omit advertising the "No authentication required" 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   |   1    |  1    | Variable            |
+-----------------+--------+---------------------+

Figure 12: 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.
8.4. Idempotence options

To protect against duplicate SOCKS Requests, authenticated 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:

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Type</th>
<th>Option Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Variable</td>
</tr>
</tbody>
</table>

Figure 13: Idempotence Option

- Kind: 0x04 (Idempotence option)
- Length: The length of the option.
- Type:
  * 0x00: Token Request
  * 0x01: Token Window Advertisement
  * 0x02: Token Expenditure
  * 0x03: Token Expenditure Reply
- Option Data: The contents are specific to each type.
8.4.1. 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   |   1    |  1   |      4      |
+------+--------+------+-------------+
```

Figure 14: Token Request

- Kind: MUST be allocated by IANA. (See Section 12.)
- Length: 7
- Type: 0x00 (Token Request)
- 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   |   1    |  1   |      4      |      4      |
+------------------------------------------------------------------+
```

Figure 15: Token Window Advertisement

- Kind: 0x04 (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.

8.4.2. 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   |   1    |  1   |   4   |
+---------------+------+-------+

Figure 16: Token Expenditure

- Kind: 0x04 (Idempotence option)
- Length: 7
- Type: 0x02 (Token Expenditure)
- Token: The token being spent.

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   |   1    |  1   |       1       |
+---------------+------------+

Figure 17: Token Expenditure Response

- Kind: 0x04 (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 as soon as the client authenticates. If the token is not eligible for spending, the proxy MUST NOT attempt to honor the client’s SOCKS 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.

8.4.3. 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, provided that it is no longer than 252 bytes.
Figure 18: Password authentication via a SOCKS Option

- Kind: MUST be allocated by IANA. (See Section 12.)
- 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 impossible, or
- SOCKS is running on top of TLS and Early Data is not used.

11. Security Considerations

11.1. Large requests

Given the format of the request message, a malicious client could craft a request that is in excess of 100 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 exceeds an imposed hard cap, or
- the total size of the options exceeds an imposed hard cap, or
- the size of the initial data exceeds a hard cap.

Further, the server MAY choose not to buffer any initial data beyond what would be expected to fit in a TFO SYN’s payload.

### 11.2. 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 when sending a Token Request.

### 12. 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
- Idempotence options: 0x04
- A range for vendor-specific options

This document also requests that IANA allocate a port for SOCKS over TLS.
13. Acknowledgements

The protocol described in this draft builds upon and is a direct continuation of SOCKS 5 [RFC1928].

14. References

14.1. Normative References


14.2. Informative References


Authors' Addresses

Vladimir Olteanu
University Politehnica of Bucharest

Email: vladimir.olteanu@cs.pub.ro

Olteanu & Niculescu Expires February 18, 2019 [Page 24]
Simplemux. A generic multiplexing protocol
draft-saldana-tsvwg-simplemux-10

Abstract

The high amount of small packets present in nowadays’ networks results in a low efficiency, as the size of the headers and the payload of these packets can be in the same order of magnitude. In some situations, multiplexing a number of small packets into a bigger one is desirable in order to improve the efficiency. For example, a number of small packets can be sent together between a pair of machines if they share a common network path. This may happen between machines in different locations or even inside a datacenter with a number of servers hosting virtual machines. Thus, the traffic profile can be shifted from small to larger packets, reducing the network overhead and the number of packets per second to be managed by intermediate routers.

This document describes Simplemux, a protocol able to encapsulate a number of packets belonging to different protocols into a single packet. Small headers (separators) are added at the beginning of each multiplexed packet, including some flags, the packet length and a "Protocol" field. This allows the inclusion of a number of packets belonging to different protocols (the "multiplexed packets") on a packet of another protocol (the "tunneling protocol").

In order to reduce the overhead, the size of the multiplexing headers is kept very low (it may be a single byte when multiplexing packets of small size).

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). 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/.
1. Introduction

The high amount of small packets present in nowadays’ networks results in a low efficiency, when the size of the headers and the payload are in the same order of magnitude. In some situations, multiplexing a number of small packets into a bigger one is desirable in order to improve the efficiency. For example, a number of small packets can be sent together between a pair of machines if they share a common network path. This may happen between machines in different locations or even inside a datacenter with a number of servers hosting virtual machines. Thus, the traffic profile can be shifted from small to larger packets, thus reducing the network overhead and
the number of packets per second to be managed by intermediate routers.

This document describes Simplemux, a protocol able to encapsulate a number of packets belonging to different protocols into a single packet. This can be useful e.g. for grouping small packets and thus reducing the number of packets per second in a network.

Simplemux is a generic multiplexing protocol, i.e. it can be used to aggregate a number of packets belonging to a protocol, on a single packet belonging to other (or the same) protocol.

In this document we will talk about the "multiplexed" protocol, and the "tunneling" protocol, being Simplemux the "multiplexing" protocol. The "external header" will be the one of the "tunneling" protocol (see the figure (Figure 1))

```
+--------------------------------+                +--------------------------------+
|       Multiplexed Packet       |     Multiplexed protocol
+--------------------------------+                +--------------------------------+
|           Simplemux            |     Multiplexing protocol
+--------------------------------+                +--------------------------------+
|       Tunneling header         |     Tunneling protocol
+--------------------------------+                +--------------------------------+
```

Figure 1

As an example, if a number of small IPv6 packets have to travel over an IPv4 network, they can be multiplexed and put into a single IPv4 packet. In this case, IPv4 is the "tunneling" protocol and IPv6 is the "multiplexed" protocol. The IPv4 header is called in this case the "tunneling" or the "external" header. The simplified scheme of this packet would be:

|IPv4 hdr|Simplemux hdr|IPv6 packet|Simplemux hdr|IPv6 packet|...

1.1. 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].
1.2. Existing multiplexing protocols

Different multiplexing protocols have been approved by the IETF in the past:

- **TMux [RFC1692]**

  TMux is able to combine multiple short transport segments, independent of application type, and send them between a server and host pair. As stated in the reference, "The TMux protocol is intended to optimize the transmission of large numbers of small data packets. In particular, communication load is not measured only in bits per seconds but also in packets per seconds, and in many situation the latter is the true performance limit, not the former. The proposed multiplexing is aimed at alleviating this situation."

  A TMux message appears as:
  
  ```plaintext
  | IP hdr | TMux hdr | Transport segment | TMux hdr | Transport segment | ... |
  ```

  Therefore, the Transport Segment is not an entire IP packet, since it does not include the IP header.

  TMux works "between a server and host pair," so it multiplexes a number of segments between the same pair of machines. However, there are scenarios where a number of low-efficiency flows share a common path, but they do not travel between the same pair of machines.

- **PPPMux [RFC3153]**

  PPPMux "sends multiple PPP encapsulated packets in a single PPP frame. As a result, the PPP overhead per packet is reduced." Thus, it is able to multiplex complete IP packets, using separators.

  However, the use of PPPMux requires the use of PPP and L2TP in order to multiplex a number of packets together, as done in TCRTP [RFC4170]. Thus, it introduces more overhead and complexity.

  An IP packet including a number of them using PPPMux appears as:
  
  ```plaintext
  | IP hdr | L2TP hdr | PPP hdr | PPPMux hdr | packet | PPPMux hdr | packet | ... |
  ```

  The scheme proposed by PPPMux is similar to the Compound-Frames of PPP LCP Extensions [RFC1570]. The key differences are that PPPMux is more efficient and that it allows concatenation of variable sized frames.

***
The definition of a protocol able to multiplex complete packets, avoiding the need of other protocols as PPP is seen as convenient. The multiplexed packets can be of any kind, since a "Protocol Number" field can be added to each of them. Not all the packets multiplexed together must belong to the same protocol. The general scheme of Simplemux is:

```
|tunnel hdr|Simplemux hdr|packet||Simplemux hdr|packet||...|
```

The Simplemux header includes the "Protocol Number" field, so it permits the multiplexing of different kinds of packets in the same bundle.

We will also refer to the Simplemux header with the terms "separator," "Simplemux separator" or "mux separator". In the figures we will also use the abbreviation "Smux".

When applied to IP packets, the scheme of a multiplexed packet becomes:

```
|tunnel hdr|Simplemux hdr|IP packet||Simplemux hdr|IP packet||...|
```

1.3. Benefits of multiplexing

The benefits of multiplexing are:

- Tunneling a number of packets together. If a number of packets have to be tunneled through a network segment, they can be multiplexed and then sent together using a single external header. This will avoid the need for adding a tunneling header to each of the packets, thus reducing the overhead.

- Reduction of the amount of packets per second in the network. It is desirable for two main reasons: first, network equipment has a limitation in terms of the number of packets per second it can manage, i.e. many devices are not able to send small packets back to back due to processing delay.

- Bandwidth reduction. The presence of high rates of tiny packets translates into an inefficient usage of network resources, so there is a need for mechanisms able to reduce the overhead introduced by low-efficiency flows. When combined with header compression, as done in TCRTP [RFC4170] multiplexing may produce significant bandwidth savings, which are interesting for network operators, since they may alleviate the traffic load in their networks.

- Energy savings: a lower amount of packets per second will reduce energy consumption in network equipment since, according to [Bolla],

internal packet processing engines and switching fabric require 60% and 18% of the power consumption of high-end routers respectively. Thus, reducing the number of packets to be managed and switched will reduce the overall energy consumption. The measurements deployed in [Chabarek]on commercial routers corroborate this. A study using different packet sizes was presented, and the tests with big packets showed that energy consumption gets reduced, since a non-negligible amount of energy is associated to header processing tasks, and not only to the sending of the packet itself.

2. Description of the scenario

Simplemux works between a pair of machines. It creates a tunnel between an "ingress" and an "egress". They MAY be the endpoints of the communication, but they MAY also be middleboxes able to multiplex packets belonging to different flows. Different mechanisms MAY be used in order to classify flows according to some criteria (sharing a common path, kind of service, etc.) and to select the flows to be multiplexed and sent to the egress (see Figure 2).

```
+-------+          +-------+          +-------+
|       |          |       |          |       |
|       |          |       |          |       |
|       |          |       |          |       |
|       |          +---------+      (  Network  )      +---------+
|       |          _  _              _  _              |
|       |          ( '   )_     ===>  ( '   )_     ===>  |
|       |          ----------------> (_   (_ .  _) _)  ----------------->
|       |          (_ ( _ . _ ) _ ) --//--<_------Simplemux-------->
|-------+          +-------+          +-------+
```

Figure 2

3. Protocol description

A Simplemux packet consists of:

- An external header that is used as the tunneling header for the whole packet.

- A series of pairs "Simplemux header" + "packet" of the multiplexed protocol.

This is the scheme of a Simplemux packet:

```
|tun hdr||Simplemux hdr|packet||Simplemux hdr|packet||...
```
The Simplemux header has two different forms: one for the "First Simplemux header," and another one for the rest of the Simplemux headers (called "Non-first Simplemux headers"):

- First Simplemux header (after the tunneling header, and before the first multiplexed packet):

In order to allow the multiplexing of packets of any length, the number of bytes expressing the length is variable, and a field called "Length Extension" (LXT, one bit) is used to flag if the current byte is the last one including length information. This is the structure of a First Simplemux header:

```
| SPB (1 bit) | LXT (1 bit) | length (6 bits) | LXT (1 bit) | length (7 bits) | ... | Protocol (8 bits) |
```

- Single Protocol Bit (SPB, one bit) only appears in the first Simplemux header. It is set to 1 if all the multiplexed packets belong to the same protocol (in this case, the "Protocol" field will only appear in the first Simplemux header). It is set to 0 when each packet MAY belong to a different protocol.

- Length Extension (LXT, one bit) is 0 if the current byte is the last byte where the length of the first packet is included, and 1 in other case.

- Length (LEN, 6, 13, 20, etc. bits): This is the length of the multiplexed packet (in bytes), not including the length field. If the length of the multiplexed packet is less than 64 bytes (less than or equal to 63 bytes), the first LXT is set to 0 and the 6 bits of the length field are the length of the multiplexed packet. If the length of the multiplexed packet is equal or greater than 64 bytes, additional bytes are added. The first bit of each of the added bytes is the LXT. If LXT is set to 1, it means that there is an additional byte for expressing the length. This allows to multiplex packets of any length (see the next figures).

- Protocol (8 bits) is the Protocol field of the multiplexed packet, according to IANA "Assigned Internet Protocol Numbers."

As an example, a First Simplemux header before a packet smaller than 64 (2^6) bytes would be 2 bytes long:
A First Simplemux header before a packet with a length greater or equal to 64 bytes, and smaller than 8192 bytes (2^13) will be 3 bytes long:

```
0                   1                   2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
S |L |           |               |
P |X |  Length   |   Protocol    |
B |T | (6 bits)  |   (8 bits)    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

Figure 3

In this case, the length of the packet will be the number expressed by the concatenation of the bits of Length 1 - Length 2 (total 13 bits). Length 1 includes the 6 most significant bits and Length 2 the 7 less significant bits.

A First Simplemux header before a packet with a length greater of equal to 8192 bytes, and smaller than 1048576 bytes (2^20) would be 4 bytes long:

```
0                   1                   2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
S |L |           |L |             |               |
P |X |  Length 1 |X |  Length 2   |   Protocol    |
B |T | (6 bits)  |T |  (7 bits)   |   (8 bits)    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4
In this case, the length of the packet will be the number expressed by the concatenation of the bits of Length 1 - Length 2 - Length 3 (total 20 bits). Length 1 includes the 6 most significant bits and Length 3 the less 7 significant bits.

More bytes can be added to the length if required, using the same scheme: 1 LXT byte plus 7 bits for expressing the length.

- Subsequent (Non-first) Simplemux headers (before the other packets):

The Non-first Simplemux headers also employ a format allowing the multiplexing of packets of any length, so the number of bytes expressing the length is variable, and the field Length Extension (LXT, one bit) is used to flag if the current byte is the last one including length information. This is the structure of a Non-first Simplemux header:

```
| LXT (1 bit) | length (7 bits) | ... | Protocol (8 bits, optional) |
```

- Length Extension (LXT, one bit) is 0 if the current byte is the last byte where the length of the packet is included, and 1 in other case.

- Length (LEN, 7, 14, 21, etc. bits): This is the length of the multiplexed packet (in bytes), not including the length field. If the length of the multiplexed packet is less than 128 bytes (less than or equal to 127 bytes), LXT is set to 0 and the 7 bits of the length field represent the length of the multiplexed packet. If the length of the multiplexed packet is greater than 127 bytes, additional bytes are added. The first bit of each of the added bytes is the LXT. If LXT is set to 1, it means that there is an additional byte for expressing the length. This allows to multiplex packets of any length (see the next figures).
- Protocol (8 bits) is the Protocol field of the multiplexed packet, according to IANA "Assigned Internet Protocol Numbers". It only appears in Non-first headers if the Single Protocol Bit (SPB) of the First Simplemux header is set to 1.

As an example, a Non-first Simplemux header before a packet smaller than 128 bytes, when the protocol bit has been set to 0 in the first header, would be 1 byte long:

```
0 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+-+
|L| X  Length |
|T|  (7 bits) |
^+-+-+-+-+-+-+-+-+
```

SPB = 0 in the first header

Figure 6

A Non-first Simplemux header before a packet with a length greater or equal to 128 bytes, and smaller than 16384 (2^14), when the protocol bit has been set to 0 in the first header, will be 2 bytes long:

```
0 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|L| L  Length 1 |L| Length 2 |
|T|  (7 bits) |T|  (7 bits) |
^+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

SPB = 0 in the first header

Figure 7

A Non-first Simplemux header before a packet with a length greater or equal to 16384 bytes, and smaller than 2097152 bytes (2^21), when the protocol bit has been set to 0 in the first header, will be 3 bytes long:
SPB = 0 in the first header

Figure 8

In this case, the length of the packet will be the number expressed by the concatenation of the bits of Length 1 - Length 2 - Length 3 (total 21 bits). Length 1 includes the 7 most significant bits and Length 3 the 7 less significant bits.

More bytes can be added to the length if required, using the same scheme: 1 LXT byte plus 7 bits for expressing the length.

A Non-first Simplemux header before a packet smaller than 128 bytes, when the protocol bit has been set to 1 in the first header, will be 2 bytes long:

SPB = 1 in the first header

Figure 9

A Non-first Simplemux header before a packet with a length greater or equal to 128 bytes, and smaller than 16384 (2^14), when the protocol bit has been set to 1 in the first header, will be 3 bytes long:
SPB = 1 in the first header

Figure 10

A Non-first Simplemux header before a packet with a length greater of equal to 16384 bytes, and smaller than 2097152 bytes (2^21), when the protocol bit has been set to 1 in the first header, will be 4 bytes long:

SPB = 1 in the first header

Figure 11

In this case, the length of the packet will be the number expressed by the concatenation of the bits of Length 1 - Length 2 - Length 3 (total 21 bits). Length 1 includes the 7 most significant bits and Length 3 the 7 least significant bits.

More bytes can be added to the length if required, using the same scheme: 1 LXT byte plus 7 bits for expressing the length.

These would be some examples of the whole bundles:

Case 1: All the packets belong to the same protocol: The first Simplemux header would be 2 or 3 bytes (for usual packet sizes), and the other Simplemux headers would be 1 or 2 bytes. For small packets (< 128 bytes), the Simplemux header would only require one byte.
Case 2: Each packet may belong to a different protocol: All the Simplemux headers would be 2 or 3 bytes (for usual packet sizes).

| tun | 1 | 0 | len | Protocol | pkt | 0 | len | pkt | 1 | len | pkt | ... |
|     |   |   |     |          |     |   |     |     |   |     |     |     |
|     | v |     |     |          | v   | v | v   |     | v |     |     |     |
| (6 bits) | (7 bits) | (14 bits) |

| tun | 1 | 1 | len | 0 | len | Protocol | pkt | 0 | len | pkt | 1 | len | 0 | len | pkt | ... |
|     |   |   |     |   |     |          |     |   |     |     |   |     |   |     |     |     |
|     | v | v | v   |     | v   | v       |     | v |     | v   | v |     | v   |     | v   |     |
| (13 bits) | (7 bits) | (14 bits) |

Figure 12

| tun | 0 | 0 | len | Prot | pkt | 0 | len | Prot | pkt | 1 | len | 0 | len | Prot | pkt | ... |
|     |   |   |     |     |     |   |     |     |     |   |     |   |     |     |     |     |
|     | v |     |     |     |     | v   | v   | v   |     | v |     | v   | v   | v   |     | v   |
| (6 bits) | (7 bits) | (14 bits) |

| tun | 0 | 1 | len | 0 | len | Prot | pkt | 0 | len | Prot | pkt | 1 | len | 0 | len | Prot | pkt | ... |
|     |   |   |     |   |     |     |     |   |     |     |     |   |     |   |     |     |     |     |
|     | v | v | v   |     | v   | v       |     | v |     | v   | v |     | v   |     | v   |     |
| (13 bits) | (7 bits) | (14 bits) |

Figure 13

4. Acknowledgements

This work has been partially funded by the EU H2020 Wi-5 project (Grant Agreement no: 644262), the Spanish Ministry of Economy and Competitiveness project TIN2015-64770-R, in cooperation with the European Regional Development Fund (TIN2016-76770-R) and Gobierno de Aragon and FEDER "Construyendo Europa desde Aragon" (Research Group T31_17R).

5. IANA Considerations

A protocol number for Simplemux should be requested to IANA.

As a provisional solution for IP networks, the ingress and the egress optimizers may agree on a UDP port, and use IP/UDP as the multiplexing protocol.
6. Security Considerations

Simplemux protocol has been developed in such a way that packet aggregation and security can be simultaneously applied to the same traffic flows, i.e. a single security header could protect a number of packets belonging to different flows.

As a consequence, the overall efficiency could be improved, as the number of security headers could be reduced from \( N \) (being \( N \) the number of multiplexed packets) to 1.

7. References

7.1. Normative References


7.2. Informative References


Authors' Addresses

Jose Saldana
University of Zaragoza
Dpt. IEC Ada Byron Building
Zaragoza  50018
Spain
Phone: +34 976 762 698
Email: jsaldana@unizar.es

Julian Fernandez Navajas
University of Zaragoza
Dpt. IEC Ada Byron Building
Zaragoza  50018
Spain
Phone: +34 976 761 963
Email: navajas@unizar.es

Jose Ruiz Mas
University of Zaragoza
Dpt. IEC Ada Byron Building
Zaragoza  50018
Spain
Phone: +34 976 762 158
Email: jruiz@unizar.es