

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

B. Briscoe
CableLabs
November 1, 2018

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

Abstract

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

Status of This Memo

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

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

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

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

Copyright Notice

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

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

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

Table of Contents

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

1. Introduction

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

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

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

1.1. Terminology

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

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

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

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

Pure L4S: L4S without unresponsive traffic.

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

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

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

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

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

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

1.2. Document Roadmap

{ToDo}

2. Architectural Comparison of L4S and Diffserv

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

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

2.1. Overlaps between L4S and Diffserv

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

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

2.2. Differences between L4S and Diffserv

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

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

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

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

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

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

3. Low Latency Diffserv Classes within a DualQ Bandwidth Pool

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

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

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

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

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

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

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

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

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

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

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

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

Notes for Table 1:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4.1. DualQ Complemented by an Assured Bandwidth Service

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

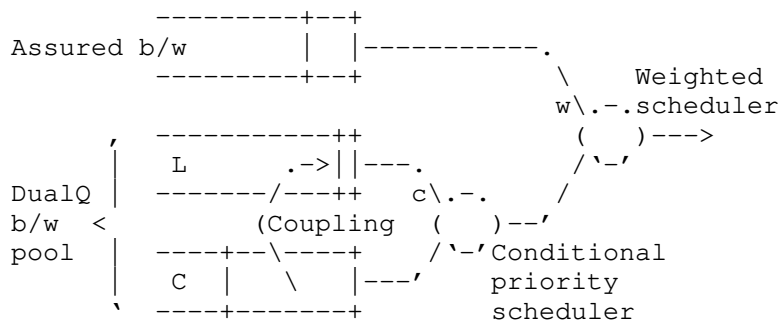


Figure 1: How to Complement a DualQ with an Assured Bandwidth Service

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

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

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

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

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

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

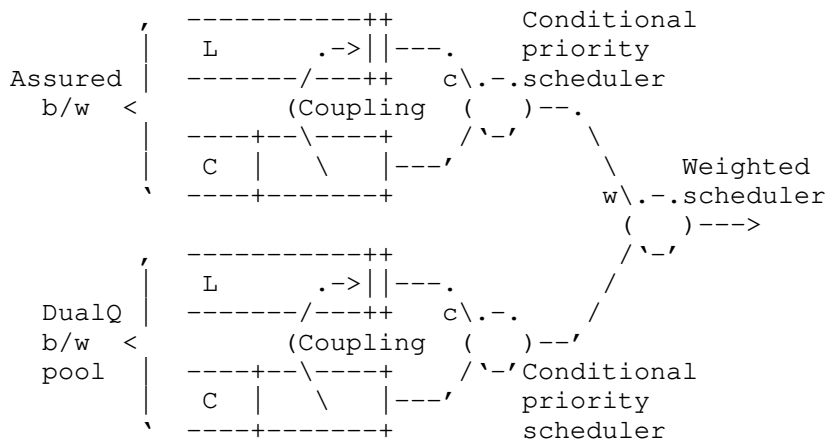


Figure 2: How to Complement a DualQ with an Assured Bandwidth Service that also Supports L4S

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

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

4.2. DualQ Complemented by a Guaranteed Low Latency Service

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

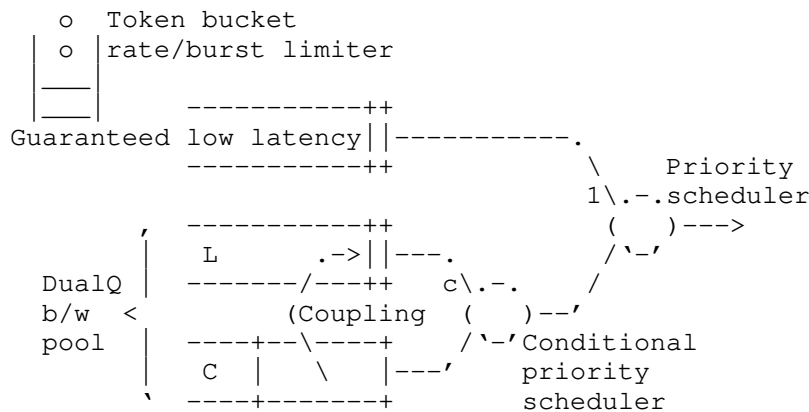


Figure 3: How to Complement a DualQ with a Guaranteed Low Latency Service

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

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

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

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

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

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

4.3. DualQ Complemented by a Scavenger Service

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

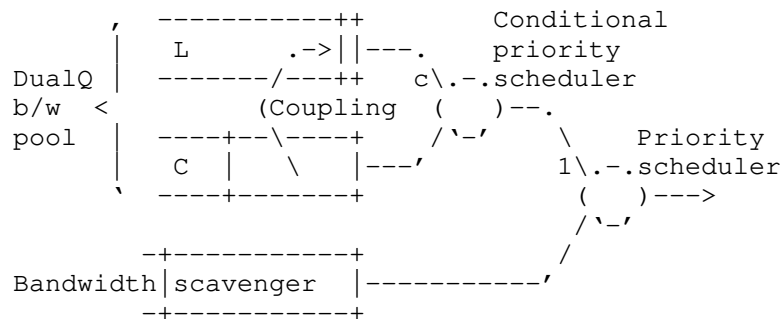


Figure 4: How to Complement a DualQ with a Bandwidth Scavenger Service

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

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

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

5. Coupling More than Two AQMs within a Bandwidth Pool

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

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

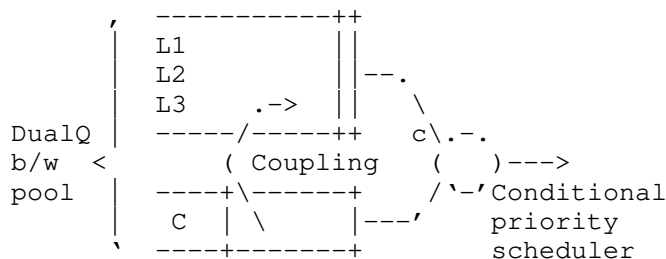


Figure 5: Coupling the Classic AQM to Multiple L4S AQMs

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

6. Applicability of Coupled AQM to Global Diffserv PHBs

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

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

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

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

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

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

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

7. Best Practice for Classification and Marking

7.1. Never Re-Mark a DSCP

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

7.2. Classification Order

7.2.1. Classification Order: Problem

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

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

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

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

7.2.2. Classification Order: Solutions

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

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

8. Policing and Traffic Conditioning

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

9. IANA Considerations

This specification contains no IANA considerations.

10. Security Considerations

{ToDo}

11. Comments Solicited

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

12. Acknowledgements

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

13. References

13.1. Normative References

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

13.2. Informative References

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

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

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

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

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

Appendix A. Open Issues

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

Author's Address

Bob Briscoe
CableLabs
UK

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

INTERNET-DRAFT
Intended status: Proposed Standard

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

Expires: August 4, 2019

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

Abstract

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

Status of This Memo

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

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

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

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

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

Table of Contents

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

1. Introduction

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

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

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

1.1 NSH Background

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

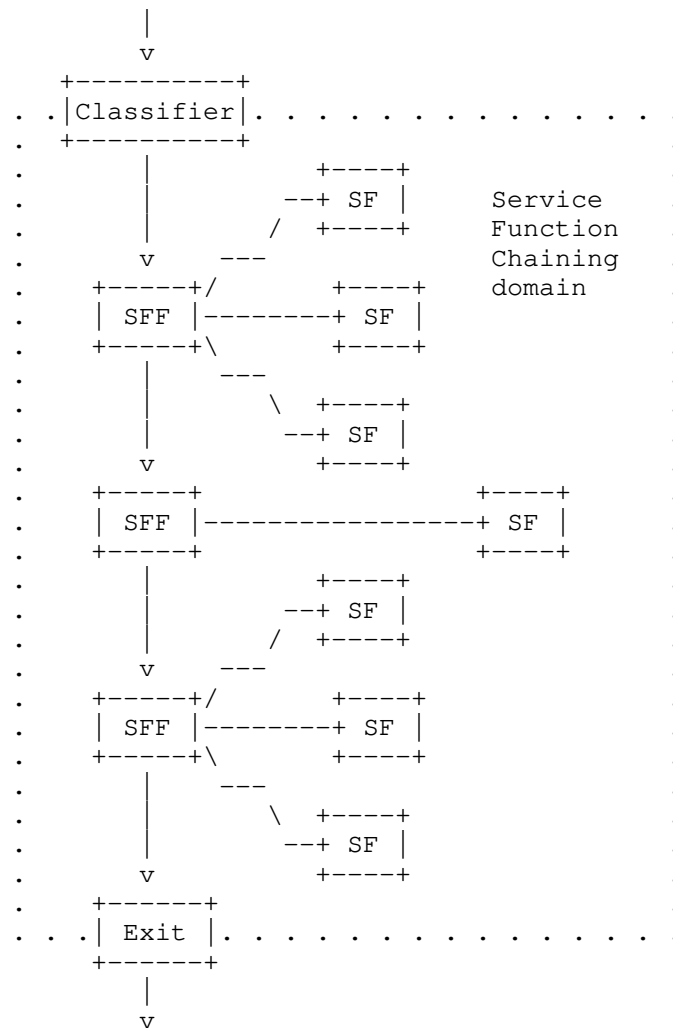


Figure 1. Example SFC Path Forwarding Nodes

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

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

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

1.2 ECN Background

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

1.3 Tunnel Congestion Feedback Background

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

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

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

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

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

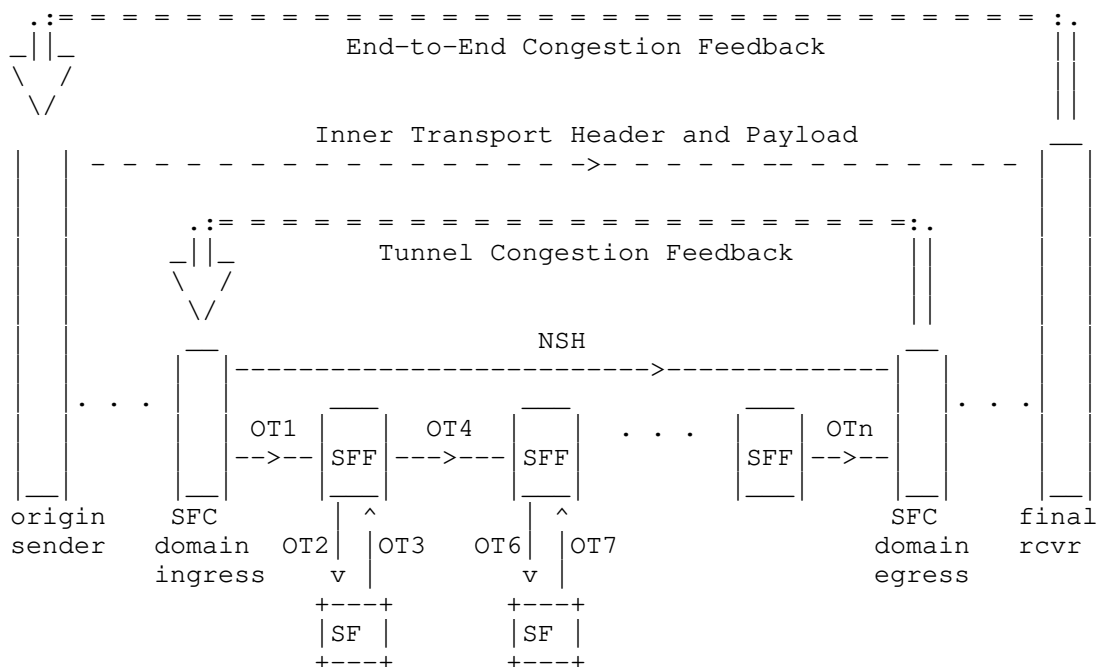


Figure 2: Congestion Feedback across an SFC Domain

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

1.4 Conventions Used in This Document

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

Acronyms:

AQM - Active Queue Management [RFC7567]

CE - Congestion Experienced [RFC3168]

downstream - The direction from ingress to egress

ECN - Explicit Congestion Notification [RFC3168]

ECT - ECN Capable Transport [RFC3168]

IPFIX - IP Flow Information Export [RFC7011]

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

NSH - Network Service Header [RFC8300]

SF - Service Function [RFC7665]

SFC - Service Function Chaining [RFC7665]

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

TLV - Type Length Value

upstream - The direction from egress to ingress

2. The NSH ECN Field

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

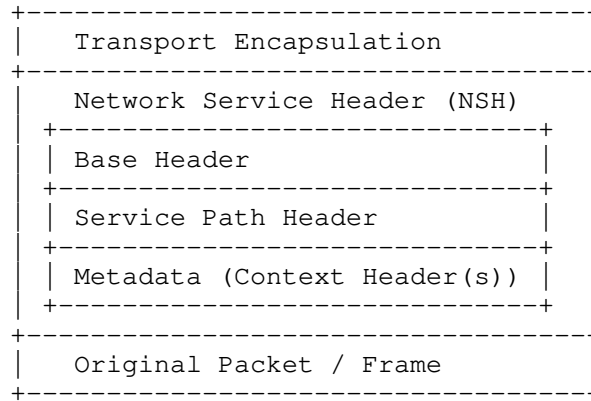


Figure 3. Data Encapsulation with the NSH

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

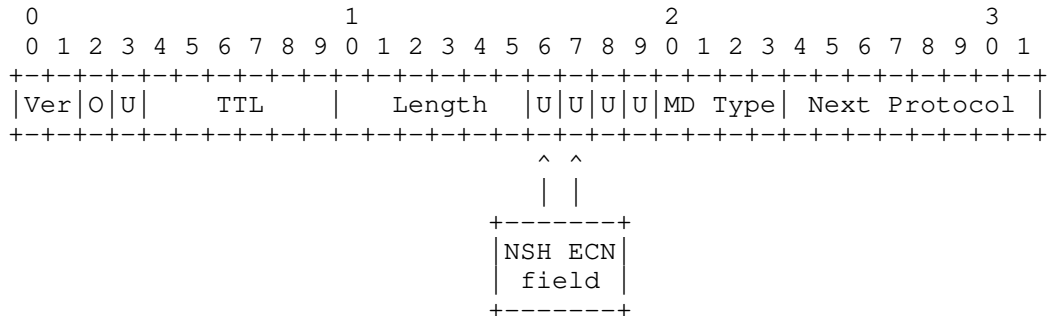


Figure 4: NSH Base Header

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

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

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

Table 1. ECN Field Code Points

3. ECN Support in the NSH

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

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

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

ECN Propagation:

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

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

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

ECN Marking:

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

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

3.1 At The Ingress

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

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

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

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

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

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

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

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

3.2 At Transit Nodes

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

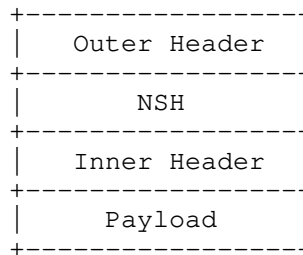


Figure 5. Packet in Transit

3.2.1 At NSH Transit Nodes

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

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

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

3.2.2 At an SF/Proxy

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

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

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

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

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

3.2.3 At Other Forwarding Nodes

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

3.3 At Exit/Egress

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

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

Table 3. Exit ECN Fields Merger

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

3.4 Conservation of Packets

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

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

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

4. Tunnel Congestion Feedback Support

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

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

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

5. IANA Considerations

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

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

6. Security Considerations

For general NSH security considerations, see [RFC8300].

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

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

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

7. Acknowledgements

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

Joel Halpern, Tal Mizrahi, Xinpeng Wei

Normative References

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

Informative References

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

Authors' Addresses

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

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

Bob Briscoe
Independent
UK

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

Andrew G. Malis
Huawei Technologies

Email: agmalis@gmail.com

Copyright and IPR Provisions

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

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

TSVWG
Internet-Draft
Intended status: Informational
Expires: February 28, 2019

G. Fairhurst
University of Aberdeen
C. Perkins
University of Glasgow
August 27, 2018

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.

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

Table of Contents

1. Introduction	2
2. Context and Rationale	3
3. Current uses of Transport Headers within the Network	9
3.1. Observing Transport Information in the Network	9
3.2. Transport Measurement	15
3.3. Use for Network Diagnostics and Troubleshooting	18
3.4. Observing Headers to Implement Network Policy	19
4. Encryption and Authentication of Transport Headers	19
4.1. Authenticating the Transport Protocol Header	21
4.2. Encrypting the Transport Payload	22
4.3. Encrypting the Transport Header	22
4.4. Authenticating Transport Information and Selectively Encrypting the Transport Header	22
4.5. Optional Encryption of Header Information	23
5. Addition of Transport Information to Network-Layer Protocol Headers	23
6. Implications of Protecting the Transport Headers	24
6.1. Independent Measurement	24
6.2. Characterising "Unknown" Network Traffic	25
6.3. Accountability and Internet Transport Protocols	25
6.4. Impact on Research, Development and Deployment	26
7. Conclusions	27
8. Security Considerations	29
9. IANA Considerations	31
10. Acknowledgements	31
11. Informative References	31
Appendix A. Revision information	37
Authors' Addresses	37

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

also considers anticipated implications on transport and application evolution.

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

[Measure], Section 3.2. 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:

- o 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.
- o 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:

- o Flows need to be identified at the level required to perform the observation;
- o The protocol and version of the header need to be visible. 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;
- o 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 make 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 of 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 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 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 Section 3.2.1. 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 case, 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/back-haul, point-to-point radio) has the complexity of a subsystem that performs radio resource management, with direct impact on the available capacity, and potentially loss/reordering of packets. The impact of the pattern of loss and congestion, differs for different traffic types, correlation with propagation and interference can all have significant impact on the cost and performance of a provided service. The need for this type of information is expected to increase as operators bring together heterogeneous types of network equipment and seek to deploy opportunistic methods to access radio spectrum.

3.4. 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:

- o 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 led 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].
- o 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.

- o 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.
- o 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.

The next subsections briefly review some security design options for transport protocols. A Survey of Transport Security Protocols [I-D.ietf-taps-transport-security] provides more details concerning commonly used encryption methods at the transport layer.

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 observe 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 802.1lag or in-situ OAM [I-D.ietf-ippm-ioam-data]) and removing the additional header at the egress of the maintenance domain. This approach enables some types of measurements, but does not cover the entire range of measurements described in this document. In some cases, it can be difficult to position measurement tools at the required segments/nodes and there can be challenges in correlating the downstream/upstream information when in-band OAM data is inserted by an on-path device.

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 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.

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:

- o 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).
- o 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]
Brockners, F., Bhandari, S., Pignataro, C., Gredler, H., Leddy, J., Youell, S., Mizrahi, T., Mozes, D., Lapukhov, P., Chang, R., daniel.bernier@bell.ca, d., and J. Lemon, "Data Fields for In-situ OAM", draft-ietf-ippm-ioam-data-03 (work in progress), June 2018.

- [I-D.ietf-quic-transport]
Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-14 (work in progress), August 2018.
- [I-D.ietf-taps-transport-security]
Pauly, T., Perkins, C., Rose, K., and C. Wood, "A Survey of Transport Security Protocols", draft-ietf-taps-transport-security-02 (work in progress), June 2018.
- [I-D.ietf-tcpinc-tcpcrypt]
Bittau, A., Giffin, D., Handley, M., Mazieres, D., Slack, Q., and E. Smith, "Cryptographic protection of TCP Streams (tcpcrypt)", draft-ietf-tcpinc-tcpcrypt-12 (work in progress), June 2018.
- [I-D.ietf-tsvwg-l4s-arch]
Briscoe, B., Schepper, K., and M. Bagnulo, "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture", draft-ietf-tsvwg-l4s-arch-02 (work in progress), March 2018.
- [I-D.thomson-quic-grease]
Thomson, M., "More Apparent Randomization for QUIC", draft-thomson-quic-grease-00 (work in progress), December 2017.
- [I-D.trammell-plus-abstract-mech]
Trammell, B., "Abstract Mechanisms for a Cooperative Path Layer under Endpoint Control", draft-trammell-plus-abstract-mech-00 (work in progress), September 2016.
- [Latency] Briscoe, B., "Reducing Internet Latency: A Survey of Techniques and Their Merits", November 2014.
- [Measure] Fairhurst, G., Kuehlewind, M., and D. Lopez, "Measurement-based Protocol Design", June 2017.
- [RFC1273] Schwartz, M., "Measurement Study of Changes in Service-Level Reachability in the Global TCP/IP Internet: Goals, Experimental Design, Implementation, and Policy Considerations", RFC 1273, DOI 10.17487/RFC1273, November 1991, <<https://www.rfc-editor.org/info/rfc1273>>.

- [RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <<https://www.rfc-editor.org/info/rfc2474>>.
- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, DOI 10.17487/RFC2914, September 2000, <<https://www.rfc-editor.org/info/rfc2914>>.
- [RFC3135] Border, J., Kojo, M., Griner, J., Montenegro, G., and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations", RFC 3135, DOI 10.17487/RFC3135, June 2001, <<https://www.rfc-editor.org/info/rfc3135>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC3234] Carpenter, B. and S. Brim, "Middleboxes: Taxonomy and Issues", RFC 3234, DOI 10.17487/RFC3234, February 2002, <<https://www.rfc-editor.org/info/rfc3234>>.
- [RFC3393] Demichelis, C. and P. Chimento, "IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)", RFC 3393, DOI 10.17487/RFC3393, November 2002, <<https://www.rfc-editor.org/info/rfc3393>>.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <<https://www.rfc-editor.org/info/rfc3550>>.
- [RFC4302] Kent, S., "IP Authentication Header", RFC 4302, DOI 10.17487/RFC4302, December 2005, <<https://www.rfc-editor.org/info/rfc4302>>.
- [RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <<https://www.rfc-editor.org/info/rfc4303>>.
- [RFC4585] Ott, J., Wenger, S., Sato, N., Burmeister, C., and J. Rey, "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, DOI 10.17487/RFC4585, July 2006, <<https://www.rfc-editor.org/info/rfc4585>>.

- [RFC4737] Morton, A., Ciavattone, L., Ramachandran, G., Shalunov, S., and J. Perser, "Packet Reordering Metrics", RFC 4737, DOI 10.17487/RFC4737, November 2006, <<https://www.rfc-editor.org/info/rfc4737>>.
- [RFC5218] Thaler, D. and B. Aboba, "What Makes for a Successful Protocol?", RFC 5218, DOI 10.17487/RFC5218, July 2008, <<https://www.rfc-editor.org/info/rfc5218>>.
- [RFC5236] Jayasumana, A., Piratla, N., Banka, T., Bare, A., and R. Whitner, "Improved Packet Reordering Metrics", RFC 5236, DOI 10.17487/RFC5236, June 2008, <<https://www.rfc-editor.org/info/rfc5236>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC5481] Morton, A. and B. Claise, "Packet Delay Variation Applicability Statement", RFC 5481, DOI 10.17487/RFC5481, March 2009, <<https://www.rfc-editor.org/info/rfc5481>>.
- [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <<https://www.rfc-editor.org/info/rfc5925>>.
- [RFC6056] Larsen, M. and F. Gont, "Recommendations for Transport-Protocol Port Randomization", BCP 156, RFC 6056, DOI 10.17487/RFC6056, January 2011, <<https://www.rfc-editor.org/info/rfc6056>>.
- [RFC6269] Ford, M., Ed., Boucadair, M., Durand, A., Levis, P., and P. Roberts, "Issues with IP Address Sharing", RFC 6269, DOI 10.17487/RFC6269, June 2011, <<https://www.rfc-editor.org/info/rfc6269>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6437] Amante, S., Carpenter, B., Jiang, S., and J. Rajahalme, "IPv6 Flow Label Specification", RFC 6437, DOI 10.17487/RFC6437, November 2011, <<https://www.rfc-editor.org/info/rfc6437>>.

- [RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <<https://www.rfc-editor.org/info/rfc7258>>.
- [RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <<https://www.rfc-editor.org/info/rfc7525>>.
- [RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <<https://www.rfc-editor.org/info/rfc7567>>.
- [RFC7624] Barnes, R., Schneier, B., Jennings, C., Hardie, T., Trammell, B., Huitema, C., and D. Borkmann, "Confidentiality in the Face of Pervasive Surveillance: A Threat Model and Problem Statement", RFC 7624, DOI 10.17487/RFC7624, August 2015, <<https://www.rfc-editor.org/info/rfc7624>>.
- [RFC7872] Gont, F., Linkova, J., Chown, T., and W. Liu, "Observations on the Dropping of Packets with IPv6 Extension Headers in the Real World", RFC 7872, DOI 10.17487/RFC7872, June 2016, <<https://www.rfc-editor.org/info/rfc7872>>.
- [RFC7928] Kuhn, N., Ed., Natarajan, P., Ed., Khademi, N., Ed., and D. Ros, "Characterization Guidelines for Active Queue Management (AQM)", RFC 7928, DOI 10.17487/RFC7928, July 2016, <<https://www.rfc-editor.org/info/rfc7928>>.
- [RFC8033] Pan, R., Natarajan, P., Baker, F., and G. White, "Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem", RFC 8033, DOI 10.17487/RFC8033, February 2017, <<https://www.rfc-editor.org/info/rfc8033>>.
- [RFC8084] Fairhurst, G., "Network Transport Circuit Breakers", BCP 208, RFC 8084, DOI 10.17487/RFC8084, March 2017, <<https://www.rfc-editor.org/info/rfc8084>>.
- [RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <<https://www.rfc-editor.org/info/rfc8085>>.

- [RFC8086] Yong, L., Ed., Crabbe, E., Xu, X., and T. Herbert, "GRE-in-UDP Encapsulation", RFC 8086, DOI 10.17487/RFC8086, March 2017, <<https://www.rfc-editor.org/info/rfc8086>>.
- [RFC8087] Fairhurst, G. and M. Welzl, "The Benefits of Using Explicit Congestion Notification (ECN)", RFC 8087, DOI 10.17487/RFC8087, March 2017, <<https://www.rfc-editor.org/info/rfc8087>>.
- [RFC8095] Fairhurst, G., Ed., Trammell, B., Ed., and M. Kuehlewind, Ed., "Services Provided by IETF Transport Protocols and Congestion Control Mechanisms", RFC 8095, DOI 10.17487/RFC8095, March 2017, <<https://www.rfc-editor.org/info/rfc8095>>.
- [RFC8250] Elkins, N., Hamilton, R., and M. Ackermann, "IPv6 Performance and Diagnostic Metrics (PDM) Destination Option", RFC 8250, DOI 10.17487/RFC8250, September 2017, <<https://www.rfc-editor.org/info/rfc8250>>.
- [RFC8257] Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <<https://www.rfc-editor.org/info/rfc8257>>.
- [RFC8289] Nichols, K., Jacobson, V., McGregor, A., Ed., and J. Iyengar, Ed., "Controlled Delay Active Queue Management", RFC 8289, DOI 10.17487/RFC8289, January 2018, <<https://www.rfc-editor.org/info/rfc8289>>.
- [RFC8290] Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm", RFC 8290, DOI 10.17487/RFC8290, January 2018, <<https://www.rfc-editor.org/info/rfc8290>>.
- [RFC8404] Moriarty, K., Ed. and A. Morton, Ed., "Effects of Pervasive Encryption on Operators", RFC 8404, DOI 10.17487/RFC8404, July 2018, <<https://www.rfc-editor.org/info/rfc8404>>.

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/>

Colin Perkins
University of Glasgow
School of Computing Science
Glasgow G12 8QQ
Scotland

EMail: csp@csperskins.org
URI: <https://csperskins.org/>

TSVWG Working Group
Internet-Draft
Intended status: Informational
Expires: April 17, 2020

L. Han
Y. Qu
L. Dong
R. Li
Futurewei Technologies
T. Nadeau
Lucid Vision
K. Smith
Vodafone
J. Tantsura
Apstra
October 15, 2019

Resource Reservation Protocol for IP Transport QoS
draft-han-tsvwg-ip-transport-qos-03

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on April 17, 2020.

Copyright Notice

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

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

Table of Contents

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

10.2. Informative References	26
Appendix A. Acknowledgements	28
Appendix B. Message Objects	29
B.1. Setup State Object	29
B.2. Bandwidth Object	31
B.3. Burst Msg	31
B.4. Latency Object	32
B.5. Authentication Object	32
B.6. OAM Object	33
B.7. Forwarding State Object	34
B.8. Setup State Report Object	34
B.9. Forward State Report Object	35
Authors' Addresses	36

1. Introduction

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

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

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

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

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

Traditionally, an IP network provides an unreliable or best-effort datagram service over a collection of underlying networks (i.e.:

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

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

1. The IP layer can only provide basic P2P (point to point) or P2MP (point to multi-point) end-to-end connectivity in the Internet, but the connectivity is not reliable and does not guarantee any quality of service to end-user or application, such as bandwidth, packet loss, latency etc. Due to this assumption, the transport layer or application must have its own control mechanism in congestion and flow to obtain the reliable and satisfactory service to cooperate with the under layer network quality.
2. The transport layer assumes that the IP layer can only process all IP flows equally in the hardware since the best effort service is actually an un-differentiated service. The process includes scheduling, queuing and forwarding. Thus, the transport layer must behave nicely and friendly to make sure all flows will only obtain its own faired share of resource, and no one could consume more and no one could be starved.

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

2. Terminology

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

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

Abbreviations used in this documents:

E2E

End-to-end.

EH

IPv6 Extension Header or Extension Option.

QoS

Quality of Service.

OAM

Operation and Management.

In-band Signaling

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

Out-of-band Signaling

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

IP flow

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

IP path

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

QoS channel

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

CIR

Committed Information Rate.

PIR

Peak Information Rate.

HbH-EH

IPv6 Hop-by-Hop Extension Header.

Dst-EH

IPv6 Destination Extension Header.

HbH-EH-aware node

Network nodes that are configured to process the IPv6 Hop-by-Hop Extension Header.

3. Overview

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

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

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

3.1. Design Targets

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

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

3.2. Scope and Assumptions

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

1. The new service with QoS is aimed to be supplementary to regular IP service. It is targeted for the applications that are bandwidth and/or latency sensitive. It is not intended to replace the TCP/IP variants that have been proved to be efficient and successful for current applications.
2. The new service is limited within one administrative domain, even it does not exclude the possibilities of extending the mechanism for inter-domain scenarios. Currently only inter-domain security is considered, and the inter-domain SLA, accounting and other issues are not discussed.
3. Due to high bandwidth requirement of new service for individual flow, the total number of the flows with the new service cannot be high for a port, or a system. From another point of view, the new service is targeted for applications that really need it, the number of supported applications/users should be controlled and cannot be unlimited. Hence the scalability requirement for the new service is limited.
4. The new service must be able to coexist with the regular transport service in the same hardware, and be backward compatible. Also, a transport flow can switch between regular transport and new service without service interruption.

3.3. Sub-layer in IP for Transport Control

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

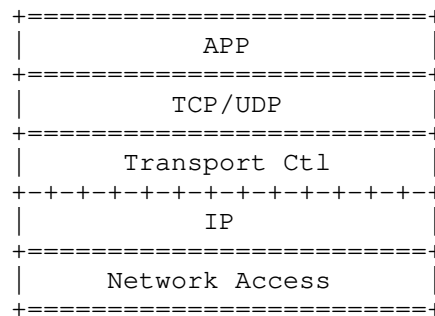


Figure 1: The new stack with a sub-layer in Layer 3

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

In-band Signaling

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

Congestion control

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

IP Path OAM

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

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

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

3.4. IP In-band signaling

In-band signaling messages are carried along with the payload. It is guaranteed that the signaling follows the same path as the data flow, and this can bring up some advantages that other methods can hardly provide:

Diagnosis

The in-band signaling message takes the same path, same hops, same processing at each hop as the data packet, this will make the diagnosis for both signaling and data path easier.

Simplicity

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

Performance and scalability

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

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

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

The major differences from the previous works are:

1. Focus on IPv6 only.
2. The proposed solution could be driven by end-user operating system's protocol stack such as TCP, UDP or other protocols, or by network device working as a proxy.
3. Simplified signaling process with minimal information carried, reduced QoS state maintenance at network devices.
4. Use different IPv6 options for signaling and signaling state report.
5. Support both bandwidth reservation and latency expectation at each hop.
6. Support dynamic resource reservation.
7. Support dynamic QoS forwarding state monitoring.

3.5. IPv6 Approach

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

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

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

The following figure illustrates the path setup process:

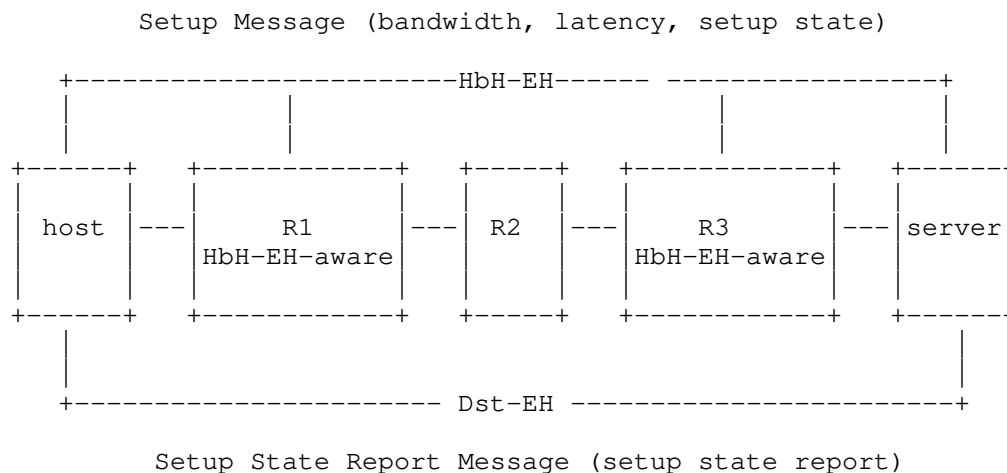


Figure 2: Path Setup

Using the figure.2 for illustration, to set up a path with resource reservation, a setup message including QoS requirements, such as max/min bandwidth, burst size, the latency, and the setup state is sent from the host to the server. After each HbH-EH-aware node along the path receives the message, it reads the QoS information and programs the hardware for resource reservation, queuing management etc. The setup state object is updated at each HbH-EH-aware node to include

the QoS programming and provisioning result and the necessary hardware reference information for IP forwarding with QoS. After the setup message reaches the server, the server will send a setup state report message encoded as Dst-EH to the host. The setup state report message carries the path setup results from the setup state object.

4. Key Messages and Parameters

4.1. Setup and Setup State Report messages

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

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

- o Reserved bandwidth is required: The throttle router is the critical point to be configured to process the hop-by-hop EH for the bandwidth reservation. Moreover, the direction of congestion must be considered.
- o Bounded latency is required: In theory, each router and switch could contribute some delay to the end-to-end latency, but the throttle router will contribute more than non-throttle routers, and slow device will contribute more than fast device. We can use OAM to detect the latency contribution in a network, and configure those worst-case devices to process the HbH-EH.

Setup State Report message is the message sent from the destination host to the source host (from the point of view of the Setup message). The message is embedded into the Dst-EH in any data packet. The Setup State Report in the message is just a copy from the Setup message received at the destination host for a typical TCP session. The message is used at the source host to forward the packet later and to do the congestion control.

```

<Setup Message> ::= <Setup State Object> [ <Bandwidth
Object> ]
                        [ <Burst Object> ] [ <Latency
Object> ]
                        [ <OAM Object> ] [ <Authentication
Object> ]
<Setup State Report Message> ::= <Setup State Report
Object>
                                [ <OAM Object> ]

```

4.2. Forwarding State and Forwarding State Report messages

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

1. Host directly send the IP packet without any changes to the packet, this is for the following cases:
 - * The hardware was programmed to use the tuples in IP header as identification for QoS process (SIS = 0), and
 - * The packet does not function to collect the QoS forwarding state on the path.
2. Host add the Forward State message into a data packet's IP header as HbH-EH and send the packet, this is for the cases:
 - * The hardware was programmed to use the Service ID as identification for QoS process (SIS != 0).
 - * The hardware was programmed to use the tuples in IP header as identification for QoS process (SIS = 0), and the data packet functions to collect the QoS forwarding state on the path. This is the situation that host wants to detect the QoS forwarding state for the purpose of failure handling (See section 4.3).

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

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

```

<Forward State Message> ::= <Forward State Object> [
  <Latency Object> ]
                                [ <OAM Object> ] [ <Authentication
                                Object> ]
<Forward State Report Message> ::= <Forward State Report
Object>
                                [ <OAM Object> ]

```

4.3. Hop Number

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

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

4.4. Flow Identifying Method and Service ID

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

Flow level

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

Address level In-band Signaling

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

Transport level In-band Signaling

Packets share the same source, destination IP address, protocol number, but with different source or destination port number (non-IPSec) or different flow label (IPSec). This could be for the

aggregated TCP or UDP flows that started and terminated at the same IP addresses.

DiffServ level In-band Signaling

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

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

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

4.5. QoS State and life of Time

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

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

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

4.6. Authentication

The in-band signaling is designed to have a basic security mechanism to protect the integrity of a signaling message. The Authentication message is to attach to a signaling message, the source host

calculates the hash value of a key and all invariable part of a signaling message (Setup message: ver, FI, R, SIS, P, Time; Bandwidth message, Latency message, Burst message). The key is only known to the hosts and all HbH-EH-aware nodes. The secure distribution of the key is out the scope of the document.

5. Packet Forwarding

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

5.1. Basic Hardware Capability

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

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

The following capabilities are RECOMMENDED:

1. The in-band signaling is processed in network processor without punting to controller CPU for help
2. The QoS forwarding state is kept and maintained in network processor without the involvement from controller CPU.
3. The QoS state has a life of a pre-configured time and will be automatically deleted if there is no data packet processed by that QoS state. The timer can be changed on the fly.
4. The data forwarding does not need to be done at the controller CPU, or so called slow path. It is at the same hardware as the normal IP forwarding. For any IP packet, the QoS forwarding is executed first. Normal forwarding will be executed if there is no QoS state associated with the identification of the flow.
5. The QoS forwarding and normal forwarding can be switched on the fly.

The details of data plane and hardware related implementations, such as traffic classification, shaping, queuing and scheduling, are out of scope of this document. The report of [NGP] has given some experiments and results by using commercial hardwares.

5.2. Flow Identification in Packet Forwarding

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

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

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

5.3. QoS Forwarding State Detection and Failure Handling

QoS forwarding may fail due to different reasons:

1. Hardware failure in HbH-EH-aware node.
2. IP path change due to link failure, node failure or routing changes; And the IP path change has impact to the HbH-EH-aware node.
3. Network topology change; and the change leads to the changes of HbH-EH-aware nodes.

Application may need to be aware of the service status of QoS guarantee when the application is using a TCP session with QoS. In order to provide such feature, the TCP stack in the source host can detect the QoS forwarding state by sending TCP data packet with Forwarding State message coded as HbH-EH. After the TCP data packet

reaches the destination host, the host will copy the forwarding state into a Forwarding State Report message, and send it with another TCP packet (for example, TCP-ACK) in reverse direction to the source host. Thereafter, the source host can obtain the QoS forwarding state on all HbH-EH-aware nodes.

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

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

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

6. Details of Working with Transport Layer

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

6.1. Working with TCP

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

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

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

- o Setup State Report: This is the state report of a setup message.
- o Forwarding State: This is the forwarding state message used for data packet.
- o Forwarding State Report: This is the forwarding state report of a QoS channel.

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

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

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

2. Downstream: This is for the direction of server to client. A application decides to open a TCP session with downstream QoS (for downloading), it will call TCP API to open a socket and connect to a server. The client host will form a TCP SYN packet with the Dst-EH in the IPv6 header. The EH includes Bandwidth

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

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

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

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

Based on this characteristics, application can request a relatively small CIR for the time of P-frame and P-frame, and request a big CIR for the time of I-frame.

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

6.2. Working with UDP and other Protocols

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

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

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

1. Modify the protocol itself to have some state machine to establish the closed-loop for the protocol. This can be done in the kernel of the OS by modifying the protocol stack.
2. Modify the user data stream to introduce the closed-loop scheme, this becomes as application work. It is up to application to add or modify codes for the state machine of the closed-loop control.

7. Additional Considerations

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

7.1. User and Application driven

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

1. The detailed QoS parameters in signaling message are set by end user application. New socket option must be added, the option is

a place holder for QoS parameters (Setup, Bandwidth, etc.), Setup State Report and Forwarding State Report messages.

2. The Setup State Report and Forwarding State Report message received at host are processed by transport service in kernel. The Setup State Report message processed at host can result in the notification to the application whether the setup is successful. If the setup is successful, the application can start to use the socket having the QoS support; If the setup is failed, the application may have three choices:
 - * Lower the QoS requirement and re-setup a new QoS channel with new in-band signaling message.
 - * Use the TCP session as traditional transport without any QoS support.
 - * Lookup the service provider for help to locate the problem in network.

7.2. Traffic Management in Host

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

1. For current traffic management in a host, all TCP/UDP sessions will share the bandwidth for all egress links. For the purpose to work with the differentiated service provided by under layer network in bandwidth and latency, the kernel may allocate expected resource to applications that are using the QoS transport service. For example, kernel can queue different packets from different applications or users to different queue and schedule them in different priority. Only after this change, some application can use more bandwidth and get less queuing delay for a link than others.
2. The congestion control in a host manages the behavior of TCP flow(s). This includes important features like slow start, AIMD, fast retransmit, selective ACK, etc. To accommodate the benefit of the QoS guaranteed transport service, the congestion control can be much simpler [I-D.han-tsvwg-cc]. The new congestion control is related to the implementation of QoS guarantee. Following is a simple congestion control algorithm assuming that the CIR is guaranteed and PIR is shared between flows:

- * There is no slow start, the TCP can start sending traffic at the rate of CIR.
- * The AIMD is kept, but the range of the sawtooth pattern should be maintained between CIR and PIR.
- * Other congestion control features can be kept.

7.3. Heterogeneous Network

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

7.4. Proxy Control

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

1. A proxy can check if the in-band signaling from an end user meets the SLA compliance. This adds extra security and DOS attack prevention.
2. A proxy can collect the statistics for user's TCP flows and check the in-band signaling for accounting and charging.
3. A proxy can insert and process appropriate in-band signaling for TCP flows if the host does not support this new feature. This can provide backward compatibility, also enable the host to use the new feature.

8. IANA Considerations

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

Hex Value	Binary Value			Description	Reference
	act	chg	rest		
0x0	00	0	10000	In-band Signaling	Section 4 in this doc

Figure 3: The New Option Type

1. The highest-order 2 bits: 00, indicating if the processing IPv6 node does not recognize the Option type, skip over this option and continue processing the header.
2. The third-highest-order bit: 0, indicating the Option Data does not change en route.
3. The low-order 5 bits: 10000, assigned by IANA.

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

Type	Mnemonic	Description	Reference
0	SETUP	Setup object	Appendix B
1	BANDWIDTH	Bandwidth object	Appendix B
2	BURST	Burst object	Appendix B
3	LATENCY	Latency object	Appendix B
4	AUTH	Authentication object	Appendix B
5	OAM	OAM object	Appendix B
6	FWD STATE	Forward state	Appendix B
7	SETUP REPORT	Setup state report	Appendix B
8	FWD REPORT	Forwarding state report	Appendix B

Figure 4: The In-band Signaling Sub Type

9. Security Considerations

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

Authentication of user

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

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

Proxy

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

Authentication of signaling message

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

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

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2581] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion Control", RFC 2581, DOI 10.17487/RFC2581, April 1999, <<https://www.rfc-editor.org/info/rfc2581>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <<https://www.rfc-editor.org/info/rfc8200>>.

10.2. Informative References

- [eNodeB] wikipedia, "eNodeB", 2018,
 <<https://en.wikipedia.org/wiki/ENodeB>>.
- [EPC] 3GPP, "The Evolved Packet Core", 2018,
 <<http://www.3gpp.org/technologies/keywords-acronyms/100-the-evolved-packet-core>>.
- [HU5G] Huawei, "5G Vision: 100 Billion connections, 1 ms Latency, and 10 Gbps Throughput", 2015,
 <<http://www.huawei.com/minisite/5g/en/defining-5g.html>>.
- [I-D.falk-xcp-spec]
 Falk, A., "Specification for the Explicit Control Protocol (XCP)", draft-falk-xcp-spec-03 (work in progress), July 2007.
- [I-D.han-iccrgr-arvr-transport-problem]
 Han, L. and K. Smith, "Problem Statement: Transport Support for Augmented and Virtual Reality Applications", draft-han-iccrgr-arvr-transport-problem-01 (work in progress), March 2017.
- [I-D.han-tsvwg-cc]
 Han, L., Qu, Y., and T. Nadeau, "A New Congestion Control in Bandwidth Guaranteed Network", draft-han-tsvwg-cc-00 (work in progress), March 2018.
- [I-D.harper-inband-signalling-requirements]
 Harper, J., "Requirements for In-Band QoS Signalling", draft-harper-inband-signalling-requirements-00 (work in progress), January 2007.
- [I-D.ietf-aqm-codel]
 Nichols, K., Jacobson, V., McGregor, A., and J. Iyengar, "Controlled Delay Active Queue Management", draft-ietf-aqm-codel-06 (work in progress), December 2016.
- [I-D.ietf-aqm-fq-codel]
 Hoeiland-Joergensen, T., McKeeney, P., dave.taht@gmail.com, d., Gettys, J., and E. Dumazet, "The FlowQueue-CoDel Packet Scheduler and Active Queue Management Algorithm", draft-ietf-aqm-fq-codel-06 (work in progress), March 2016.

- [I-D.ietf-aqm-pie]
Pan, R., Natarajan, P., Baker, F., and G. White, "PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem", draft-ietf-aqm-pie-10 (work in progress), September 2016.
- [I-D.ietf-detnet-use-cases]
Grossman, E., "Deterministic Networking Use Cases", draft-ietf-detnet-use-cases-20 (work in progress), December 2018.
- [I-D.ietf-spring-segment-routing]
Filsfils, C., Previdi, S., Ginsberg, L., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", draft-ietf-spring-segment-routing-15 (work in progress), January 2018.
- [I-D.ietf-tcpm-dctcp]
Bensley, S., Eggert, L., Thaler, D., Balasubramanian, P., and G. Judd, "Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters", draft-ietf-tcpm-dctcp-03 (work in progress), November 2016.
- [I-D.roberts-inband-qos-ipv6]
Roberts, L. and J. Harford, "In-Band QoS Signaling for IPv6", draft-roberts-inband-qos-ipv6-00 (work in progress), July 2005.
- [I-D.sridharan-tcpm-ctcp]
Sridharan, M., Tan, K., Bansal, D., and D. Thaler, "Compound TCP: A New TCP Congestion Control for High-Speed and Long Distance Networks", draft-sridharan-tcpm-ctcp-02 (work in progress), November 2008.
- [IPv6_Parameters]
IANA, "Internet Protocol Version 6 (IPv6) Parameters", 2015, <<https://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml#ipv6-parameters-2>>.
- [MEC]
ETSI, "Multi-access Edge Computing", 2018, <<https://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing>>.
- [NGP]
ETSI, "Next Generation Protocols (NGP); Recommendation for New Transport Technologies", 2018, <https://www.etsi.org/deliver/etsi_gr/NGP/001_099/010/01.01.01_60/gr_NGP010v010101p.pdf>.

- [QU2016] Qualcomm, "Leading the world to 5G", 2016, <<https://www.qualcomm.com/media/documents/files/qualcomm-5g-vision-presentation.pdf>>.
- [RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <<https://www.rfc-editor.org/info/rfc2474>>.
- [RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, DOI 10.17487/RFC2475, December 1998, <<https://www.rfc-editor.org/info/rfc2475>>.
- [RFC3175] Baker, F., Iturralde, C., Le Faucheur, F., and B. Davie, "Aggregation of RSVP for IPv4 and IPv6 Reservations", RFC 3175, DOI 10.17487/RFC3175, September 2001, <<https://www.rfc-editor.org/info/rfc3175>>.
- [Tactile] JDavid Szabo, et al. Proceedings of European Wireless 2015; 21th European Wireless Conference, "Towards the Tactile Internet: Decreasing Communication Latency with Network Coding and Software Defined Networking", 2015, <<http://fastpass.mit.edu/Fastpass-SIGCOMM14-Perry.pdf>>.
- [TCP-vegas] Peterson, L., "TCP Vegas: New Techniques for Congestion Detection and Avoidance - CiteSeer page on the 1994 SIGCOMM paper", 1994.
- [TCP_Targets] Andreas Benthin, Stefan Mischke, University of Paderborn, "Bandwidth Allocation of TCP", 2004.
- [TIA] TIA 1039 Revision A, "QoS Signaling for IP QoS Support and Sender Authentication", 2015, <https://global.ihs.com/doc_detail.cfm?&csf=TIA&item_s_key=00480715&item_key_date=880431>.

Appendix A. Acknowledgements

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

We appreciate the following people who made lots of contributions to this draft: Guoping Li, Boyan Tu, and Xuefei Tan, and thank Huawei Nanjing research team led by Feng Li to provide the Product on

Concept (POC) development and test, the team members include Fengxin Sun, Xingwang Zhou, and Weiguang Wang. We also like to thank other people involved in the discussion of solution: Tao Ma from Future Network Strategy dept.

Appendix B. Message Objects

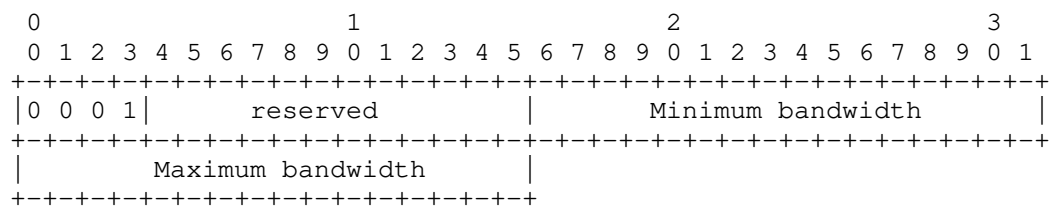
This section defines detailed objects used in different messages.

B.1. Setup State Object

Service ID list for hops: it is for all hops on the path, each service ID bit size is defined in SIS. The 1st service ID is at the top of the stack. Each hop add its service ID at the correct position indexed by the current hop number for the router.

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

B.2. Bandwidth Object



Type = 1,

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

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

Figure 6: The Bandwidth Object

B.3. Burst Msg

```

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

```

Type = 2,

Burst size : The burst size, unit M bytes

Figure 7: The burst message

B.4. Latency Object

```

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

```

Type = 3,

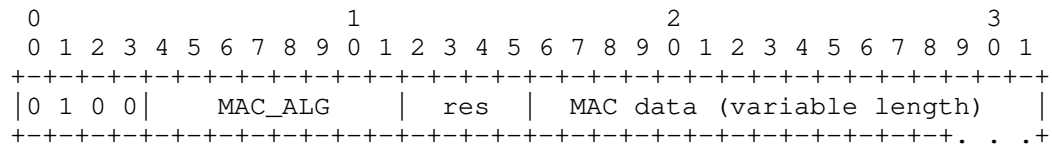
u: the unit of the latency

0: ms; 1: us

Latency: Expected maximum latency for each hop

Figure 8: The Latency Object

B.5. Authentication Object



Type = 4,

MAC_ALG: Message Authentication Algorithm

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

MAC data: Message Authentication Data;

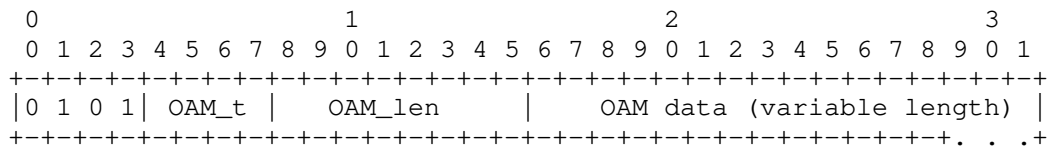
Res: Reserved bits

Size of signaling data (opt_len): Size of MAC data + 2

MD5: 18; SHA-0: 22; SHA-1: 22; SHA-256: 34; SHA-512: 66

Figure 9: The Authentication Object

B.6. OAM Object



Type = 5,

OAM_t : OAM type

OAM_len : 8-bit unsigned integer. Length of the OAM data, in octets;

OAM data: OAM data, details of OAM data are TBD.

Figure 10: The OAM Object

B.7. Forwarding State Object

```

      0               1               2               3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
| 0 1 1 0 | ver | FI | R | SIS | P | Time | Hop_num | u | Total_latency |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Forwarding state for each hop index |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Service ID list for hops |
+-----+-----+-----+-----+-----+-----+-----+-----+
. . .

```

Type = 6, Forwarding state;

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

Forward state for each hop index : each bit is the fwd state on each

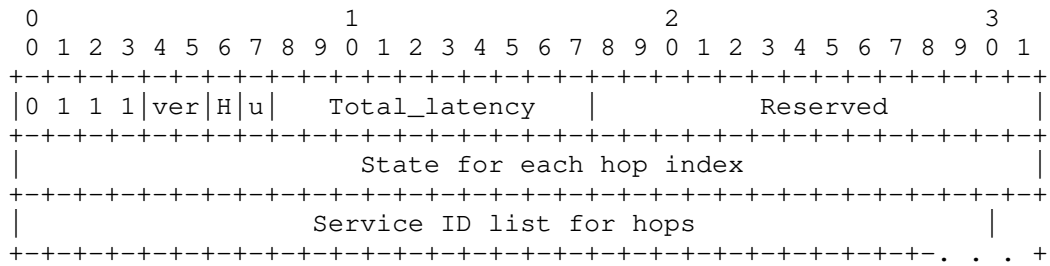
hop on the path, 0: failed; 1: success; The 1st hop is at the most significant bit.

Service ID list for hops: it is for all hops on the path, each index bit

size is defined in SIS. The list is from the setup report message.

Figure 11: The Forwarding State Object

B.8. Setup State Report Object



Type = 7, Setup state report;

H: Hop number bit. When a host receives a setup message and form a setup report message, it must check if the Hop_num in setup message is zero. If it is zero, the H bit is set to one, and if it is not zero, the H bit is clear. This will notify the source of setup message that if the original Hop_num was correct.

Following are directly copied from the setup message:

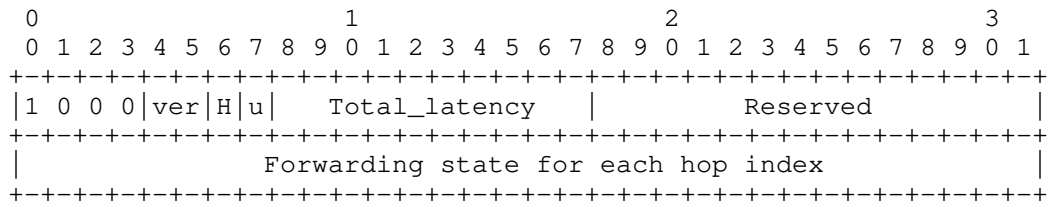
u, Total_latency;

State for each hop index

Service ID list for hops.

Figure 12: The Setup State Report Object

B.9. Forward State Report Object



Type = 8, Forwarding state report;

H: Hop number bit. When a host receives a Forward State message and form a Forward State Report message, it must check if the Hop_num in Forward State message is zero. If it is zero, the H bit is set to one, and if it is not zero, the H bit is clear.

This will notify the source of Forward State message that if the original Hop_num was set correct.

Following are directly copied from the Forward State message:

u, Total_latency;

Forwarding State for each hop index

Figure 13: The Fwd State Report Object

Authors' Addresses

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

Phone: +10 408 330 4613
 Email: lin.han@futurewei.com

Yingzhen Qu
Futurewei Technologies
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: yingzhen.qu@futurewei.com

Lijun Dong
Futurewei Technologies
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: lijun.dong@futurewei.com

Richard Li
Futurewei Technologies
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: renwei.li@futurewei.com

Thomas Nadeau
Lucid Vision
Hampton NH 03842
USA

Email: tnadeau@lucidvision.com

Kevin Smith
Vodafone
UK

Email: Kevin.Smith@vodafone.com

Jeff Tantsura
Apstra

Email: jefftant.ietf@gmail.com

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

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

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

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on 5 November 2022.

Copyright Notice

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

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

Table of Contents

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

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

1. Introduction

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

1.1. Outline of the Problem

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

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

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

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

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

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

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

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

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

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

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

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

1.2. Scope

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

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

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

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

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

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

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

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

Latency is not the only concern of L4S:

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

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

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

1.3. Terminology

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.4. Features

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

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

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

In all these experiments, the host was connected to the home network by fixed Ethernet, in order to quantify the queuing delay that can be achieved by a user who cares about delay. It should be emphasized that L4S support at the bottleneck link cannot 'undelay' bursts introduced by another link on the path, for instance by legacy WiFi equipment. However, if L4S support is added to the queue feeding the `_outgoing_` WAN link of a home gateway, it would be counterproductive not to also reduce the burstiness of the `_incoming_` WiFi. Also, trials of WiFi equipment with an L4S DualQ Coupled AQM on the `_outgoing_` WiFi interface are in progress, and early results of an L4S DualQ Coupled AQM in a 5G radio access network testbed with emulated outdoor cell edge radio fading are given in [L4S_5G].

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

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

The two queues are only necessary because:

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

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

2. DualQ Coupled AQM

There are two main aspects to the approach:

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

2.1. Coupled AQM

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

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

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

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

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

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

Stating this as a formula, the relation between Classic drop probability, `p_C`, and the coupled L4S probability `p_CL` needs to take the form:

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

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

2.2. Dual Queue

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

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

2.3. Traffic Classification

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

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

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

2.4. Overall DualQ Coupled AQM Structure

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

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

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

Substituting for p_C in Eqn (1) gives:

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

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

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

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

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

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

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

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

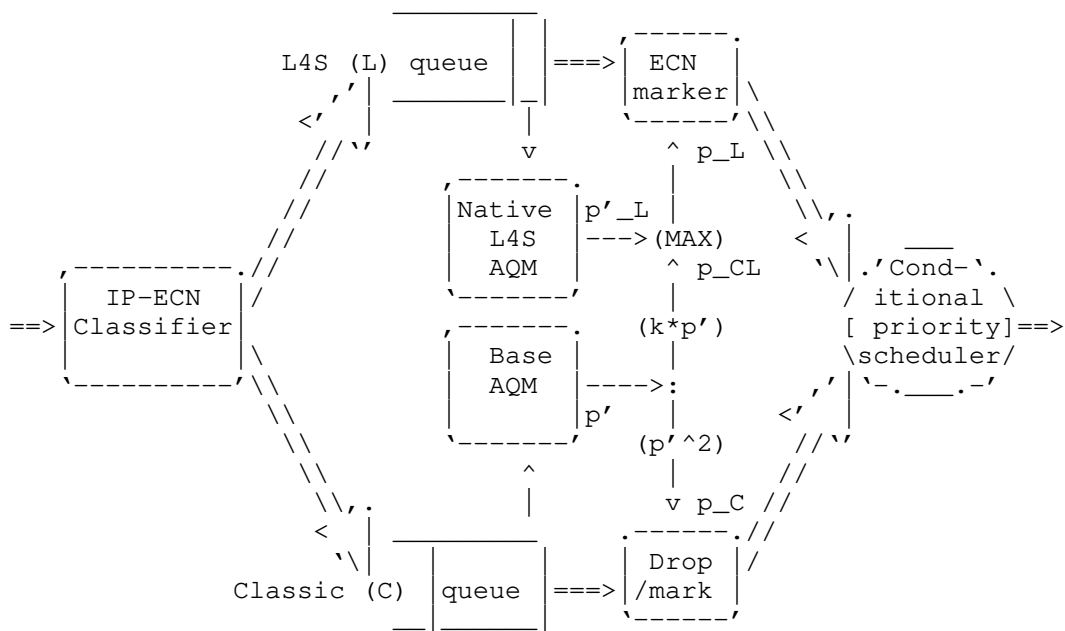


Figure 1: DualQ Coupled AQM Schematic

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

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

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

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

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

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

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

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

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

2.5. Normative Requirements for a DualQ Coupled AQM

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

2.5.1. Functional Requirements

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

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

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

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

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

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

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

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

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

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

2.5.1.1. Requirements in Unexpected Cases

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

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

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

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

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

2.5.2. Management Requirements

2.5.2.1. Configuration

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

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

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

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

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

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

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

2.5.2.2. Monitoring

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

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

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

2.5.2.3. Anomaly Detection

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

- * Start-time and duration of overload state.

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

2.5.2.4. Deployment, Coexistence and Scaling

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

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

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

This specification contains no IANA considerations.

4. Security Considerations

4.1. Low Delay without Requiring Per-Flow Processing

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

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

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

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

4.2. Handling Unresponsive Flows and Overload

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

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

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

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

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

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

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

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

4.2.1. Unresponsive Traffic without Overload

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

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

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

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

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

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

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

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

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

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

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

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

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

4.2.3. L4S ECN Saturation: Introduce Drop or Delay?

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

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

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

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

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

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

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

4.2.3.1. Protecting against Overload by Unresponsive ECN-Capable Traffic

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

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

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

5. Acknowledgements

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

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

6. Contributors

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

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

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

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

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

7. References

7.1. Normative References

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

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

7.2. Informative References

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

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

- [I-D.briscoe-iccrp-prague-congestion-control]
Schepper, K. D., Tilmans, O., and B. Briscoe, "Prague Congestion Control", Work in Progress, Internet-Draft, draft-briscoe-iccrp-prague-congestion-control-00, 9 March 2021, <<https://datatracker.ietf.org/doc/html/draft-briscoe-iccrp-prague-congestion-control-00>>.
- [I-D.briscoe-tsvwg-l4s-diffserv]
Briscoe, B., "Interactions between Low Latency, Low Loss, Scalable Throughput (L4S) and Differentiated Services", Work in Progress, Internet-Draft, draft-briscoe-tsvwg-l4s-diffserv-02, 4 November 2018, <<https://datatracker.ietf.org/doc/html/draft-briscoe-tsvwg-l4s-diffserv-02>>.
- [I-D.cardwell-iccrp-bbr-congestion-control]
Cardwell, N., Cheng, Y., Yeganeh, S. H., Swett, I., and V. Jacobson, "BBR Congestion Control", Work in Progress, Internet-Draft, draft-cardwell-iccrp-bbr-congestion-control-02, 7 March 2022, <<https://datatracker.ietf.org/doc/html/draft-cardwell-iccrp-bbr-congestion-control-02>>.
- [I-D.ietf-tsvwg-l4s-arch]
Briscoe, B., Schepper, K. D., Bagnulo, M., and G. White, "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture", Work in Progress, Internet-Draft, draft-ietf-tsvwg-l4s-arch-17, 4 March 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-l4s-arch-17>>.
- [L4Sdemo16]
Bondarenko, O., De Schepper, K., Tsang, I., and B. Briscoe, "Ultra-Low Delay for All: Live Experience, Live Analysis", Proc. MMSYS'16 pp33:1--33:4, May 2016, <<http://dl.acm.org/citation.cfm?doid=2910017.2910633>> (videos of demos: <https://riteproject.eu/dctth/#1511dispatchwg>)>.
- [L4S_5G]
Willars, P., Wittenmark, E., Ronkainen, H., Östberg, C., Johansson, I., Strand, J., Lédl, P., and D. Schnieders, "Enabling time-critical applications over 5G with rate adaptation", Ericsson - Deutsche Telekom White Paper BNEW-21:025455 Uen, May 2021, <<https://www.ericsson.com/en/reports-and-papers/white-papers/enabling-time-critical-applications-over-5g-with-rate-adaptation>>.

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

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

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

Appendix A. Example DualQ Coupled PI2 Algorithm

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

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

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

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

A.1. Pass #1: Core Concepts

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

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

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

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

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

```

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

Figure 2: Example Header Pseudocode for DualQ Coupled PI2 AQM

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

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

```

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

```

Figure 3: Example Enqueue Pseudocode for DualQ Coupled PI2 AQM

```

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

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

```

Figure 4: Example Dequeue Pseudocode for DualQ Coupled PI2 AQM

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

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

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

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

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

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

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

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

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

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

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

```

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

```

Figure 5: Example Pseudocode for the Native L4S AQM

```

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

```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Because the coupled L4S marking probability (p_{CL}) is factored up by k , the dynamic gain parameters alpha and beta are also inherently factored up by k for the L4S queue. So, the effective gain factor for the L4S queue is $k * \alpha$ (with defaults $\alpha = 0.16 \text{ Hz}$ and $k=2$, effective L4S $\alpha = 0.32 \text{ Hz}$).

Unlike in PIE [RFC8033], alpha and beta do not need to be tuned every Tupdate dependent on p' . Instead, in PI2, alpha and beta are independent of p' because the squaring applied to Classic traffic tunes them inherently. This is explained in [PI2], which also explains why this more principled approach removes the need for most of the heuristics that had to be added to PIE.

Nonetheless, an implementer might wish to add selected details to either AQM. For instance the Linux reference DualPI2 implementation includes the following (not shown in the pseudocode above):

- * Classic and coupled marking or dropping (i.e. based on p_C and p_CL from the PI controller) is not applied to a packet if the aggregate queue length in bytes is < 2 MTU (prior to enqueueing the packet or dequeuing it, depending on whether the AQM is configured to be applied at enqueue or dequeue);
- * In the WRR scheduler, the 'credit' indicating which queue should transmit is only changed if there are packets in both queues (i.e. if there is actual resource contention). This means that a properly paced L flow might never be delayed by the WRR. The WRR credit is reset in favour of the L queue when the link is idle.

An implementer might also wish to add other heuristics, e.g. burst protection [RFC8033] or enhanced burst protection [RFC8034].

Notes:

- a. The drain rate of the queue can vary if it is scheduled relative to other queues, or to cater for fluctuations in a wireless medium. To auto-adjust to changes in drain rate, the queue needs to be measured in time, not bytes or packets [AQMetrics], [CoDel]. Queuing delay could be measured directly by storing a per-packet time-stamp as each packet is enqueued, and subtracting this from the system time when the packet is dequeued. If time-stamping is not easy to introduce with certain hardware, queuing delay could be predicted indirectly by dividing the size of the queue by the predicted departure rate, which might be known precisely for some link technologies (see for example in DOCSIS PIE [RFC8034]).
- b. Line 2 of the `dualpi2_enqueue()` function (Figure 3) assumes an implementation where lq and cq share common buffer memory. An alternative implementation could use separate buffers for each queue, in which case the arriving packet would have to be classified first to determine which buffer to check for available space. The choice is a trade off; a shared buffer can use less memory whereas separate buffers isolate the L4S queue from tail-drop due to large bursts of Classic traffic (e.g. a Classic Reno TCP during slow-start over a long RTT).
- c. There has been some concern that using the step function of DCTCP for the Native L4S AQM requires end-systems to smooth the signal for an unnecessarily large number of round trips to ensure sufficient fidelity. A ramp is no worse than a step in initial experiments with existing DCTCP. Therefore, it is recommended that a ramp is configured in place of a step, which will allow congestion control algorithms to investigate faster smoothing algorithms.

A ramp is more general than a step, because an operator can effectively turn the ramp into a step function, as used by DCTCP, by setting the range to zero. There will not be a divide by zero problem at line 5 of Figure 5 because, if minTh is equal to maxTh, the condition for this ramp calculation cannot arise.

A.2. Pass #2: Edge-Case Details

This section takes a second pass through the pseudocode adding details of two edge-cases: low link rate and overload. Figure 7 repeats the dequeue function of Figure 4, but with details of both edge-cases added. Similarly Figure 8 repeats the core PI algorithm of Figure 6, but with overload details added. The initialization, enqueue, L4S AQM and recur functions are unchanged.

The link rate can be so low that it takes a single packet queue longer to serialize than the threshold delay at which ECN marking starts to be applied in the L queue. Therefore, a minimum marking threshold parameter in units of packets rather than time is necessary (Th_len, default 1 packet in line 19 of Figure 2) to ensure that the ramp does not trigger excessive marking on slow links. Where an implementation knows the link rate, it can set up this minimum at the time it is configured. For instance, it would divide 1 MTU by the link rate to convert it into a serialization time, then if the lower threshold of the Native L AQM ramp was lower than this serialization time, it could increase the thresholds to shift the bottom of the ramp to 2 MTU. This is the approach used in DOCSIS [DOCSIS3.1], because the configured link rate is dedicated to the DualQ.

The pseudocode given here applies where the link rate is unknown, which is more common for software implementations that might be deployed in scenarios where the link is shared with other queues. In lines 5a to 5d in Figure 7 the native L4S marking probability, p'_L , is zeroed if the queue is only 1 packet (in the default configuration).

Linux implementation note:

- * In Linux, the check that the queue exceeds Th_len before marking with the native L4S AQM is actually at enqueue, not dequeue, otherwise it would exempt the last packet of a burst from being marked. The result of the check is conveyed from enqueue to the dequeue function via a boolean in the packet metadata.

Persistent overload is deemed to have occurred when Classic drop/marking probability reaches p_{Cmax} . Above this point, the Classic drop probability is applied to both L and C queues, irrespective of whether any packet is ECN-capable. ECT packets that are not dropped can still be ECN-marked.

In line 10 of the initialization function (Figure 2), the maximum Classic drop probability $p_{Cmax} = \min(1/k^2, 1)$ or $1/4$ for the default coupling factor $k=2$. In practice, 25% has been found to be a good threshold to preserve fairness between ECN capable and non ECN capable traffic. This protects the queues against both temporary overload from responsive flows and more persistent overload from any unresponsive traffic that falsely claims to be responsive to ECN.

When the Classic ECN marking probability reaches the p_{Cmax} threshold ($1/k^2$), the marking probability coupled to the L4S queue, p_{CL} will always be 100% for any k (by equation (1) in Section 2). So, for readability, the constant p_{Lmax} is defined as 1 in line 22 of the initialization function (Figure 2). This is intended to ensure that the L4S queue starts to introduce dropping once ECN-marking saturates at 100% and can rise no further. The 'Prague L4S' requirements [I-D.ietf-tsvwg-ecn-l4s-id] state that, when an L4S congestion control detects a drop, it falls back to a response that coexists with 'Classic' Reno congestion control. So it is correct that, when the L4S queue drops packets, it drops them proportional to p'^2 , as if they are Classic packets.

The two queues each test for overload in lines 4b and 12b of the dequeue function (Figure 7). Lines 8c to 8g drop L4S packets with probability p'^2 . Lines 8h to 8i mark the remaining packets with probability p_{CL} . Given $p_{Lmax} = 1$, all remaining packets will be marked because, to have reached the else block at line 8b, $p_{CL} \geq 1$.

Line 2a in the core PI algorithm (Figure 8) deals with overload of the L4S queue when there is little or no Classic traffic. This is necessary, because the core PI algorithm maintains the appropriate drop probability to regulate overload, but it depends on the length of the Classic queue. If there is little or no Classic queue the naive PI update function in Figure 6 would drop nothing, even if the L4S queue were overloaded - so tail drop would have to take over (lines 2 and 3 of Figure 3).

Instead, line 2a of the full PI update function in Figure 8 ensures that the base PI AQM in line 3 is driven by whichever of the two queue delays is greater, but line 3 still always uses the same Classic target (default 15 ms). If L queue delay is greater just because there is little or no Classic traffic, normally it will still be well below the base AQM target. This is because L4S traffic is

also governed by the shallow threshold of its own native AQM (lines 5 and 6 of the dequeue algorithm in Figure 7). So the base AQM will be driven to zero and not contribute. However, if the L queue is overloaded by traffic that is unresponsive to its marking, the `max()` in line 2 enables the L queue to smoothly take over driving the base AQM into overload mode even if there is little or no Classic traffic. Then the base AQM will keep the L queue to the Classic target (default 15 ms) by shedding L packets.

```

1:  dualpi2_dequeue(lq, cq, pkt) {      % Couples L4S & Classic queues
2:    while ( lq.by() + cq.by() > 0 ) {
3:      if ( scheduler() == lq ) {
4a:        lq.dequeue(pkt)                % L4S scheduled
4b:        if ( p_CL < p_Lmax ) {          % Check for overload saturation
5a:          if (lq.len()>Th_len)          % >1 packet queued
5b:            p'_L = laqm(lq.time())      % Native LAQM
5c:          else
5d:            p'_L = 0                    % Suppress marking 1 pkt queue
6:          p_L = max(p'_L, p_CL)          % Combining function
7:          if ( recur(lq, p_L)            %Linear marking
8a:            mark(pkt)
8b:          } else {                      % overload saturation
8c:            if ( recur(lq, p_C) ) {      % probability p_C = p'^2
8e:              drop(pkt)                  % revert to Classic drop due to overload
8f:              continue                  % continue to the top of the while loop
8g:            }
8h:            if ( recur(lq, p_CL) )       % probability p_CL = k * p'
8i:              mark(pkt)                  % linear marking of remaining packets
8j:          }
9:        } else {
10:         cq.dequeue(pkt)                 % Classic scheduled
11:         if ( recur(cq, p_C) ) {          % probability p_C = p'^2
12a:          if ( (ecn(pkt) == 0)          % ECN field = not-ECT
12b:            OR (p_C >= p_Cmax) ) {      % Overload disables ECN
13:            drop(pkt)                    % squared drop, redo loop
14:            continue                    % continue to the top of the while loop
15:          }
16:          mark(pkt)                      % squared mark
17:        }
18:      }
19:      return(pkt)                        % return the packet and stop
20:    }
21:    return(NULL)                         % no packet to dequeue
22:  }

```

Figure 7: Example Dequeue Pseudocode for DualQ Coupled PI2 AQM
(Including Code for Edge-Cases)

```

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

```

Figure 8: Example PI-Update Pseudocode for DualQ Coupled PI2 AQM
(Including Overload Code)

The choice of scheduler technology is critical to overload protection (see Section 4.2.2).

- * A well-understood weighted scheduler such as weighted round robin (WRR) is recommended. As long as the scheduler weight for Classic is small (e.g. 1/16), its exact value is unimportant because it does not normally determine capacity shares. The weight is only important to prevent unresponsive L4S traffic starving Classic traffic in the short term (see Section 4.2.2). This is because capacity sharing between the queues is normally determined by the coupled congestion signal, which overrides the scheduler, by making L4S sources leave roughly equal per-flow capacity available for Classic flows.
- * Alternatively, a time-shifted FIFO (TS-FIFO) could be used. It works by selecting the head packet that has waited the longest, biased against the Classic traffic by a time-shift of *tshift*. To implement time-shifted FIFO, the scheduler() function in line 3 of the dequeue code would simply be implemented as the scheduler() function at the bottom of Figure 10 in Appendix B. For the public Internet a good value for *tshift* is 50ms. For private networks with smaller diameter, about 4*target would be reasonable. TS-FIFO is a very simple scheduler, but complexity might need to be added to address some deficiencies (which is why it is not recommended over WRR):
 - TS-FIFO does not fully isolate latency in the L4S queue from uncontrolled bursts in the Classic queue;
 - TS-FIFO is only appropriate if time-stamping of packets is feasible;
 - Even if time-stamping is supported, the sojourn time of the head packet is always stale. For instance, if a burst arrives at an empty queue, the sojourn time only fully measures the burst's delay when its last packet is dequeued, even though the

queue knew about the burst from the start - so it could have signalled congestion earlier. To remedy this, each head packet can be marked when it is dequeued based on the expected delay of the tail packet behind it, as explained below, rather than based on the head packet's own delay due to the packets in front of it. [Heist21] identifies a specific scenario where bursty traffic significantly hits utilization of the L queue. If this effect proves to be more widely applicable, it is believed that using the delay behind the head would improve performance.

The delay behind the head can be implemented by dividing the backlog at dequeue by the link rate or equivalently multiplying the backlog by the delay per unit of backlog. The implementation details will depend on whether the link rate is known; if it is not, a moving average of the delay per unit backlog can be maintained. This delay consists of serialization as well as media acquisition for shared media. So the details will depend strongly on the specific link technology. This approach should be less sensitive to timing errors and cost less in operations and memory than the otherwise equivalent 'scaled sojourn time' metric, which is the sojourn time of a packet scaled by the ratio of the queue sizes when the packet departed and arrived [SigQ-Dyn].

- * A strict priority scheduler would be inappropriate as discussed in Section 4.2.2.

Appendix B. Example DualQ Coupled Curvy RED Algorithm

As another example of a DualQ Coupled AQM algorithm, the pseudocode below gives the Curvy RED based algorithm. Although the AQM was designed to be efficient in integer arithmetic, to aid understanding it is first given using floating point arithmetic (Figure 10). Then, one possible optimization for integer arithmetic is given, also in pseudocode (Figure 11). To aid comparison, the line numbers are kept in step between the two by using letter suffixes where the longer code needs extra lines.

B.1. Curvy RED in Pseudocode

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

- * The initialization function `cred_params_init(...)` (Figure 2) that sets parameter defaults (the API for setting non-default values is omitted for brevity);

- * The dequeue function `cred_dequeue(lq, cq, pkt)` (Figure 4);
- * The scheduling function `scheduler()`, which selects between the head packets of the two queues.

It also uses the following functions that are either shown elsewhere, or not shown in full here:

- * The enqueue function, which is identical to that used for DualPI2, `dualpi2_enqueue(lq, cq, pkt)` in Figure 3;
- * `mark(pkt)` and `drop(pkt)` for ECN-marking and dropping a packet;
- * `cq.bytest()` or `lq.bytest()` returns the current length (aka. backlog) of the relevant queue in bytes;
- * `cq.time()` or `lq.time()` returns the current queuing delay (aka. sojourn time or service time) of the relevant queue in units of time (see Note a in Appendix A.1).

Because Curvy RED was evaluated before DualPI2, certain improvements introduced for DualPI2 were not evaluated for Curvy RED. In the pseudocode below, the straightforward improvements have been added on the assumption they will provide similar benefits, but that has not been proven experimentally. They are: i) a conditional priority scheduler instead of strict priority ii) a time-based threshold for the native L4S AQM; iii) ECN support for the Classic AQM. A recent evaluation has proved that a minimum ECN-marking threshold (`minTh`) greatly improves performance, so this is also included in the pseudocode.

Overload protection has not been added to the Curvy RED pseudocode below so as not to detract from the main features. It would be added in exactly the same way as in Appendix A.2 for the DualPI2 pseudocode. The native L4S AQM uses a step threshold, but a ramp like that described for DualPI2 could be used instead. The scheduler uses the simple TS-FIFO algorithm, but it could be replaced with WRR.

The Curvy RED algorithm has not been maintained or evaluated to the same degree as the DualPI2 algorithm. In initial experiments on broadband access links ranging from 4 Mb/s to 200 Mb/s with base RTTs from 5 ms to 100 ms, Curvy RED achieved good results with the default parameters in Figure 9.

The parameters are categorised by whether they relate to the Classic AQM, the L4S AQM or the framework coupling them together. Constants and variables derived from these parameters are also included at the end of each category. These are the raw input parameters for the

algorithm. A configuration front-end could accept more meaningful parameters (e.g. `RTT_max` and `RTT_typ`) and convert them into these raw parameters, as has been done for DualPI2 in Appendix A. Where necessary, parameters are explained further in the walk-through of the pseudocode below.

```

1: cred_params_init(...) {           % Set input parameter defaults
2:   % DualQ Coupled framework parameters
3:   limit = MAX_LINK_RATE * 250 ms   % Dual buffer size
4:   k' = 1                           % Coupling factor as a power of 2
5:   tshift = 50 ms                   % Time shift of TS-FIFO scheduler
6:   % Constants derived from Classic AQM parameters
7:   k = 2^k'                         % Coupling factor from Equation (1)
6:
7:   % Classic AQM parameters
8:   g_C = 5                          % EWMA smoothing parameter as a power of 1/2
9:   S_C = -1                         % Classic ramp scaling factor as a power of 2
10:  minTh = 500 ms                   % No Classic drop/mark below this queue delay
11:  % Constants derived from Classic AQM parameters
12:  gamma = 2^(-g_C)                 % EWMA smoothing parameter
13:  range_C = 2^S_C                  % Range of Classic ramp
14:
15:  % L4S AQM parameters
16:  T = 1 ms                         % Queue delay threshold for native L4S AQM
17:  % Constants derived from above parameters
18:  S_L = S_C - k'                   % L4S ramp scaling factor as a power of 2
19:  range_L = 2^S_L                  % Range of L4S ramp
20: }
```

Figure 9: Example Header Pseudocode for DualQ Coupled Curvy RED AQM

```

1: cred_dequeue(lq, cq, pkt) {           % Couples L4S & Classic queues
2:   while ( lq.bytt() + cq.bytt() > 0 ) {
3:     if ( scheduler() == lq ) {
4:       lq.dequeue(pkt)                  % L4S scheduled
5a:      p_CL = (Q_C - minTh) / range_L
5b:      if ( ( lq.time() > T )
5c:          OR ( p_CL > maxrand(U) ) )
6:        mark(pkt)
7:      } else {
8:        cq.dequeue(pkt)                  % Classic scheduled
9a:        Q_C = gamma * cq.time() + (1-gamma) * Q_C % Classic Q EWMA
10a:       sqrt_p_C = (Q_C - minTh) / range_C
10b:       if ( sqrt_p_C > maxrand(2*U) ) {
11:         if ( (ecn(pkt) == 0) ) {        % ECN field = not-ECT
12:           drop(pkt)                    % Squared drop, redo loop
13:           continue                    % continue to the top of the while loop
14:         }
15:         mark(pkt)
16:       }
17:     }
18:     return(pkt)                        % return the packet and stop here
19:   }
20:   return(NULL)                        % no packet to dequeue
21: }

22: maxrand(u) {                          % return the max of u random numbers
23:   maxr=0
24:   while (u-- > 0)
25:     maxr = max(maxr, rand())            % 0 <= rand() < 1
26:   return(maxr)
27: }

28: scheduler() {
29:   if ( lq.time() + tshift >= cq.time() )
30:     return lq;
31:   else
32:     return cq;
33: }

```

Figure 10: Example Dequeue Pseudocode for DualQ Coupled Curvy RED AQM

The dequeue pseudocode (Figure 10) is repeatedly called whenever the lower layer is ready to forward a packet. It schedules one packet for dequeuing (or zero if the queue is empty) then returns control to the caller, so that it does not block while that packet is being forwarded. While making this dequeue decision, it also makes the necessary AQM decisions on dropping or marking. The alternative of applying the AQMs at enqueue would shift some processing from the

critical time when each packet is dequeued. However, it would also add a whole queue of delay to the control signals, making the control loop very sloppy.

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

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

L4S: If the test at line 3 determines there is an L4S packet to dequeue, the tests at lines 5b and 5c determine whether to mark it. The first is a simple test of whether the L4S queue delay (`lq.time()`) is greater than a step threshold T (Note 3). The second test is similar to the random ECN marking in RED, but with the following differences: i) marking depends on queuing time, not bytes, in order to scale for any link rate without being reconfigured; ii) marking of the L4S queue depends on a logical OR of two tests; one against its own queuing time and one against the queuing time of the `_other_` (Classic) queue; iii) the tests are against the instantaneous queuing time of the L4S queue, but a smoothed average of the other (Classic) queue; iv) the queue is compared with the maximum of U random numbers (but if $U=1$, this is the same as the single random number used in RED).

Specifically, in line 5a the coupled marking probability p_{CL} is set to the amount by which the averaged Classic queueing delay Q_C exceeds the minimum queuing delay threshold (`minTh`) all divided by the L4S scaling parameter `range_L`. `range_L` represents the queuing delay (in seconds) added to `minTh` at which marking probability would hit 100%. Then in line 5c (if $U=1$) the result is compared with a uniformly distributed random number between 0 and 1, which ensures that, over `range_L`, marking probability will linearly increase with queueing time.

Classic: If the scheduler at line 3 chooses to dequeue a Classic packet and jumps to line 7, the test at line 10b determines whether to drop or mark it. But before that, line 9a updates Q_C , which is an exponentially weighted moving average (Note 4) of the queuing time of the Classic queue, where `cq.time()` is the current

instantaneous queueing time of the packet at the head of the Classic queue (zero if empty) and gamma is the EWMA constant (default 1/32, see line 12 of the initialization function).

Lines 10a and 10b implement the Classic AQM. In line 10a the averaged queueing time Q_C is divided by the Classic scaling parameter $range_C$, in the same way that queueing time was scaled for L4S marking. This scaled queueing time will be squared to compute Classic drop probability so, before it is squared, it is effectively the square root of the drop probability, hence it is given the variable name $sqrt_p_C$. The squaring is done by comparing it with the maximum out of two random numbers (assuming $U=1$). Comparing it with the maximum out of two is the same as the logical 'AND' of two tests, which ensures drop probability rises with the square of queueing time.

The AQM functions in each queue (lines 5c & 10b) are two cases of a new generalization of RED called Curvy RED, motivated as follows. When the performance of this AQM was compared with FQ-CoDel and PIE, their goal of holding queueing delay to a fixed target seemed misguided [CRED_Insights]. As the number of flows increases, if the AQM does not allow host congestion controllers to increase queueing delay, it has to introduce abnormally high levels of loss. Then loss rather than queueing becomes the dominant cause of delay for short flows, due to timeouts and tail losses.

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

From our limited experiments with Curvy RED so far, recommended values of these parameters are: $S_C = -1$; $g_C = 5$; $T = 5 * MTU$ at the link rate (about 1ms at 60Mb/s) for the range of base RTTs typical on the public Internet. [CRED_Insights] explains why these parameters are applicable whatever rate link this AQM implementation is deployed on and how the parameters would need to be adjusted for a scenario with a different range of RTTs (e.g. a data centre). The setting of k depends on policy (see Section 2.5 and Appendix C.2 respectively for its recommended setting and guidance on alternatives).

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

Notes:

1. The alternative of applying the AQMs at enqueue would shift some processing from the critical time when each packet is dequeued. However, it would also add a whole queue of delay to the control signals, making the control loop sloppier (for a typical RTT it would double the Classic queue's feedback delay). On a platform where packet timestamping is feasible, e.g. Linux, it is also easiest to apply the AQMs at dequeue because that is where queuing time is also measured.
2. WRR better isolates the L4S queue from large delay bursts in the Classic queue, but it is slightly less simple than TS-FIFO. If WRR were used, a low default Classic weight (e.g. 1/16) would need to be configured in place of the time shift in line 5 of the initialization function (Figure 9).
3. A step function is shown for simplicity. A ramp function (see Figure 5 and the discussion around it in Appendix A.1) is recommended, because it is more general than a step and has the potential to enable L4S congestion controls to converge more rapidly.
4. An EWMA is only one possible way to filter bursts; other more adaptive smoothing methods could be valid and it might be appropriate to decrease the EWMA faster than it increases, e.g. by using the minimum of the smoothed and instantaneous queue delays, `min(Q_C, qc.time())`.

B.2. Efficient Implementation of Curvy RED

Although code optimization depends on the platform, the following notes explain where the design of Curvy RED was particularly motivated by efficient implementation.

The Classic AQM at line 10b calls `maxrand(2*U)`, which gives twice as much curviness as the call to `maxrand(U)` in the marking function at line 5c. This is the trick that implements the square rule in equation (1) (Section 2.1). This is based on the fact that, given a number X from 1 to 6, the probability that two dice throws will both be less than X is the square of the probability that one throw will be less than X . So, when $U=1$, the L4S marking function is linear and the Classic dropping function is squared. If $U=2$, L4S would be a square function and Classic would be quartic. And so on.

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

```

1: cred_dequeue(lq, cq, pkt) {           % Couples L4S & Classic queues
2:   while ( lq.bytt() + cq.bytt() > 0 ) {
3:     if ( scheduler() == lq ) {
4:       lq.dequeue(pkt)                  % L4S scheduled
5:       if ((lq.time() > T) OR (Q_C >> (S_L-2) > maxrand(U)))
6:         mark(pkt)
7:     } else {
8:       cq.dequeue(pkt)                  % Classic scheduled
9:       Q_C += (qc.ns() - Q_C) >> g_C    % Classic Q EWMA
10:      if ( (Q_C >> (S_C-2) ) > maxrand(2*U) ) {
11:        if ( (ecn(pkt) == 0) ) {        % ECN field = not-ECT
12:          drop(pkt)                    % Squared drop, redo loop
13:          continue                    % continue to the top of the while loop
14:        }
15:        mark(pkt)
16:      }
17:    }
18:    return(pkt)                        % return the packet and stop here
19:  }
20:  return(NULL)                        % no packet to dequeue
21: }
```

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

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

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

Queuing delays are also scaled up by 2^{32} , but in two stages: i) In line 9 queuing time `qc.ns()` is returned in integer nanoseconds, making the value about 2^{30} times larger than when the units were seconds, ii) then in lines 5 and 10 an adjustment of -2 to the right bit-shift multiplies the result by 2^2 , to complete the scaling by 2^{32} .

In line 8 of the initialization function, the EWMA constant `gamma` is represented as an integer power of 2, `g_C`, so that in line 9 of the integer code the division needed to weight the moving average can be implemented by a right bit-shift (`>> g_C`).

Appendix C. Choice of Coupling Factor, k

C.1. RTT-Dependence

Where Classic flows compete for the same capacity, their relative flow rates depend not only on the congestion probability, but also on their end-to-end RTT (= base RTT + queue delay). The rates of Reno [RFC5681] flows competing over an AQM are roughly inversely proportional to their RTTs. Cubic exhibits similar RTT-dependence when in Reno-compatibility mode, but it is less RTT-dependent otherwise.

Until the early experiments with the DualQ Coupled AQM, the importance of the reasonably large Classic queue in mitigating RTT-dependence when the base RTT is low had not been appreciated. Appendix A.1.6 of the L4S ECN protocol [I-D.ietf-tsvwg-ecn-l4s-id] uses numerical examples to explain why bloated buffers had concealed the RTT-dependence of Classic congestion controls before that time. Then it explains why, the more that queuing delays have reduced, the more that RTT-dependence has surfaced as a potential starvation problem for long RTT flows, when competing against very short RTT flows.

Given that congestion control on end-systems is voluntary, there is no reason why it has to be voluntarily RTT-dependent. The RTT-dependence of existing Classic traffic cannot be 'undeployed'. Therefore, [I-D.ietf-tsvwg-ecn-l4s-id] requires L4S congestion controls to be significantly less RTT-dependent than the standard Reno congestion control [RFC5681], at least at low RTT. Then RTT-

dependence ought to be no worse than it is with appropriately sized Classic buffers. Following this approach means there is no need for network devices to address RTT-dependence, although there would be no harm if they did, which per-flow queuing inherently does.

C.2. Guidance on Controlling Throughput Equivalence

The coupling factor, k , determines the balance between L4S and Classic flow rates (see Section 2.5.2.1 and equation (1)).

For the public Internet, a coupling factor of $k=2$ is recommended, and justified below. For scenarios other than the public Internet, a good coupling factor can be derived by plugging the appropriate numbers into the same working.

To summarize the maths below, from equation (7) it can be seen that choosing $k=1.64$ would theoretically make L4S throughput roughly the same as Classic, if their actual end-to-end RTTs were the same. However, even if the base RTTs are the same, the actual RTTs are unlikely to be the same, because Classic traffic needs a fairly large queue to avoid under-utilization and excess drop. Whereas L4S does not.

Therefore, to determine the appropriate coupling factor policy, the operator needs to decide at what base RTT it wants L4S and Classic flows to have roughly equal throughput, once the effect of the additional Classic queue on Classic throughput has been taken into account. With this approach, a network operator can determine a good coupling factor without knowing the precise L4S algorithm for reducing RTT-dependence - or even in the absence of any algorithm.

The following additional terminology will be used, with appropriate subscripts:

r : Packet rate [pkt/s]

R : RTT [s/round]

p : ECN marking probability []

On the Classic side, we consider Reno as the most sensitive and therefore worst-case Classic congestion control. We will also consider Cubic in its Reno-friendly mode ('CReno'), as the most prevalent congestion control, according to the references and analysis in [PI2param]. In either case, the Classic packet rate in steady state is given by the well-known square root formula for Reno congestion control:

$$r_C = 1.22 / (R_C * p_C^{0.5}) \quad (5)$$

On the L4S side, we consider the Prague congestion control [I-D.briscoe-iccrp-prague-congestion-control] as the reference for steady-state dependence on congestion. Prague conforms to the same equation as DCTCP, but we do not use the equation derived in the DCTCP paper, which is only appropriate for step marking. The coupled marking, p_{CL} , is the appropriate one when considering throughput equivalence with Classic flows. Unlike step marking, coupled markings are inherently spaced out, so we use the formula for DCTCP packet rate with probabilistic marking derived in Appendix A of [PI2]. We use the equation without RTT-independence enabled, which will be explained later.

$$r_L = 2 / (R_L * p_{CL}) \quad (6)$$

For packet rate equivalence, we equate the two packet rates and rearrange into the same form as Equation (1), so the two can be equated and simplified to produce a formula for a theoretical coupling factor, which we shall call k^* :

$$\begin{aligned} r_C &= r_L \\ \Rightarrow p_C &= (p_{CL}/1.64 * R_L/R_C)^2 \end{aligned}$$

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

$$k^* = 1.64 * (R_C / R_L) \quad (7)$$

We say that this coupling factor is theoretical, because it is in terms of two RTTs, which raises two practical questions: i) for multiple flows with different RTTs, the RTT for each traffic class would have to be derived from the RTTs of all the flows in that class (actually the harmonic mean would be needed); ii) a network node cannot easily know the RTT of any of the flows anyway.

RTT-dependence is caused by window-based congestion control, so it ought to be reversed there, not in the network. Therefore, we use a fixed coupling factor in the network, and reduce RTT-dependence in L4S senders. We cannot expect Classic senders to all be updated to reduce their RTT-dependence. But solely addressing the problem in L4S senders at least makes RTT-dependence no worse – not just between L4S senders, but also between L4S and Classic senders.

Traditionally, throughput equivalence has been defined for flows under comparable conditions, including with the same base RTT [RFC2914]. So if we assume the same base RTT, R_b , for comparable flows, we can put both R_C and R_L in terms of R_b .

We can approximate the L4S RTT to be hardly greater than the base RTT, i.e. $R_L \approx R_b$. And we can replace R_C with $(R_b + q_C)$, where the Classic queue, q_C , depends on the target queue delay that the operator has configured for the Classic AQM.

Taking PI2 as an example Classic AQM, it seems that we could just take $R_C = R_b + \text{target}$ (recommended 15 ms by default in Appendix A.1). However, target is roughly the queue depth reached by the tips of the sawteeth of a congestion control, not the average [PI2param]. That is $R_{\text{max}} = R_b + \text{target}$.

The position of the average in relation to the max depends on the amplitude and geometry of the sawteeth. We consider two examples: Reno [RFC5681], as the most sensitive worst-case, and Cubic [RFC8312] in its Reno-friendly mode ('Creno') as the most prevalent congestion control algorithm on the Internet according to the references in [PI2param]. Both are AIMD, so we will generalize using b as the multiplicative decrease factor ($b_r = 0.5$ for Reno, $b_c = 0.7$ for Creno). Then:

$$\begin{aligned} R_C &= (R_{\text{max}} + b \cdot R_{\text{max}}) / 2 \\ &= R_{\text{max}} * (1+b)/2 \end{aligned}$$

$$R_{\text{reno}} = 0.75 * (R_b + \text{target}); \quad R_{\text{creno}} = 0.85 * (R_b + \text{target}). \quad (8)$$

Plugging all this into equation (7) we get a fixed coupling factor for each:

$$\begin{aligned} k_{\text{reno}} &= 1.64 * 0.75 * (R_b + \text{target}) / R_b \\ &= 1.23 * (1 + \text{target} / R_b); \quad k_{\text{creno}} = 1.39 * (1 + \text{target} / R_b) \end{aligned}$$

An operator can then choose the base RTT at which it wants throughput to be equivalent. For instance, if we recommend that the operator chooses $R_b = 25$ ms, as a typical base RTT between Internet users and CDNs [PI2param], then these coupling factors become:

$$\begin{aligned} k_{\text{reno}} &= 1.23 * (1 + 15/25) & k_{\text{creno}} &= 1.39 * (1 + 15/25) \\ &= 1.97 & &= 2.22 \\ &\approx 2 & &\approx 2 \end{aligned} \quad (9)$$

The approximation is relevant to any of the above example DualQ Coupled algorithms, which use a coupling factor that is an integer power of 2 to aid efficient implementation. It also fits best to the worst case (Reno).

To check the outcome of this coupling factor, we can express the ratio of L4S to Classic throughput by substituting from their rate equations (5) and (6), then also substituting for p_C in terms of p_{CL} , using equation (1) with $k=2$ as just determined for the Internet:

$$\begin{aligned} r_L / r_C &= 2 (R_C * p_C^{0.5}) / 1.22 (R_L * p_{CL}) \\ &= (R_C * p_{CL}) / (1.22 * R_L * p_{CL}) \\ &= R_C / (1.22 * R_L) \end{aligned} \tag{10}$$

As an example, we can then consider single competing CReNO and Prague flows, by expressing both their RTTs in (10) in terms of their base RTTs, R_{bC} and R_{bL} . So R_C is replaced by equation (8) for CReNO. And R_L is replaced by the $\max()$ function below, which represents the effective RTT of the current Prague congestion control [I-D.briscoe-iccrp-prague-congestion-control] in its (default) RTT-independent mode, because it sets a floor to the effective RTT that it uses for additive increase:

$$\begin{aligned} \tilde{R} &= 0.85 * (R_{bC} + \text{target}) / (1.22 * \max(R_{bL}, R_{\text{typ}})) \\ \tilde{R} &= (R_{bC} + \text{target}) / (1.4 * \max(R_{bL}, R_{\text{typ}})) \end{aligned}$$

It can be seen that, for base RTTs below target (15 ms), both the numerator and the denominator plateau, which has the desired effect of limiting RTT-dependence.

At the start of the above derivations, an explanation was promised for why the L4S throughput equation in equation (6) did not need to model RTT-independence. This is because we only use one point - at the the typical base RTT where the operator chooses to calculate the coupling factor. Then, throughput equivalence will at least hold at that chosen point. Nonetheless, assuming Prague senders implement RTT-independence over a range of RTTs below this, the throughput equivalence will then extend over that range as well.

Congestion control designers can choose different ways to reduce RTT-dependence. And each operator can make a policy choice to decide on a different base RTT, and therefore a different k , at which it wants throughput equivalence. Nonetheless, for the Internet, it makes sense to choose what is believed to be the typical RTT most users experience, because a Classic AQM's target queuing delay is also derived from a typical RTT for the Internet.

As a non-Internet example, for localized traffic from a particular ISP's data centre, using the measured RTTs, it was calculated that a value of $k = 8$ would achieve throughput equivalence, and experiments verified the formula very closely.

But, for a typical mix of RTTs across the general Internet, a value of $k=2$ is recommended as a good workable compromise.

Authors' Addresses

Koen De Schepper
Nokia Bell Labs
Antwerp
Belgium
Email: koen.de_schepper@nokia.com
URI: https://www.bell-labs.com/usr/koen.de_schepper

Bob Briscoe (editor)
Independent
United Kingdom
Email: ietf@bobbriscoe.net
URI: <http://bobbriscoe.net/>

Greg White
CableLabs
Louisville, CO,
United States of America
Email: G.White@CableLabs.com

Internet Engineering Task Force	G. Fairhurst
Internet-Draft	T. Jones
Updates: 4821, 4960, 6951, 8085, 8261 (if approved)	University of Aberdeen
Intended status: Standards Track	M. Tuexen
Expires: 12 December 2020	I. Ruengeler
	T. Voelker
	Muenster University of Applied Sciences
	10 June 2020

Packetization Layer Path MTU Discovery for Datagram Transports
draft-ietf-tsvwg-datagram-plpmtud-22

Abstract

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

This document updates RFC 4821 to specify the PLPMTUD method for datagram PLs. It also updates RFC 8085 to refer to the method specified in this document instead of the method in RFC 4821 for use with UDP datagrams. Section 7.3 of RFC 4960 recommends an endpoint apply the techniques in RFC 4821 on a per-destination-address basis. RFC 4960, RFC 6951, and RFC 8261 are updated to recommend that SCTP, SCTP encapsulated in UDP and SCTP encapsulated in DTLS use the method specified in this document instead of the method in RFC 4821.

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

When published, this specification updates RFC 4960, RFC 4821, RFC 8085 and RFC 8261.

Status of This Memo

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

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

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

This Internet-Draft will expire on 12 December 2020.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	4
1.1. Classical Path MTU Discovery	4
1.2. Packetization Layer Path MTU Discovery	6
1.3. Path MTU Discovery for Datagram Services	7
2. Terminology	8
3. Features Required to Provide Datagram PLPMTUD	11
4. DPLPMTUD Mechanisms	14
4.1. PLPMTU Probe Packets	14
4.2. Confirmation of Probed Packet Size	15
4.3. Black Hole Detection and Reducing the PLPMTU	15
4.4. The Maximum Packet Size (MPS)	17
4.5. Disabling the Effect of PMTUD	18
4.6. Response to PTB Messages	18
4.6.1. Validation of PTB Messages	18
4.6.2. Use of PTB Messages	19

5.	Datagram Packetization Layer PMTUD	20
5.1.	DPLPMTUD Components	21
5.1.1.	Timers	21
5.1.2.	Constants	22
5.1.3.	Variables	23
5.1.4.	Overview of DPLPMTUD Phases	24
5.2.	State Machine	26
5.3.	Search to Increase the PLPMTU	29
5.3.1.	Probing for a larger PLPMTU	29
5.3.2.	Selection of Probe Sizes	30
5.3.3.	Resilience to Inconsistent Path Information	30
5.4.	Robustness to Inconsistent Paths	31
6.	Specification of Protocol-Specific Methods	31
6.1.	Application support for DPLPMTUD with UDP or UDP-Lite	31
6.1.1.	Application Request	32
6.1.2.	Application Response	32
6.1.3.	Sending Application Probe Packets	32
6.1.4.	Initial Connectivity	32
6.1.5.	Validating the Path	32
6.1.6.	Handling of PTB Messages	32
6.2.	DPLPMTUD for SCTP	33
6.2.1.	SCTP/IPv4 and SCTP/IPv6	33
6.2.1.1.	Initial Connectivity	33
6.2.1.2.	Sending SCTP Probe Packets	33
6.2.1.3.	Validating the Path with SCTP	34
6.2.1.4.	PTB Message Handling by SCTP	34
6.2.2.	DPLPMTUD for SCTP/UDP	34
6.2.2.1.	Initial Connectivity	35
6.2.2.2.	Sending SCTP/UDP Probe Packets	35
6.2.2.3.	Validating the Path with SCTP/UDP	35
6.2.2.4.	Handling of PTB Messages by SCTP/UDP	35
6.2.3.	DPLPMTUD for SCTP/DTLS	35
6.2.3.1.	Initial Connectivity	35
6.2.3.2.	Sending SCTP/DTLS Probe Packets	36
6.2.3.3.	Validating the Path with SCTP/DTLS	36
6.2.3.4.	Handling of PTB Messages by SCTP/DTLS	36
6.3.	DPLPMTUD for QUIC	36
7.	Acknowledgments	36
8.	IANA Considerations	36
9.	Security Considerations	37
10.	References	38
10.1.	Normative References	38
10.2.	Informative References	39
	Appendix A. Revision Notes	41
	Authors' Addresses	46

1. Introduction

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

1.1. Classical Path MTU Discovery

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

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

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

Examples where PTB messages are not delivered include:

- * The generation of ICMP messages is usually rate limited. This could result in no PTB messages being generated to the sender (see section 2.4 of [RFC4443])
- * ICMP messages can be filtered by middleboxes (including firewalls) [RFC4890]. A firewall could be configured with a policy to block incoming ICMP messages, which would prevent reception of PTB messages to a sending endpoint behind this firewall.

- * When the router issuing the ICMP message drops a tunneled packet, the resulting ICMP message will be directed to the tunnel ingress. This tunnel endpoint is responsible for forwarding the ICMP message and also processing the quoted packet within the payload field to remove the effect of the tunnel, and return a correctly formatted ICMP message to the sender [I-D.ietf-intarea-tunnels]. Failure to do this prevents the PTB message reaching the original sender.
- * Asymmetry in forwarding can result in there being no return route to the original sender, which would prevent an ICMP message being delivered to the sender. This issue can also arise when policy-based routing is used, Equal Cost Multipath (ECMP) routing is used, or a middlebox acts as an application load balancer. An example is where the path towards the server is chosen by ECMP routing depending on bytes in the IP payload. In this case, when a packet sent by the server encounters a problem after the ECMP router, then any resulting ICMP message also needs to be directed by the ECMP router towards the original sender.
- * There are additional cases where the next hop destination fails to receive a packet because of its size. This could be due to misconfiguration of the layer 2 path between nodes, for instance the MTU configured in a layer 2 switch, or misconfiguration of the Maximum Receive Unit (MRU). If a packet is dropped by the link, this will not cause a PTB message to be sent to the original sender.

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

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

Note: The recommendation in RFC1812 [RFC1812] is that IPv4 routers return a quoted packet with as much of the original datagram as possible without the length of the ICMP datagram exceeding 576

bytes. IPv6 routers include as much of the invoking packet as possible without the ICMPv6 packet exceeding 1280 bytes [RFC4443].

- * The use of tunnels/encryption can reduce the size of the quoted packet returned to the original source address, increasing the risk that there could be insufficient bytes remaining for the sender to interpret the quoted transport information.
- * Even when the PTB message includes sufficient bytes of the quoted packet, the network layer could lack sufficient context to validate the message, because validation depends on information about the active transport flows at an endpoint node (e.g., the socket/address pairs being used, and other protocol header information).
- * When a packet is encapsulated/tunneled over an encrypted transport, the tunnel/encapsulation ingress might have insufficient context, or computational power, to reconstruct the transport header that would be needed to perform validation.
- * When an ICMP message is generated by a router in a network segment that has inserted a header into a packet, the quoted packet could contain additional protocol header information that was not included in the original sent packet, and which the PL sender does not process or may not know how to process. This could disrupt the ability of the sender to validate this PTB message.
- * A Network Address Translation (NAT) device that translates a packet header, ought to also translate ICMP messages and update the ICMP quoted packet [RFC5508] in that message. If this is not correctly translated then the sender would not be able to associate the message with the PL that originated the packet, and hence this ICMP message cannot be validated.

1.2. Packetization Layer Path MTU Discovery

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

In contrast to PMTUD, Packetization Layer Path MTU Discovery (PLPMTUD) [RFC4821] introduced a method that does not rely upon reception and validation of PTB messages. It is therefore more robust than Classical PMTUD. This has become the recommended approach for implementing discovery of the PMTU [BCP145].

It uses a general strategy where the PL sends probe packets to search for the largest size of unfragmented datagram that can be sent over a network path. Probe packets are sent to explore using a larger packet size. If a probe packet is successfully delivered (as determined by the PL), then the PLPMTU is raised to the size of the successful probe. If a black hole is detected (e.g., where packets of size PLPMTU are consistently not received), the method reduces the PLPMTU.

Datagram PLPMTUD introduces flexibility in implementation. At one extreme, it can be configured to only perform Black Hole Detection and recovery with increased robustness compared to Classical PMTUD. At the other extreme, all PTB processing can be disabled, and PLPMTUD replaces Classical PMTUD.

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

1.3. Path MTU Discovery for Datagram Services

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

The message size guidelines in section 3.2 of the UDP Usage Guidelines [BCP145] state "an application SHOULD either use the Path MTU information provided by the IP layer or implement Path MTU Discovery (PMTUD)", but does not provide a mechanism for discovering the largest size of unfragmented datagram that can be used on a network path. The present document updates RFC 8085 to specify this method in place of PLPMTUD [RFC4821] and provides a mechanism for sharing the discovered largest size as the MPS (see Section 4.4).

Section 10.2 of [RFC4821] recommended a PLPMTUD probing method for the Stream Control Transport Protocol (SCTP). SCTP utilizes probe packets consisting of a minimal sized HEARTBEAT chunk bundled with a PAD chunk as defined in [RFC4820]. However, RFC 4821 did not provide a complete specification. The present document replaces that description by providing a complete specification.

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

Section 4 and Section 5 define the protocol mechanisms and specification for Datagram Packetization Layer Path MTU Discovery (DPLPMTUD).

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

Section 6 also provides updated recommendations for [RFC6951] and [RFC8261].

2. Terminology

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

The following terminology is defined. Relevant terms are directly copied from [RFC4821], and the definitions in [RFC1122].

Acknowledged PL: A PL that includes a mechanism that can confirm successful delivery of datagrams to the remote PL endpoint (e.g., SCTP). Typically, the PL receiver returns acknowledgments corresponding to the received datagrams, which can be utilised to detect black-holing of packets (c.f., Unacknowledged PL).

Actual PMTU: The Actual PMTU is the PMTU of a network path between a sender PL and a destination PL, which the DPLPMTUD algorithm seeks to determine.

Black Hole: A Black Hole is encountered when a sender is unaware that packets are not being delivered to the destination end point. Two types of Black Hole are relevant to DPLPMTUD:

- * Packets encounter a packet Black Hole when packets are not delivered to the destination endpoint (e.g., when the sender transmits packets of a particular size with a previously known

effective PMTU and they are discarded by the network).

- * An ICMP Black Hole is encountered when the sender is unaware that packets are not delivered to the destination endpoint because PTB messages are not received by the originating PL sender.

Classical Path MTU Discovery: Classical PMTUD is a process described in [RFC1191] and [RFC8201], in which nodes rely on PTB messages to learn the largest size of unfragmented packet that can be used across a network path.

Datagram: A datagram is a transport-layer protocol data unit, transmitted in the payload of an IP packet.

Effective PMTU: The Effective PMTU is the current estimated value for PMTU that is used by a PMTUD. This is equivalent to the PLPMTU derived by PLPMTUD plus the size of any headers added below the PL, including the IP layer headers.

EMTU_S: The Effective MTU for sending (EMTU_S) is defined in [RFC1122] as "the maximum IP datagram size that may be sent, for a particular combination of IP source and destination addresses...".

EMTU_R: The Effective MTU for receiving (EMTU_R) is designated in [RFC1122] as "the largest datagram size that can be reassembled".

Link: A Link is a communication facility or medium over which nodes can communicate at the link layer, i.e., a layer below the IP layer. Examples are Ethernet LANs and Internet (or higher) layer tunnels.

Link MTU: The Link Maximum Transmission Unit (MTU) is the size in bytes of the largest IP packet, including the IP header and payload, that can be transmitted over a link. Note that this could more properly be called the IP MTU, to be consistent with how other standards organizations use the acronym. This includes the IP header, but excludes link layer headers and other framing that is not part of IP or the IP payload. Other standards organizations generally define the link MTU to include the link layer headers. This specification continues the requirement in [RFC4821], that states "All links MUST enforce their MTU: links that might non-deterministically deliver packets that are larger than their rated MTU MUST consistently discard such packets."

MAX_PLPMTU: The MAX_PLPMTU is the largest size of PLPMTU that DPLPMTUD will attempt to use (see the constants defined in Section 5.1.2).

MIN_PLPMTU: The MIN_PLPMTU is the smallest size of PLPMTU that DPLPMTUD will attempt to use (see the constants defined in Section 5.1.2).

MPS: The Maximum Packet Size (MPS) is the largest size of application data block that can be sent across a network path by a PL using a single Datagram (see Section 4.4).

MSL: Maximum Segment Lifetime (MSL) The maximum delay a packet is expected to experience across a path, taken as 2 minutes [BCP145].

Packet: A Packet is the IP header(s) and any extension headers/options plus the IP payload.

Packetization Layer (PL): The PL is a layer of the network stack that places data into packets and performs transport protocol functions. Examples of a PL include: TCP, SCTP, SCTP over UDP, SCTP over DTLS, or QUIC.

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

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

PTB: In this document, the term PTB message is applied to both IPv4 ICMP Unreachable messages (type 3) that carry the error Fragmentation Needed (Type 3, Code 4) [RFC0792] and ICMPv6 Packet Too Big messages (Type 2) [RFC4443].

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

PL_PTBSIZE: The size reported in a validated PTB message, reduced by the size of all headers added by layers below the PL.

PLPMTU: The Packetization Layer PMTU is an estimate of the largest size of PL datagram that can be sent by a path, controlled by PLPMTUD.

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

Probe packet: A probe packet is a datagram sent with a purposely chosen size (typically the current PLPMTU or larger) to detect if

packets of this size can be successfully sent end-to-end across the network path.

Unacknowledged PL: A PL that does not itself provide a mechanism to confirm delivery of datagrams to the remote PL endpoint (e.g., UDP), and therefore requires DPLPMTUD to provide a mechanism to detect black-holing of packets (c.f., Acknowledged PL).

3. Features Required to Provide Datagram PLPMTUD

The principles expressed in [RFC4821] apply to the use of the technique with any PL. TCP PLPMTUD has been defined using standard TCP protocol mechanisms. Unlike TCP, a datagram PL requires additional mechanisms and considerations to implement PLPMTUD.

The requirements for datagram PLPMTUD are:

1. Managing the PLPMTU: For datagram PLs, the PLPMTU is managed by DPLPMTUD. A PL MUST NOT send a datagram (other than a probe packet) with a size at the PL that is larger than the current PLPMTU.
2. Probe packets: The network interface below PL is REQUIRED to provide a way to transmit a probe packet that is larger than the PLPMTU. In IPv4, a probe packet MUST be sent with the Don't Fragment (DF) bit set in the IP header, and without network layer endpoint fragmentation. In IPv6, a probe packet is always sent without source fragmentation (as specified in section 5.4 of [RFC8201]).
3. Reception feedback: The destination PL endpoint is REQUIRED to provide a feedback method that indicates to the DPLPMTUD sender when a probe packet has been received by the destination PL endpoint. Section 6 provides examples of how a PL can provide this acknowledgment of received probe packets.
4. Probe loss recovery: It is RECOMMENDED to use probe packets that do not carry any user data that would require retransmission if lost. Most datagram transports permit this. If a probe packet contains user data requiring retransmission in case of loss, the PL (or layers above) are REQUIRED to arrange any retransmission/repair of any resulting loss. The PL is REQUIRED to be robust in the case where probe packets are lost due to other reasons (including link transmission error, congestion).
5. PMTU parameters: A DPLPMTUD sender is RECOMMENDED to utilize information about the maximum size of packet that can be transmitted by the sender on the local link (e.g., the local Link

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

6. Processing PTB messages: A DPLPMTUD sender MAY optionally utilize PTB messages received from the network layer to help identify when a network path does not support the current size of probe packet. Any received PTB message MUST be validated before it is used to update the PLPMTU discovery information [RFC8201]. This validation confirms that the PTB message was sent in response to a packet originating by the sender, and needs to be performed before the PLPMTU discovery method reacts to the PTB message. A PTB message MUST NOT be used to increase the PLPMTU [RFC8201], but could trigger a probe to test for a larger PLPMTU. A valid PTB_SIZE is converted to a PL_PTBSIZE before it is to be used in the DPLPMTUD state machine. A PL_PTBSIZE that is greater than that currently probed SHOULD be ignored. (This PTB message ought to be discarded without further processing, but could be utilized as an input that enables a resilience mode).
7. Probing and congestion control: A PL MAY use a congestion controller to decide when to send a probe packet. If transmission of probe packets is limited by the congestion controller, this could result in transmission of probe packets being delayed or suspended during congestion. When the transmission of probe packets is not controlled by the congestion controller, the interval between probe packets MUST be at least one RTT. Loss of a probe packet SHOULD NOT be treated as an indication of congestion and SHOULD NOT trigger a congestion control reaction [RFC4821], because this could result in unnecessary reduction of the sending rate. An update to the PLPMTU (or MPS) MUST NOT increase the congestion window measured in bytes [RFC4821]. Therefore, an increase in the packet size does not cause an increase in the data rate in bytes per second. A PL that maintains the congestion window in terms of a limit to the number of outstanding fixed size packets SHOULD adapt this limit to compensate for the size of the actual packets. The transmission of probe packets can interact with the operation of a PL that performs burst mitigation or pacing and could need transmission of probe packets to be regulated by these methods.

8. Probing and flow control: Flow control at the PL concerns the end-to-end flow of data using the PL service. Flow control SHOULD NOT apply to DPLPMTU when probe packets use a design that does not carry user data to the remote application.
9. Shared PLPMTU state: The PMTU value calculated from the PLPMTU MAY also be stored with the corresponding entry associated with the destination in the IP layer cache, and used by other PL instances. The specification of PLPMTUD [RFC4821] states: "If PLPMTUD updates the MTU for a particular path, all Packetization Layer sessions that share the path representation (as described in Section 5.2 of [RFC4821]) SHOULD be notified to make use of the new MTU". Such methods MUST be robust to the wide variety of underlying network forwarding behaviors. Section 5.2 of [RFC8201] provides guidance on the caching of PMTU information and also the relation to IPv6 flow labels.

In addition, the following principles are stated for design of a DPLPMTUD method:

- * A PL MAY be designed to segment data blocks larger than the MPS into multiple datagrams. However, not all datagram PLs support segmentation of data blocks. It is RECOMMENDED that methods avoid forcing an application to use an arbitrary small MPS for transmission while the method is searching for the currently supported PLPMTU. A reduced MPS can adversely impact the performance of an application.
- * To assist applications in choosing a suitable data block size, the PL is RECOMMENDED to provide a primitive that returns the MPS derived from the PLPMTU to the higher layer using the PL. The value of the MPS can change following a change in the path, or loss of probe packets.
- * Path validation: It is RECOMMENDED that methods are robust to path changes that could have occurred since the path characteristics were last confirmed, and to the possibility of inconsistent path information being received.
- * Datagram reordering: A method is REQUIRED to be robust to the possibility that a flow encounters reordering, or the traffic (including probe packets) is divided over more than one network path.
- * Datagram delay and duplication: The feedback mechanism is REQUIRED to be robust to the possibility that packets could be significantly delayed or duplicated along a network path.

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

4. DPLPMTUD Mechanisms

This section lists the protocol mechanisms used in this specification.

4.1. PLPMTU Probe Packets

The DPLPMTUD method relies upon the PL sender being able to generate probe packets with a specific size. TCP is able to generate these probe packets by choosing to appropriately segment data being sent [RFC4821]. In contrast, a datagram PL that constructs a probe packet has to either request an application to send a data block that is larger than that generated by an application, or to utilize padding functions to extend a datagram beyond the size of the application data block. Protocols that permit exchange of control messages (without an application data block) can generate a probe packet by extending a control message with padding data. The total size of a probe packet includes all headers and padding added to the payload data being sent (e.g., including protocol option fields, security-related fields such as an Authenticated Encryption with Associated Data (AEAD) tag and TLS record layer padding).

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

This results in three possible ways that a sender can create a probe packet:

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

Probing using application data and padding data: A probe packet that contains a data block supplied by an application that is combined with padding to inflate the length of the datagram to the size of the probe packet.

Probing using application data: A probe packet that contains a data block supplied by an application that matches the size of the

probe packet. This method requests the application to issue a data block of the desired probe size.

A PL that uses a probe packet carrying application data and needs protection from the loss of this probe packet could perform transport-layer retransmission/repair of the data block (e.g., by retransmission after loss is detected or by duplicating the data block in a datagram without the padding data). This retransmitted data block might possibly need to be sent using a smaller PLPMTU, which could force the PL to use a smaller packet size to traverse the end-to-end path. (This could utilize endpoint network-layer fragmentation or a PL that can re-segment the data block into multiple datagrams).

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

Probe messages sent by a PL MUST contain enough information to uniquely identify the probe within Maximum Segment Lifetime (e.g., including a unique identifier from the PL or the DPLPMTUD implementation), while being robust to reordering and replay of probe response and PTB messages.

4.2. Confirmation of Probed Packet Size

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

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

A PL that does not acknowledge data reception (e.g., UDP and UDP-Lite) is unable itself to detect when the packets that it sends are discarded because their size is greater than the actual PMTU. These PLs need to rely on an application protocol to detect this loss.

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

4.3. Black Hole Detection and Reducing the PLPMTU

The description that follows uses the set of constants defined in Section 5.1.2 and variables defined in Section 5.1.3.

Black Hole Detection is triggered by an indication that the network path could be unable to support the current PLPMTU size.

There are three indicators that can detect black holes:

- * A validated PTB message can be received that indicates a PL_PTB_SIZE less than the current PLPMTU. A DPLPMTUD method MUST NOT rely solely on this method.
- * A PL can use the DPLPMTUD probing mechanism to periodically generate probe packets of the size of the current PLPMTU (e.g., using the confirmation timer Section 5.1.1). A timer tracks whether acknowledgments are received. Successive loss of probes is an indication that the current path no longer supports the PLPMTU (e.g., when the number of probe packets sent without receiving an acknowledgment, PROBE_COUNT, becomes greater than MAX_PROBES).
- * A PL can utilize an event that indicates the network path no longer sustains the sender's PLPMTU size. This could use a mechanism implemented within the PL to detect excessive loss of data sent with a specific packet size and then conclude that this excessive loss could be a result of an invalid PLPMTU (as in PLPMTUD for TCP [RFC4821]).

The three methods can result in different transmission patterns for packet probes and are expected to result in different responsiveness following a change in the actual PMTU.

A PL MAY inhibit sending probe packets when no application data has been sent since the previous probe packet. A PL that resumes sending user data MAY continue PLPMTU discovery for each path. This allows it to use an up-to-date PLPMTU. However, this could result in additional packets being sent.

When the method detects the current PLPMTU is not supported, DPLPMTUD sets a lower PLPMTU, and sets a lower MPS. The PL then confirms that the new PLPMTU can be successfully used across the path. A probe packet could need to have a size less than the size of the data block generated by the application.

4.4. The Maximum Packet Size (MPS)

The result of probing determines a usable PLPMTU, which is used to set the MPS used by the application. The MPS is smaller than the PLPMTU because it is reduced by the size of PL headers (including the overhead of security-related fields such as an AEAD tag and TLS record layer padding). The relationship between the MPS and the PLPMTU is illustrated in Figure 1.

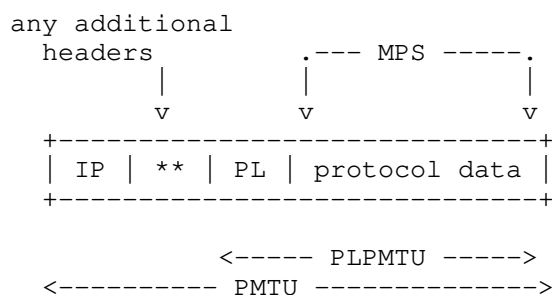


Figure 1: Relationship between MPS and PLPMTU

A PL is unable to send a packet (other than a probe packet) with a size larger than the current PLPMTU at the network layer. To avoid this, a PL MAY be designed to segment data blocks larger than the MPS into multiple datagrams.

DPLPMTUD seeks to avoid IP fragmentation. An attempt to send a data block larger than the MPS will therefore fail if a PL is unable to segment data. To determine the largest data block that can be sent, a PL SHOULD provide applications with a primitive that returns the MPS, derived from the current PLPMTU.

If DPLPMTUD results in a change to the MPS, the application needs to adapt to the new MPS. A particular case can arise when packets have been sent with a size less than the MPS and the PLPMTU was subsequently reduced. If these packets are lost, the PL MAY segment the data using the new MPS. If a PL is unable to re-segment a previously sent datagram (e.g., [RFC4960]), then the sender either discards the datagram or could perform retransmission using network-layer fragmentation to form multiple IP packets not larger than the PLPMTU. For IPv4, the use of endpoint fragmentation by the sender is preferred over clearing the DF bit in the IPv4 header. Operational experience reveals that IP fragmentation can reduce the reliability of Internet communication [I-D.ietf-intarea-frag-fragile], which may reduce the probability of successful retransmission.

4.5. Disabling the Effect of PMTUD

A PL implementing this specification MUST suspend network layer processing of outgoing packets that enforces a PMTU [RFC1191][RFC8201] for each flow utilizing DPLPMTUD, and instead use DPLPMTUD to control the size of packets that are sent by a flow. This removes the need for the network layer to drop or fragment sent packets that have a size greater than the PMTU.

4.6. Response to PTB Messages

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

Section 4.6.1 first describes validation for both IPv4 ICMP Unreachable messages (type 3) and ICMPv6 Packet Too Big messages, both of which are referred to as PTB messages in this document.

4.6.1. Validation of PTB Messages

This section specifies utilization and validation of PTB messages.

- * A simple implementation MAY ignore received PTB messages and in this case the PLPMTU is not updated when a PTB message is received.
- * A PL that supports PTB messages MUST validate these messages before they are further processed.

A PL that receives a PTB message from a router or middlebox performs ICMP validation (see Section 4 of [RFC8201] and Section 5.2 of [BCP145]). Because DPLPMTUD operates at the PL, the PL needs to check that each received PTB message is received in response to a packet transmitted by the endpoint PL performing DPLPMTUD.

The PL MUST check the protocol information in the quoted packet carried in an ICMP PTB message payload to validate the message originated from the sending node. This validation includes determining that the combination of the IP addresses, the protocol, the source port and destination port match those returned in the quoted packet - this is also necessary for the PTB message to be passed to the corresponding PL.

The validation SHOULD utilize information that it is not simple for an off-path attacker to determine [BCP145]. For example, it could

check the value of a protocol header field known only to the two PL endpoints. A datagram application that uses well-known source and destination ports ought to also rely on other information to complete this validation.

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

Section 4.6.2 describes this processing of PTB messages.

4.6.2. Use of PTB Messages

PTB messages that have been validated MAY be utilized by the DPLPMTUD algorithm, but MUST NOT be used directly to set the PLPMTU.

Before using the size reported in the PTB message it must first be converted to a PL_PTB_SIZE. The PL_PTB_SIZE is smaller than the PTB_SIZE because it is reduced by headers below the PL including any IP options or extensions added to the PL packet.

A method that utilizes these PTB messages can improve the speed at which the algorithm detects an appropriate PLPMTU by triggering an immediate probe for the PL_PTB_SIZE (resulting in a network-layer packet of size PTB_SIZE), compared to one that relies solely on probing using a timer-based search algorithm.

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

This section provides a summary of how PTB messages can be utilized. (This uses the set of constants defined in Section 5.1.2). This processing depends on the PL_PTB_SIZE and the current value of a set of variables:

PL_PTB_SIZE < MIN_PLPMTU

- * Invalid PL_PTB_SIZE see Section 4.6.1.
- * PTB message ought to be discarded without further processing (i.e., PLPMTU is not modified).
- * The information could be utilized as an input that triggers enabling a resilience mode (see Section 5.3.3).

MIN_PLPMTU < PL_PTB_SIZE < BASE_PLPMTU

- * A robust PL MAY enter an error state (see Section 5.2) for an IPv4 path when the PL_PTB_SIZE reported in the PTB message is larger than or equal to 68 bytes [RFC0791] and when this is less than the BASE_PLPMTU.
- * A robust PL MAY enter an error state (see Section 5.2) for an IPv6 path when the PL_PTB_SIZE reported in the PTB message is larger than or equal to 1280 bytes [RFC8200] and when this is less than the BASE_PLPMTU.

BASE_PLPMTU <= PL_PTB_SIZE < PLPMTU

- * This could be an indication of a black hole. The PLPMTU SHOULD be set to BASE_PLPMTU (the PLPMTU is reduced to the BASE_PLPMTU to avoid unnecessary packet loss when a black hole is encountered).
- * The PL ought to start a search to quickly discover the new PLPMTU. The PL_PTB_SIZE reported in the PTB message can be used to initialize a search algorithm.

PLPMTU < PL_PTB_SIZE < PROBED_SIZE

- * The PLPMTU continues to be valid, but the size of a packet used to search (PROBED_SIZE) was larger than the actual PMTU.
- * The PLPMTU is not updated.
- * The PL can use the reported PL_PTB_SIZE from the PTB message as the next search point when it resumes the search algorithm.

PL_PTB_SIZE >= PROBED_SIZE

- * Inconsistent network signal.
- * PTB message ought to be discarded without further processing (i.e., PLPMTU is not modified).
- * The information could be utilized as an input to trigger enabling a resilience mode.

5. Datagram Packetization Layer PMTUD

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

DPLPMTUD SHOULD only be performed at one layer between a pair of endpoints. Therefore, an upper PL or application should avoid using DPLPMTUD when this is already enabled in a lower layer. A PL MUST adjust the MPS indicated by DPLPMTUD to account for any additional overhead introduced by the PL.

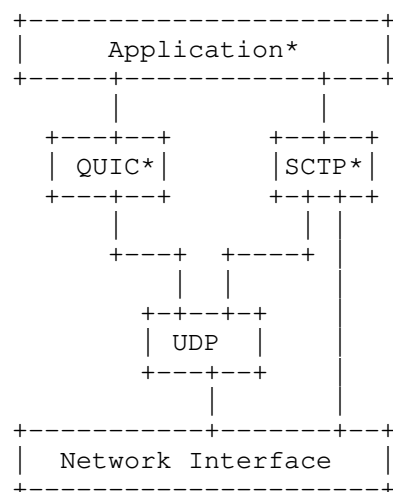


Figure 2: Examples where DPLPMTUD can be implemented

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

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

5.1. DPLPMTUD Components

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

5.1.1. Timers

The method utilizes up to three timers:

PROBE_TIMER: The PROBE_TIMER is configured to expire after a period longer than the maximum time to receive an acknowledgment to a probe packet. This value MUST NOT be smaller than 1 second, and SHOULD be larger than 15 seconds. Guidance on selection of the

timer value are provided in Section 3.1.1 of the UDP Usage Guidelines [BCP145].

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

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

CONFIRMATION_TIMER: When an acknowledged PL is used, this timer MUST NOT be used. For other PLs, the CONFIRMATION_TIMER is configured to the period a PL sender waits before confirming the current PLPMTU is still supported. This is less than the PMTU_RAISE_TIMER and used to decrease the PLPMTU (e.g., when a black hole is encountered). Confirmation needs to be frequent enough when data is flowing that the sending PL does not black hole extensive amounts of traffic. Guidance on selection of the timer value are provided in Section 3.1.1 of the UDP Usage Guidelines [BCP145].

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

DPLPMTD specifies various timers, however an implementation could choose to realise these timer functions using a single timer.

5.1.2. Constants

The following constants are defined:

MAX_PROBES: The MAX_PROBES is the maximum value of the PROBE_COUNT counter (see Section 5.1.3). MAX_PROBES represents the limit for the number of consecutive probe attempts of any size. Search algorithms benefit from a MAX_PROBES value greater than 1 because this can provide robustness to isolated packet loss. The default value of MAX_PROBES is 3.

MIN_PLPMTU: The MIN_PLPMTU is the smallest size of PLPMTU that DPLPMTUD will attempt to use. An endpoint could need to be configure the MIN_PLPMTU to provide space for extension headers and other encapsulations at layers below the PL. This value can

be interface and path dependent. For IPv6, this size is greater than or equal to the size at the PL that results in an 1280 byte IPv6 packet, as specified in [RFC8200]. For IPv4, this size is greater than or equal to the size at the PL that results in an 68 byte IPv4 packet. Note: An IPv4 router is required to be able to forward a datagram of 68 bytes without further fragmentation. This is the combined size of an IPv4 header and the minimum fragment size of 8 bytes. In addition, receivers are required to be able to reassemble fragmented datagrams at least up to 576 bytes, as stated in section 3.3.3 of [RFC1122].

MAX_PLPMTU: The MAX_PLPMTU is the largest size of PLPMTU. This has to be less than or equal to the maximum size of the PL packet that can be sent on the outgoing interface (constrained by the local interface MTU). When known, this also ought to be less than the maximum size of PL packet that can be received by the remote endpoint (constrained by EMTU_R). It can be limited by the design or configuration of the PL being used. An application, or PL, MAY choose a smaller MAX_PLPMTU when there is no need to send packets larger than a specific size.

BASE_PLPMTU: The BASE_PLPMTU is a configured size expected to work for most paths. The size is equal to or larger than the MIN_PLPMTU and smaller than the MAX_PLPMTU. For most PLs a suitable BASE_PLPMTU will be larger than 1200 bytes. When using IPv4, there is no currently equivalent size specified and a default BASE_PLPMTU of 1200 bytes is RECOMMENDED.

5.1.3. Variables

This method utilizes a set of variables:

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

PROBE_COUNT: The PROBE_COUNT is a count of the number of successive unsuccessful probe packets that have been sent. Each time a probe packet is acknowledged, the value is set to zero. (Some probe loss is expected while searching, therefore loss of a single probe is not an indication of a PMTU problem.)

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

DPLPMTUD algorithm to further increase PROBED_SIZE toward sending a probe with the size of the actual PMTU.

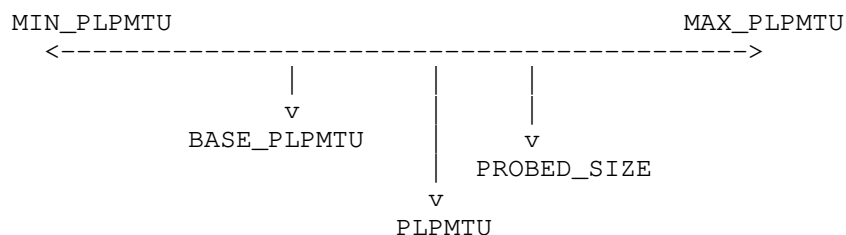


Figure 3: Relationships between packet size constants and variables

5.1.4. Overview of DPLPMTUD Phases

This section provides a high-level informative view of the DPLPMTUD method, by describing the movement of the method through several phases of operation. More detail is available in the state machine Section 5.2.

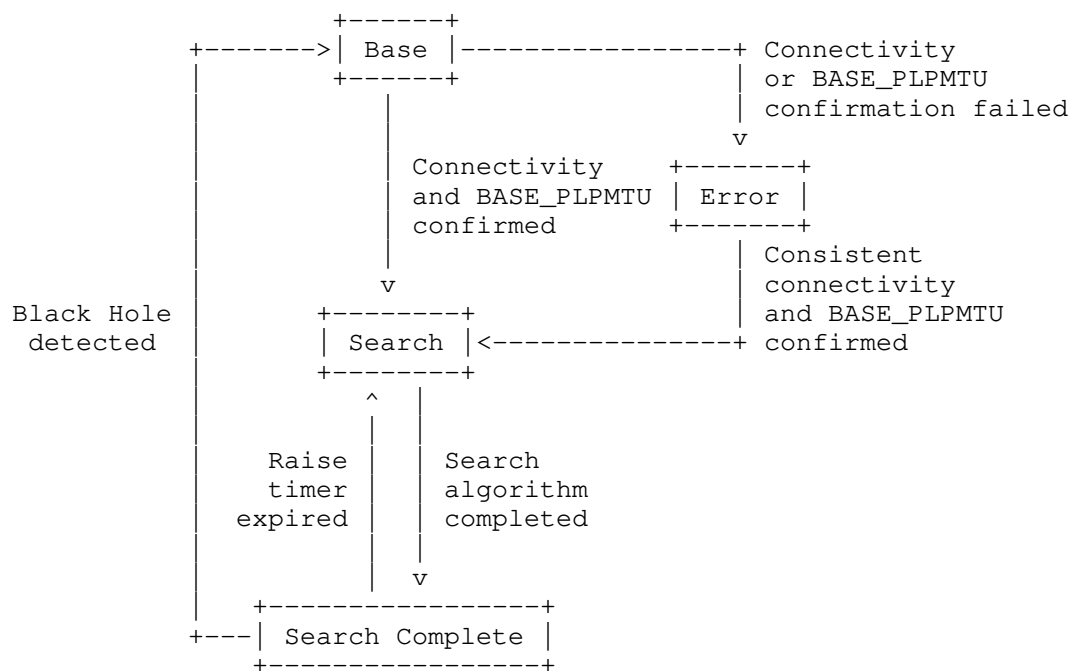


Figure 4: DPLPMTUD Phases

Base: The Base Phase confirms connectivity to the remote peer using packets of the BASE_PLPMTU. The confirmation of connectivity is implicit for a connection-oriented PL (where it can be performed in a PL connection handshake). A connectionless PL sends a probe packet and uses acknowledgment of this probe packet to confirm that the remote peer is reachable.

The sender also confirms that BASE_PLPMTU is supported across the network path. This may be achieved using a PL mechanism (e.g., using a handshake packet of size BASE_PLPMTU), or by sending a probe packet of size BASE_PLPMTU and confirming that this is received.

A probe packet of size BASE_PLPMTU can be sent immediately on the initial entry to the Base Phase (following a connectivity check). A PL that does not wish to support a path with a PLPMTU less than BASE_PLPMTU can simplify the phase into a single step by performing the connectivity checks with a probe of the BASE_PLPMTU size.

Once confirmed, DPLPMTUD enters the Search Phase. If the Base Phase fails to confirm the BASE_PLPMTU, DPLPMTUD enters the Error Phase.

Search: The Search Phase utilizes a search algorithm to send probe packets to seek to increase the PLPMTU. The algorithm concludes when it has found a suitable PLPMTU, by entering the Search Complete Phase.

A PL could respond to PTB messages using the PTB to advance or terminate the search, see Section 4.6.

Search Complete: The Search Complete Phase is entered when the PLPMTU is supported across the network path. A PL can use a CONFIRMATION_TIMER to periodically repeat a probe packet for the current PLPMTU size. If the sender is unable to confirm reachability (e.g., if the CONFIRMATION_TIMER expires) or the PL signals a lack of reachability, a black hole has been detected and DPLPMTUD enters the Base phase.

The PMTU_RAISE_TIMER is used to periodically resume the search phase to discover if the PLPMTU can be raised. Black Hole Detection causes the sender to enter the Base Phase.

Error: The Error Phase is entered when there is conflicting or invalid PLPMTU information for the path (e.g., a failure to support the BASE_PLPMTU) that cause DPLPMTUD to be unable to progress and the PLPMTU is lowered.

DPLPMTUD remains in the Error Phase until a consistent view of the path can be discovered and it has also been confirmed that the path supports the BASE_PLPMTU (or DPLPMTUD is suspended).

A method that only reduces the PLPMTU to a suitable size would be sufficient to ensure reliable operation, but can be very inefficient when the actual PMTU changes or when the method (for whatever reason) makes a suboptimal choice for the PLPMTU.

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

5.2. State Machine

A state machine for DPLPMTUD is depicted in Figure 5. If multipath or multihoming is supported, a state machine is needed for each path.

Note: Not all changes are shown to simplify the diagram.

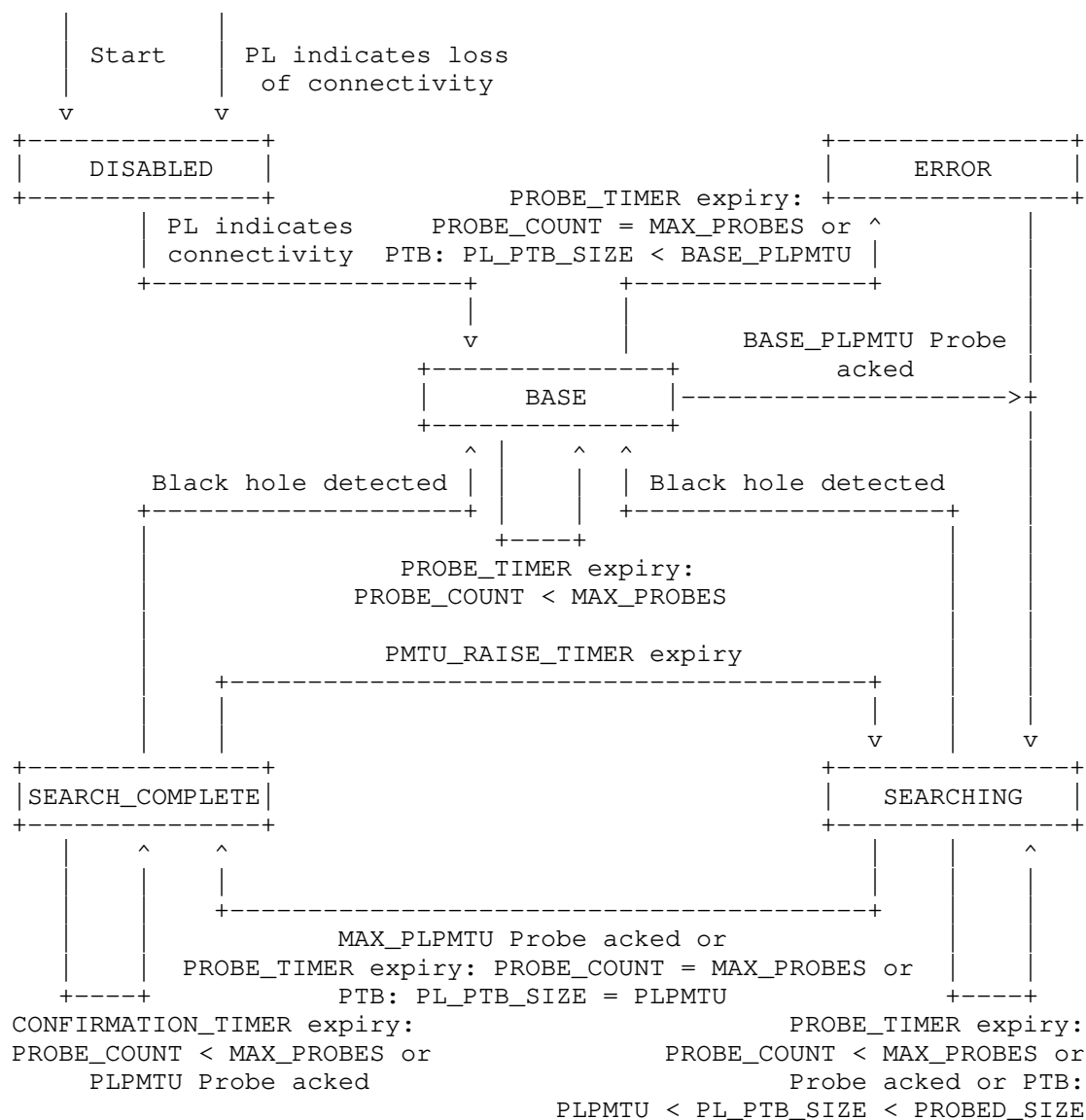


Figure 5: State machine for Datagram PLPMTUD

The following states are defined:

DISABLED: The DISABLED state is the initial state before probing has started. It is also entered from any other state, when the PL indicates loss of connectivity. This state is left once the PL

indicates connectivity to the remote PL. When transitioning to the BASE state, a probe packet of size BASE_PLPMTU can be sent immediately.

BASE: The BASE state is used to confirm that the BASE_PLPMTU size is supported by the network path and is designed to allow an application to continue working when there are transient reductions in the actual PMTU. It also seeks to avoid long periods when a sender searching for a larger PLPMTU is unaware that packets are not being delivered due to a packet or ICMP Black Hole.

On entry, the PROBED_SIZE is set to the BASE_PLPMTU size and the PROBE_COUNT is set to zero.

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

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

SEARCHING: The SEARCHING state is the main probing state. This state is entered when probing for the BASE_PLPMTU completes.

Each time a probe packet is acknowledged, the PROBE_COUNT is set to zero, the PLPMTU is set to the PROBED_SIZE and then the PROBED_SIZE is increased using the search algorithm (as described in Section 5.3).

When a probe packet is sent and not acknowledged within the period of the PROBE_TIMER, the PROBE_COUNT is incremented and a new probe packet is transmitted.

The state is exited to enter SEARCH_COMPLETE when the PROBE_COUNT reaches MAX_PROBES, a validated PTB is received that corresponds to the last successfully probed size (PL_PTB_SIZE = PLPMTU), or a probe of size MAX_PLPMTU is acknowledged (PLPMTU = MAX_PLPMTU).

When a black hole is detected in the SEARCHING state, this causes the PL sender to enter the BASE state.

SEARCH_COMPLETE: The SEARCH_COMPLETE state indicates that a search has completed. This is the normal maintenance state, where the PL is not probing to update the PLPMTU. DPLPMTUD remains in this state until either the PMTU_RAISE_TIMER expires or a black hole is detected.

When DPLPMTUD uses an unacknowledged PL and is in the SEARCH_COMPLETE state, a CONFIRMATION_TIMER periodically resets the PROBE_COUNT and schedules a probe packet with the size of the PLPMTU. If MAX_PROBES successive PLPMTUD sized probes fail to be acknowledged the method enters the BASE state. When used with an acknowledged PL (e.g., SCTP), DPLPMTUD SHOULD NOT continue to generate PLPMTU probes in this state.

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

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

Note: MIN_PLPMTU could be identical to BASE_PLPMTU, simplifying the actions in this state.

5.3. Search to Increase the PLPMTU

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

5.3.1. Probing for a larger PLPMTU

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

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

The PROBE_COUNT is initialized to zero when the first probe with a size greater than or equal to PLPMTUD is sent. Each probe packet successfully sent to the remote peer is confirmed by acknowledgment at the PL, see Section 4.1.

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

If the timer expires before a probe packet is acknowledged, the probe has failed to confirm the PROBED_SIZE. Each time the PROBE_TIMER expires, the PROBE_COUNT is incremented, the PROBE_TIMER is reinitialized, and a new probe of the same size or any other size (determined by the search algorithm) can be sent. The maximum number of consecutive failed probes is configured (MAX_PROBES). If the value of the PROBE_COUNT reaches MAX_PROBES, probing will stop, and the PL sender enters the SEARCH_COMPLETE state.

5.3.2. Selection of Probe Sizes

The search algorithm determines a minimum useful gain in PLPMTU. It would not be constructive for a PL sender to attempt to probe for all sizes. This would incur unnecessary load on the path. Implementations SHOULD select the set of probe packet sizes to maximize the gain in PLPMTU from each search step.

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

5.3.3. Resilience to Inconsistent Path Information

A decision to increase the PLPMTU needs to be resilient to the possibility that information learned about the network path is inconsistent. A path is inconsistent when, for example, probe packets are lost due to other reasons (i.e., not packet size) or due to frequent path changes. Frequent path changes could occur by unexpected "flapping" - where some packets from a flow pass along one path, but other packets follow a different path with different properties.

A PL sender is able to detect inconsistency from the sequence of PLPMTU probes that are acknowledged or the sequence of PTB messages that it receives. When inconsistent path information is detected, a PL sender could use an alternate search mode that clamps the offered

MPS to a smaller value for a period of time. This avoids unnecessary loss of packets.

5.4. Robustness to Inconsistent Paths

Some paths could be unable to sustain packets of the BASE_PLPMTU size. The Error State could be implemented to provide robustness to such paths. This allows fallback to a smaller than desired PLPMTU, rather than suffer connectivity failure. This could utilize methods such as endpoint IP fragmentation to enable the PL sender to communicate using packets smaller than the BASE_PLPMTU.

6. Specification of Protocol-Specific Methods

DPLPMTUD requires protocol-specific details to be specified for each PL that is used.

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

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

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

The DPLPMTUD method can be implemented as a part of an application built directly or indirectly on UDP or UDP-Lite, but relies on higher-layer protocol features to implement the method [BCP145].

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

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

6.1.1. Application Request

An application needs an application-layer protocol mechanism (such as a message acknowledgment method) that solicits a response from a destination endpoint. The method SHOULD allow the sender to check the value returned in the response to provide additional protection from off-path insertion of data [BCP145]. Suitable methods include a parameter known only to the two endpoints, such as a session ID or initialized sequence number.

6.1.2. Application Response

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

6.1.3. Sending Application Probe Packets

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

6.1.4. Initial Connectivity

An application that does not have other higher-layer information confirming connectivity with the remote peer SHOULD implement a connectivity mechanism using acknowledged probe packets before entering the BASE state.

6.1.5. Validating the Path

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

6.1.6. Handling of PTB Messages

An application that is able and wishes to receive PTB messages MUST perform ICMP validation as specified in Section 5.2 of [BCP145]. This requires that the application checks each received PTB message to validate that it was received in response to transmitted traffic and that the reported PL_PTB_SIZE is less than the current probed size (see Section 4.6.2). A validated PTB message MAY be used

as input to the DPLPMTUD algorithm, but MUST NOT be used directly to set the PLPMTU.

6.2. DPLPMTUD for SCTP

Section 10.2 of [RFC4821] specified a recommended PLPMTUD probing method for SCTP and Section 7.3 of [RFC4960] recommended an endpoint apply the techniques in RFC4821 on a per-destination-address basis. The specification for DPLPMTUD continues the practice of using the PL to discover the PMTU, but updates, RFC4960 with a recommendation to use the method specified in this document: The RECOMMENDED method for generating probes is to add a chunk consisting only of padding to an SCTP message. The PAD chunk defined in [RFC4820] SHOULD be attached to a minimum length HEARTBEAT (HB) chunk to build a probe packet. This enables probing without affecting the transfer of user messages and without being limited by congestion control or flow control. This is preferred to using DATA chunks (with padding as required) as path probes.

Section 6.9 of [RFC4960] describes dividing the user messages into data chunks sent by the PL when using SCTP. This notes that once an SCTP message has been sent, it cannot be re-segmented. [RFC4960] describes the method to retransmit data chunks when the MPS has reduced, and the use of IP fragmentation for this case. This is unchanged by this document.

6.2.1. SCTP/IPv4 and SCTP/IPv6

6.2.1.1. Initial Connectivity

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

6.2.1.2. Sending SCTP Probe Packets

Probe packets consist of an SCTP common header followed by a HEARTBEAT chunk and a PAD chunk. The PAD chunk is used to control the length of the probe packet. The HEARTBEAT chunk is used to trigger the sending of a HEARTBEAT ACK chunk. The reception of the HEARTBEAT ACK chunk acknowledges reception of a successful probe. A successful probe updates the association and path counters, but an unsuccessful probe is discounted (assumed to be a result of choosing too large a PLPMTU).

The SCTP sender needs to be able to determine the total size of a probe packet. The HEARTBEAT chunk could carry a Heartbeat Information parameter that includes, besides the information

suggested in [RFC4960], the probe size to help an implementation associate a HEARTBEAT-ACK with the size of probe that was sent. The sender could also use other methods, such as sending a nonce and verifying the information returned also contains the corresponding nonce. The length of the PAD chunk is computed by reducing the probing size by the size of the SCTP common header and the HEARTBEAT chunk. The payload of the PAD chunk contains arbitrary data. When transmitted at the IP layer, the PMTU size also includes the IPv4 or IPv6 header(s).

Probing can start directly after the PL handshake, this can be done before data is sent. Assuming this behavior (i.e., the PMTU is smaller than or equal to the interface MTU), this process will take several round trip time periods, dependent on the number of DPLPMTUD probes sent. The Heartbeat timer can be used to implement the PROBE_TIMER.

6.2.1.3. Validating the Path with SCTP

Since SCTP provides an acknowledged PL, a sender MUST NOT implement the CONFIRMATION_TIMER while in the SEARCH_COMPLETE state.

6.2.1.4. PTB Message Handling by SCTP

Normal ICMP validation MUST be performed as specified in Appendix C of [RFC4960]. This requires that the first 8 bytes of the SCTP common header are quoted in the payload of the PTB message, which can be the case for ICMPv4 and is normally the case for ICMPv6.

When a PTB message has been validated, the PL_PTB_SIZE calculated from the PTB_SIZE reported in the PTB message SHOULD be used with the DPLPMTUD algorithm, providing that the reported PL_PTB_SIZE is less than the current probe size (see Section 4.6).

6.2.2. DPLPMTUD for SCTP/UDP

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

This specification updates the reference to RFC 4821 in section 5.6 of RFC 6951 to refer to XXXTHISRFCXXX. RFC 6951 is updated by addition of the following sentence at the end of section 5.6: "The RECOMMENDED method for determining the MTU of the path is specified in XXXTHISRFCXXX".

XXX RFC EDITOR - please replace XXXTHISRFCXXX when published XXX

6.2.2.1. Initial Connectivity

A sender can enter the BASE state as soon as SCTP connectivity has been confirmed.

6.2.2.2. Sending SCTP/UDP Probe Packets

Packet probing can be performed as specified in Section 6.2.1.2. The size of the probe packet includes the 8 bytes of UDP Header. This has to be considered when filling the probe packet with the PAD chunk.

6.2.2.3. Validating the Path with SCTP/UDP

SCTP provides an acknowledged PL, therefore a sender does not implement the CONFIRMATION_TIMER while in the SEARCH_COMPLETE state.

6.2.2.4. Handling of PTB Messages by SCTP/UDP

ICMP validation MUST be performed for PTB messages as specified in Appendix C of [RFC4960]. This requires that the first 8 bytes of the SCTP common header are contained in the PTB message, which can be the case for ICMPv4 (but note the UDP header also consumes a part of the quoted packet header) and is normally the case for ICMPv6. When the validation is completed, the PL_PTB_SIZE calculated from the PTB_SIZE in the PTB message SHOULD be used with the DPLPMTUD providing that the reported PL_PTB_SIZE is less than the current probe size.

6.2.3. DPLPMTUD for SCTP/DTLS

The Datagram Transport Layer Security (DTLS) encapsulation of SCTP is specified in [RFC8261]. This is used for data channels in WebRTC implementations. This specification updates the reference to RFC 4821 in section 5 of RFC 8261 to refer to XXXTHISRFCXXX.

XXX RFC EDITOR - please replace XXXTHISRFCXXX when published XXX

6.2.3.1. Initial Connectivity

A sender can enter the BASE state as soon as SCTP connectivity has been confirmed.

6.2.3.2. Sending SCTP/DTLS Probe Packets

Packet probing can be done, as specified in Section 6.2.1.2. The maximum payload is reduced by the size of the DTLS headers, which has to be considered when filling the PAD chunk. The size of the probe packet includes the DTLS PL headers. This has to be considered when filling the probe packet with the PAD chunk.

6.2.3.3. Validating the Path with SCTP/DTLS

Since SCTP provides an acknowledged PL, a sender MUST NOT implement the CONFIRMATION_TIMER while in the SEARCH_COMPLETE state.

6.2.3.4. Handling of PTB Messages by SCTP/DTLS

[RFC4960] does not specify a way to validate SCTP/DTLS ICMP message payload and neither does this document. This can prevent processing of PTB messages at the PL.

6.3. DPLPMTUD for QUIC

QUIC [I-D.ietf-quic-transport] is a UDP-based PL that provides reception feedback. The UDP payload includes a QUIC packet header, a protected payload, and any authentication fields. It supports padding and packet coalescence that can be used to construct probe packets. From the perspective of DPLPMTUD, QUIC can function as an acknowledged PL. [I-D.ietf-quic-transport] describes the method for using DPLPMTUD with QUIC packets.

7. Acknowledgments

This work was partially funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644334 (NEAT). The views expressed are solely those of the author(s).

Thanks to all that have commented or contributed, the TSVWG and QUIC working groups, and Mathew Calder and Julius Flohr for providing early implementations.

8. IANA Considerations

This memo includes no request to IANA.

If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

9. Security Considerations

The security considerations for the use of UDP and SCTP are provided in the referenced RFCs.

To avoid excessive load, the interval between individual probe packets MUST be at least one RTT, and the interval between rounds of probing is determined by the PMTU_RAISE_TIMER.

A PL sender needs to ensure that the method used to confirm reception of probe packets protects from off-path attackers injecting packets into the path. This protection is provided in IETF-defined protocols (e.g., TCP, SCTP) using a randomly-initialized sequence number. A description of one way to do this when using UDP is provided in section 5.1 of [BCP145]).

There are cases where ICMP Packet Too Big (PTB) messages are not delivered due to policy, configuration or equipment design (see Section 1.1). This method therefore does not rely upon PTB messages being received, but is able to utilize these when they are received by the sender. PTB messages could potentially be used to cause a node to inappropriately reduce the PLPMTU. A node supporting DPLPMTUD MUST therefore appropriately validate the payload of PTB messages to ensure these are received in response to transmitted traffic (i.e., a reported error condition that corresponds to a datagram actually sent by the path layer, see Section 4.6.1).

An on-path attacker able to create a PTB message could forge PTB messages that include a valid quoted IP packet. Such an attack could be used to drive down the PLPMTU. An on-path device could similarly force a reduction of the PLPMTU by implementing a policy that drops packets larger than a configured size. There are two ways this method can be mitigated against such attacks: First, by ensuring that a PL sender never reduces the PLPMTU below the base size, solely in response to receiving a PTB message. This is achieved by first entering the BASE state when such a message is received. Second, the design does not require processing of PTB messages, a PL sender could therefore suspend processing of PTB messages (e.g., in a robustness mode after detecting that subsequent probes actually confirm that a size larger than the PTB_SIZE is supported by a path).

Parsing the quoted packet inside a PTB message can introduce additional per-packet processing at the PL sender. This processing SHOULD be limited to avoid a denial of service attack when arbitrary headers are included. Rate-limiting the processing could result in PTB messages not being received by a PL, however the DPLPMTUD method is robust to such loss.

The successful processing of an ICMP message can trigger a probe when the reported PTB size is valid, but this does not directly update the PLPMTU for the path. This prevents a message attempting to black hole data by indicating a size larger than supported by the path.

It is possible that the information about a path is not stable. This could be a result of forwarding across more than one path that has a different actual PMTU or a single path presents a varying PMTU. The design of a PLPMTUD implementation SHOULD consider how to mitigate the effects of varying path information. One possible mitigation is to provide robustness (see Section 5.4) in the method that avoids oscillation in the MPS.

DPLPMTUD methods can introduce padding data to inflate the length of the datagram to the total size required for a probe packet. The total size of a probe packet includes all headers and padding added to the payload data being sent (e.g., including security-related fields such as an AEAD tag and TLS record layer padding). The value of the padding data does not influence the DPLPMTUD search algorithm, and therefore needs to be set consistent with the policy of the PL.

If a PL can make use of cryptographic confidentiality or data-integrity mechanisms, then the design ought to avoid adding anything (e.g., padding) to DPLPMTUD probe packets that is not also protected by those cryptographic mechanisms.

10. References

10.1. Normative References

- [BCP145] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, March 2017.
[<https://www.rfc-editor.org/info/bcp145>](https://www.rfc-editor.org/info/bcp145)
- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980,
[<https://www.rfc-editor.org/info/rfc768>](https://www.rfc-editor.org/info/rfc768).
- [RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981,
[<https://www.rfc-editor.org/info/rfc791>](https://www.rfc-editor.org/info/rfc791).
- [RFC1191] Mogul, J.C. and S.E. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990,
[<https://www.rfc-editor.org/info/rfc1191>](https://www.rfc-editor.org/info/rfc1191).

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3828] Larzon, L-A., Degermark, M., Pink, S., Jonsson, L-E., Ed., and G. Fairhurst, Ed., "The Lightweight User Datagram Protocol (UDP-Lite)", RFC 3828, DOI 10.17487/RFC3828, July 2004, <<https://www.rfc-editor.org/info/rfc3828>>.
- [RFC4820] Tuexen, M., Stewart, R., and P. Lei, "Padding Chunk and Parameter for the Stream Control Transmission Protocol (SCTP)", RFC 4820, DOI 10.17487/RFC4820, March 2007, <<https://www.rfc-editor.org/info/rfc4820>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.
- [RFC6951] Tuexen, M. and R. Stewart, "UDP Encapsulation of Stream Control Transmission Protocol (SCTP) Packets for End-Host to End-Host Communication", RFC 6951, DOI 10.17487/RFC6951, May 2013, <<https://www.rfc-editor.org/info/rfc6951>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <<https://www.rfc-editor.org/info/rfc8200>>.
- [RFC8201] McCann, J., Deering, S., Mogul, J., and R. Hinden, Ed., "Path MTU Discovery for IP version 6", STD 87, RFC 8201, DOI 10.17487/RFC8201, July 2017, <<https://www.rfc-editor.org/info/rfc8201>>.
- [RFC8261] Tuexen, M., Stewart, R., Jesup, R., and S. Loreto, "Datagram Transport Layer Security (DTLS) Encapsulation of SCTP Packets", RFC 8261, DOI 10.17487/RFC8261, November 2017, <<https://www.rfc-editor.org/info/rfc8261>>.

10.2. Informative References

- [I-D.ietf-intarea-frag-fragile]
Bonica, R., Baker, F., Huston, G., Hinden, R., Troan, O.,

and F. Gont, "IP Fragmentation Considered Fragile", Work in Progress, Internet-Draft, draft-ietf-intarea-frag-fragile-17, 30 September 2019, <<http://www.ietf.org/internet-drafts/draft-ietf-intarea-frag-fragile-17.txt>>.

[I-D.ietf-intarea-tunnels]

Touch, J. and M. Townsley, "IP Tunnels in the Internet Architecture", Work in Progress, Internet-Draft, draft-ietf-intarea-tunnels-10, 12 September 2019, <<http://www.ietf.org/internet-drafts/draft-ietf-intarea-tunnels-10.txt>>.

[I-D.ietf-quic-transport]

Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-27, 21 February 2020, <<http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-27.txt>>.

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

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.

[RFC1812] Baker, F., Ed., "Requirements for IP Version 4 Routers", RFC 1812, DOI 10.17487/RFC1812, June 1995, <<https://www.rfc-editor.org/info/rfc1812>>.

[RFC2923] Lahey, K., "TCP Problems with Path MTU Discovery", RFC 2923, DOI 10.17487/RFC2923, September 2000, <<https://www.rfc-editor.org/info/rfc2923>>.

[RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI 10.17487/RFC4340, March 2006, <<https://www.rfc-editor.org/info/rfc4340>>.

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <<https://www.rfc-editor.org/info/rfc4443>>.

- [RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, DOI 10.17487/RFC4821, March 2007, <<https://www.rfc-editor.org/info/rfc4821>>.
- [RFC4890] Davies, E. and J. Mohacsi, "Recommendations for Filtering ICMPv6 Messages in Firewalls", RFC 4890, DOI 10.17487/RFC4890, May 2007, <<https://www.rfc-editor.org/info/rfc4890>>.
- [RFC5508] Srisuresh, P., Ford, B., Sivakumar, S., and S. Guha, "NAT Behavioral Requirements for ICMP", BCP 148, RFC 5508, DOI 10.17487/RFC5508, April 2009, <<https://www.rfc-editor.org/info/rfc5508>>.

Appendix A. Revision Notes

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

Individual draft -00:

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

Individual draft -01:

- * Contains the first representation of the algorithm, showing the states and timers
- * This update is proposed for WG comments.

Individual draft -02:

- * Contains updated representation of the algorithm, and textual corrections.
- * The text describing when to set the effective PMTU has not yet been validated by the authors
- * To determine security to off-path-attacks: We need to decide whether a received PTB message SHOULD/MUST be validated? The text on how to handle a PTB message indicating a link MTU larger than the probe has yet not been validated by the authors
- * No text currently describes how to handle inconsistent results from arbitrary re-routing along different parallel paths

- * This update is proposed for WG comments.

Working Group draft -00:

- * This draft follows a successful adoption call for TSVWG
- * There is still work to complete, please comment on this draft.

Working Group draft -01:

- * This draft includes improved introduction.
- * The draft is updated to require ICMP validation prior to accepting PTB messages - this to be confirmed by WG
- * Section added to discuss Selection of Probe Size - methods to be evaluated and recommendations to be considered
- * Section added to align with work proposed in the QUIC WG.

Working Group draft -02:

- * The draft was updated based on feedback from the WG, and a detailed review by Magnus Westerlund.
- * The document updates RFC 4821.
- * Requirements list updated.
- * Added more explicit discussion of a simpler black-hole detection mode.
- * This draft includes reorganisation of the section on IETF protocols.
- * Added more discussion of implementation within an application.
- * Added text on flapping paths.
- * Replaced 'effective MTU' with new term PLPMTU.

Working Group draft -03:

- * Updated figures
- * Added more discussion on blackhole detection
- * Added figure describing just blackhole detection

- * Added figure relating MPS sizes

Working Group draft -04:

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

Working Group draft -05:

- * Updated security considerations.
- * Feedback after speaking with Joe Touch helped improve UDP-Options description.

Working Group draft -06:

- * Updated description of ICMP issues in section 1.1
- * Update to description of QUIC.

Working group draft -07:

- * Moved description of the PTB processing method from the PTB requirements section.
- * Clarified what is performed in the PTB validation check.
- * Updated security consideration to explain PTB security without needing to read the rest of the document.

- * Reformatted state machine diagram

Working group draft -08:

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

Working group draft -09:

- * Remove final mention of UDP Options
- * Add Initial Connectivity sections to each PL
- * Add to disable outgoing pmtu enforcement of packets

Working group draft -10:

- * Address comments from Lars Eggert
- * Reinforce that PROBE_COUNT is successive attempts to probe for any size
- * Redefine MAX_PROBES to 3
- * Address PTB_SIZE of 0 or less than MIN_PLPMTU

Working group draft -11:

- * Restore a sentence removed in previous rev
- * De-acronymise QUIC
- * Address some nits

Working group draft -12:

- * Add TSVWG, QUIC and implementers to acknowledgments
- * Shorten a diagram line.
- * Address nits from Julius and Wes.

- * Be clearer when talking about IP layer caches

Working group draft -13, -14:

- * Updated after WGLC.

Working group draft -15:

- * Updated after AD evaluation and prepared for IETF-LC.

Working group draft -16:

- * Updated text after SECDIR review.

Working group draft -17:

- * Updated text after GENART and IETF-LC.

- * Renamed BASE_MTU to BASE_PLPMTU, and MIN and MAX PMTU to PLPMTU (because these are about a base for the PLPMTU), and ensured consistent separation of PMTU and PLPMTU.

- * Adopted US-style English throughout.

Working group draft -18:

- * Updated text and address nits from OPSDIR, ART and IESG reviews.

- * Order PTB processing based on PL_PTB_SIZE

Working group draft -19:

- * Updated text and address nits based on comments from Tim Chown and Murray S. Kucherawy.

Working group draft -20:

- * Address nits and comments from IESG

- * Refer to BCP 145 rather than RFC 8085 in most places.

- * Update probing method text for SCTP and QUIC.

Working group draft -21:

- * Update QUIC text for skipping into BASE state.

Working group draft -22:

- * Add a section reference to MPS
- * Clarify MIN_PLPMTU text
- * Remove most QUIC text
- * Make QUIC reference informative.

Authors' Addresses

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

Email: gorry@erg.abdn.ac.uk

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

Email: tom@erg.abdn.ac.uk

Michael Tuexen
Muenster University of Applied Sciences
Stegerwaldstrasse 39
48565 Steinfurt
Germany

Email: tuexen@fh-muenster.de

Irene Ruengeler
Muenster University of Applied Sciences
Stegerwaldstrasse 39
48565 Steinfurt
Germany

Email: i.ruengeler@fh-muenster.de

Timo Voelker
Muenster University of Applied Sciences
Stegerwaldstrasse 39
48565 Steinfurt
Germany

Email: timo.voelker@fh-muenster.de

Transport Services (tsv)
Internet-Draft
Intended status: Experimental
Expires: 5 September 2022

K. De Schepper
Nokia Bell Labs
B. Briscoe, Ed.
Independent
4 March 2022

Explicit Congestion Notification (ECN) Protocol for Very Low Queuing
Delay (L4S)
draft-ietf-tsvwg-ecn-l4s-id-25

Abstract

This specification defines the protocol to be used for a new network service called low latency, low loss and scalable throughput (L4S). L4S uses an Explicit Congestion Notification (ECN) scheme at the IP layer that is similar to the original (or 'Classic') ECN approach, except as specified within. L4S uses 'scalable' congestion control, which induces much more frequent control signals from the network and it responds to them with much more fine-grained adjustments, so that very low (typically sub-millisecond on average) and consistently low queuing delay becomes possible for L4S traffic without compromising link utilization. Thus even capacity-seeking (TCP-like) traffic can have high bandwidth and very low delay at the same time, even during periods of high traffic load.

The L4S identifier defined in this document distinguishes L4S from 'Classic' (e.g. TCP-Reno-friendly) traffic. It gives an incremental migration path so that suitably modified network bottlenecks can distinguish and isolate existing traffic that still follows the Classic behaviour, to prevent it degrading the low queuing delay and low loss of L4S traffic. This specification defines the rules that L4S transports and network elements need to follow with the intention that L4S flows neither harm each other's performance nor that of Classic traffic. Examples of new active queue management (AQM) marking algorithms and examples of new transports (whether TCP-like or real-time) are specified separately.

Status of This Memo

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

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

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

This Internet-Draft will expire on 5 September 2022.

Copyright Notice

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

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

Table of Contents

1. Introduction	4
1.1. Latency, Loss and Scaling Problems	5
1.2. Terminology	7
1.3. Scope	9
2. Choice of L4S Packet Identifier: Requirements	10
3. L4S Packet Identification	11
4. Transport Layer Behaviour (the 'Prague Requirements')	11
4.1. Codepoint Setting	12
4.2. Prerequisite Transport Feedback	12
4.3. Prerequisite Congestion Response	13
4.3.1. Guidance on Congestion Response in the RFC Series	16
4.4. Filtering or Smoothing of ECN Feedback	19
5. Network Node Behaviour	19
5.1. Classification and Re-Marking Behaviour	19
5.2. The Strength of L4S CE Marking Relative to Drop	21
5.3. Exception for L4S Packet Identification by Network Nodes with Transport-Layer Awareness	22
5.4. Interaction of the L4S Identifier with other Identifiers	22
5.4.1. DualQ Examples of Other Identifiers Complementing L4S Identifiers	22
5.4.1.1. Inclusion of Additional Traffic with L4S	22
5.4.1.2. Exclusion of Traffic From L4S Treatment	24
5.4.1.3. Generalized Combination of L4S and Other Identifiers	25

5.4.2.	Per-Flow Queuing Examples of Other Identifiers Complementing L4S Identifiers	27
5.5.	Limiting Packet Bursts from Links	27
5.5.1.	Limiting Packet Bursts from Links Fed by an L4S AQM	27
5.5.2.	Limiting Packet Bursts from Links Upstream of an L4S AQM	28
6.	Behaviour of Tunnels and Encapsulations	28
6.1.	No Change to ECN Tunnels and Encapsulations in General	28
6.2.	VPN Behaviour to Avoid Limitations of Anti-Replay	29
7.	L4S Experiments	30
7.1.	Open Questions	30
7.2.	Open Issues	32
7.3.	Future Potential	32
8.	IANA Considerations	33
9.	Security Considerations	33
10.	Acknowledgements	34
11.	References	35
11.1.	Normative References	35
11.2.	Informative References	35
Appendix A.	Rationale for the 'Prague L4S Requirements'	45
A.1.	Rationale for the Requirements for Scalable Transport Protocols	46
A.1.1.	Use of L4S Packet Identifier	46
A.1.2.	Accurate ECN Feedback	46
A.1.3.	Capable of Replacement by Classic Congestion Control	46
A.1.4.	Fall back to Classic Congestion Control on Packet Loss	47
A.1.5.	Coexistence with Classic Congestion Control at Classic ECN bottlenecks	48
A.1.6.	Reduce RTT dependence	51
A.1.7.	Scaling down to fractional congestion windows	52
A.1.8.	Measuring Reordering Tolerance in Time Units	53
A.2.	Scalable Transport Protocol Optimizations	56
A.2.1.	Setting ECT in Control Packets and Retransmissions	56
A.2.2.	Faster than Additive Increase	56
A.2.3.	Faster Convergence at Flow Start	57
Appendix B.	Compromises in the Choice of L4S Identifier	57
Appendix C.	Potential Competing Uses for the ECT(1) Codepoint	62
C.1.	Integrity of Congestion Feedback	62
C.2.	Notification of Less Severe Congestion than CE	63
Authors' Addresses	64

1. Introduction

This specification defines the protocol to be used for a new network service called low latency, low loss and scalable throughput (L4S). L4S uses an Explicit Congestion Notification (ECN) scheme at the IP layer with the same set of codepoint transitions as the original (or 'Classic') Explicit Congestion Notification (ECN [RFC3168]). RFC 3168 required an ECN mark to be equivalent to a drop, both when applied in the network and when responded to by a transport. Unlike Classic ECN marking, the network applies L4S marking more immediately and more aggressively than drop, and the transport response to each mark is reduced and smoothed relative to that for drop. The two changes counterbalance each other so that the throughput of an L4S flow will be roughly the same as a comparable non-L4S flow under the same conditions. Nonetheless, the much more frequent ECN control signals and the finer responses to these signals result in very low queuing delay without compromising link utilization, and this low delay can be maintained during high load. For instance, queuing delay under heavy and highly varying load with the example DCTCP/DualQ solution cited below on a DSL or Ethernet link is sub-millisecond on average and roughly 1 to 2 milliseconds at the 99th percentile without losing link utilization [DualPI2Linux], [DCTtH19]. Note that the inherent queuing delay while waiting to acquire a discontinuous medium such as WiFi has to be minimized in its own right, so it would be additional to the above (see section 6.3 of the L4S architecture [I-D.ietf-tsvwg-l4s-arch]).

L4S relies on 'scalable' congestion controls for these delay properties and for preserving low delay as flow rate scales, hence the name. The congestion control used in Data Center TCP (DCTCP) is an example of a scalable congestion control, but DCTCP is applicable solely to controlled environments like data centres [RFC8257], because it is too aggressive to co-exist with existing TCP-Reno-friendly traffic. The DualQ Coupled AQM, which is defined in a complementary experimental specification [I-D.ietf-tsvwg-aqm-dualq-coupled], is an AQM framework that enables scalable congestion controls derived from DCTCP to co-exist with existing traffic, each getting roughly the same flow rate when they compete under similar conditions. Note that a scalable congestion control is still not safe to deploy on the Internet unless it satisfies the requirements listed in Section 4.

L4S is not only for elastic (TCP-like) traffic - there are scalable congestion controls for real-time media, such as the L4S variant of the SCReAM [RFC8298] real-time media congestion avoidance technique (RMCAT). The factor that distinguishes L4S from Classic traffic is its behaviour in response to congestion. The transport wire protocol, e.g. TCP, QUIC, SCTP, DCCP, RTP/RTCP, is orthogonal (and therefore not suitable for distinguishing L4S from Classic packets).

The L4S identifier defined in this document is the key piece that distinguishes L4S from 'Classic' (e.g. Reno-friendly) traffic. It gives an incremental migration path so that suitably modified network bottlenecks can distinguish and isolate existing Classic traffic from L4S traffic to prevent the former from degrading the very low delay and loss of the new scalable transports, without harming Classic performance at these bottlenecks. Initial implementation of the separate parts of the system has been motivated by the performance benefits.

1.1. Latency, Loss and Scaling Problems

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

The Diffserv architecture provides Expedited Forwarding [RFC3246], so that low latency traffic can jump the queue of other traffic. If growth in high-throughput latency-sensitive applications continues, periods with solely latency-sensitive traffic will become increasingly common on links where traffic aggregation is low. For instance, on the access links dedicated to individual sites (homes, small enterprises or mobile devices). These links also tend to become the path bottleneck under load. During these periods, if all the traffic were marked for the same treatment, at these bottlenecks Diffserv would make no difference. Instead, it becomes imperative to remove the underlying causes of any unnecessary delay.

The bufferbloat project has shown that excessively-large buffering ('bufferbloat') has been introducing significantly more delay than the underlying propagation time. These delays appear only intermittently -- only when a capacity-seeking (e.g. TCP) flow is long enough for the queue to fill the buffer, making every packet in other flows sharing the buffer sit through the queue.

Active queue management (AQM) was originally developed to solve this problem (and others). Unlike Diffserv, which gives low latency to some traffic at the expense of others, AQM controls latency for all traffic in a class. In general, AQM methods introduce an increasing level of discard from the buffer the longer the queue persists above a shallow threshold. This gives sufficient signals to capacity-seeking (aka. greedy) flows to keep the buffer empty for its intended purpose: absorbing bursts. However, RED [RFC2309] and other algorithms from the 1990s were sensitive to their configuration and hard to set correctly. So, this form of AQM was not widely deployed.

More recent state-of-the-art AQM methods, e.g. FQ-CoDel [RFC8290], PIE [RFC8033], Adaptive RED [ARED01], are easier to configure, because they define the queuing threshold in time not bytes, so it is invariant for different link rates. However, no matter how good the AQM, the sawtooth sending window of a Classic congestion control will either cause queuing delay to vary or cause the link to be underutilized. Even with a perfectly tuned AQM, the additional queuing delay will be of the same order as the underlying speed-of-light delay across the network, thereby roughly doubling the total round-trip time.

If a sender's own behaviour is introducing queuing delay variation, no AQM in the network can 'un-vary' the delay without significantly compromising link utilization. Even flow-queuing (e.g. [RFC8290]), which isolates one flow from another, cannot isolate a flow from the delay variations it inflicts on itself. Therefore those applications that need to seek out high bandwidth but also need low latency will have to migrate to scalable congestion control.

Altering host behaviour is not enough on its own though. Even if hosts adopt low latency behaviour (scalable congestion controls), they need to be isolated from the behaviour of existing Classic congestion controls that induce large queue variations. L4S enables that migration by providing latency isolation in the network and distinguishing the two types of packets that need to be isolated: L4S and Classic. L4S isolation can be achieved with a queue per flow (e.g. [RFC8290]) but a DualQ [I-D.ietf-tsvwg-aqm-dualq-coupled] is sufficient, and actually gives better tail latency. Both approaches are addressed in this document.

The DualQ solution was developed to make very low latency available without requiring per-flow queues at every bottleneck. This was because per-flow-queuing (FQ) has well-known downsides - not least the need to inspect transport layer headers in the network, which makes it incompatible with privacy approaches such as IPsec VPN tunnels, and incompatible with link layer queue management, where transport layer headers can be hidden, e.g. 5G.

Latency is not the only concern addressed by L4S: It was known when TCP congestion avoidance was first developed that it would not scale to high bandwidth-delay products (footnote 6 of Jacobson and Karels [TCP-CA]). Given regular broadband bit-rates over WAN distances are already [RFC3649] beyond the scaling range of Reno congestion control, 'less unscalable' Cubic [RFC8312] and Compound [I-D.sridharan-tcpm-ctcp] variants of TCP have been successfully deployed. However, these are now approaching their scaling limits. Unfortunately, fully scalable congestion controls such as DCTCP [RFC8257] outcompete Classic ECN congestion controls sharing the same queue, which is why they have been confined to private data centres or research testbeds.

It turns out that these scalable congestion control algorithms that solve the latency problem can also solve the scalability problem of Classic congestion controls. The finer sawteeth in the congestion window have low amplitude, so they cause very little queuing delay variation and the average time to recover from one congestion signal to the next (the average duration of each sawtooth) remains invariant, which maintains constant tight control as flow-rate scales. A background paper [DCttH19] gives the full explanation of why the design solves both the latency and the scaling problems, both in plain English and in more precise mathematical form. The explanation is summarised without the maths in Section 4 of the L4S architecture [I-D.ietf-tsvwg-l4s-arch].

1.2. Terminology

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

Note: The L4S architecture [I-D.ietf-tsvwg-l4s-arch] repeats the following definitions, but if there are accidental differences those below take precedence.

Classic Congestion Control: A congestion control behaviour that can

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

Scalable Congestion Control: A congestion control where the average time from one congestion signal to the next (the recovery time) remains invariant as the flow rate scales, all other factors being equal. This maintains the same degree of control over queueing and utilization whatever the flow rate, as well as ensuring that high throughput is robust to disturbances. For instance, DCTCP averages 2 congestion signals per round-trip whatever the flow rate, as do other recently developed scalable congestion controls, e.g. Relentless TCP [Mathis09], TCP Prague [I-D.briscoe-iccrp-prague-congestion-control], [PragueLinux], BBRv2 [BBRv2], [I-D.cardwell-iccrp-bbr-congestion-control] and the L4S variant of SCREAM for real-time media [SCReAM], [RFC8298]). See Section 4.3 for more explanation.

Classic service: The Classic service is intended for all the congestion control behaviours that co-exist with Reno [RFC5681] (e.g. Reno itself, Cubic [RFC8312], Compound [I-D.sridharan-tcpm-ctcp], TFRC [RFC5348]). The term 'Classic queue' means a queue providing the Classic service.

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

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

Both Classic and L4S services can cope with a proportion of unresponsive or less-responsive traffic as well, but in the L4S case its rate has to be smooth enough or low enough not to build a queue (e.g. DNS, VoIP, game sync datagrams, etc).

Reno-friendly: The subset of Classic traffic that is friendly to the standard Reno congestion control defined for TCP in [RFC5681]. The TFRC spec. [RFC5348] indirectly implies that 'friendly' is defined as "generally within a factor of two of the sending rate of a TCP flow under the same conditions". Reno-friendly is used here in place of 'TCP-friendly', given the latter has become imprecise, because the TCP protocol is now used with so many different congestion control behaviours, and Reno is used in non-TCP transports such as QUIC [RFC9000].

Classic ECN: The original Explicit Congestion Notification (ECN) protocol [RFC3168], which requires ECN signals to be treated the same as drops, both when generated in the network and when responded to by the sender. For L4S, the names used for the four codepoints of the 2-bit IP-ECN field are unchanged from those defined in [RFC3168]: Not ECT, ECT(0), ECT(1) and CE, where ECT stands for ECN-Capable Transport and CE stands for Congestion Experienced. A packet marked with the CE codepoint is termed 'ECN-marked' or sometimes just 'marked' where the context makes ECN obvious.

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

1.3. Scope

The new L4S identifier defined in this specification is applicable for IPv4 and IPv6 packets (as for Classic ECN [RFC3168]). It is applicable for the unicast, multicast and anycast forwarding modes.

The L4S identifier is an orthogonal packet classification to the Differentiated Services Code Point (DSCP) [RFC2474]. Section 5.4 explains what this means in practice.

This document is intended for experimental status, so it does not update any standards track RFCs. Therefore it depends on [RFC8311], which is a standards track specification that:

- * updates the ECN proposed standard [RFC3168] to allow experimental track RFCs to relax the requirement that an ECN mark must be equivalent to a drop (when the network applies markings and/or

when the sender responds to them). For instance, in the ABE experiment [RFC8511] this permits a sender to respond less to ECN marks than to drops;

- * changes the status of the experimental ECN nonce [RFC3540] to historic;
- * makes consequent updates to the following additional proposed standard RFCs to reflect the above two bullets:
 - ECN for RTP [RFC6679];
 - the congestion control specifications of various DCCP congestion control identifier (CCID) profiles [RFC4341], [RFC4342], [RFC5622].

This document is about identifiers that are used for interoperation between hosts and networks. So the audience is broad, covering developers of host transports and network AQMs, as well as covering how operators might wish to combine various identifiers, which would require flexibility from equipment developers.

2. Choice of L4S Packet Identifier: Requirements

This subsection briefly records the process that led to the chosen L4S identifier.

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

- * it SHOULD survive end-to-end between source and destination endpoints: across the boundary between host and network, between interconnected networks, and through middleboxes;
- * it SHOULD be visible at the IP layer;
- * it SHOULD be common to IPv4 and IPv6 and transport-agnostic;
- * it SHOULD be incrementally deployable;
- * it SHOULD enable an AQM to classify packets encapsulated by outer IP or lower-layer headers;
- * it SHOULD consume minimal extra codepoints;
- * it SHOULD be consistent on all the packets of a transport layer flow, so that some packets of a flow are not served by a different queue to others.

Whether the identifier would be recoverable if the experiment failed is a factor that could be taken into account. However, this has not been made a requirement, because that would favour schemes that would be easier to fail, rather than those more likely to succeed.

It is recognised that any choice of identifier is unlikely to satisfy all these requirements, particularly given the limited space left in the IP header. Therefore a compromise will always be necessary, which is why all the above requirements are expressed with the word 'SHOULD' not 'MUST'.

After extensive assessment of alternative schemes, "ECT(1) and CE codepoints" was chosen as the best compromise. Therefore this scheme is defined in detail in the following sections, while Appendix B records its pros and cons against the above requirements.

3. L4S Packet Identification

The L4S treatment is an experimental track alternative packet marking treatment to the Classic ECN treatment in [RFC3168], which has been updated by [RFC8311] to allow experiments such as the one defined in the present specification. [RFC4774] discusses some of the issues and evaluation criteria when defining alternative ECN semantics. Like Classic ECN, L4S ECN identifies both network and host behaviour: it identifies the marking treatment that network nodes are expected to apply to L4S packets, and it identifies packets that have been sent from hosts that are expected to comply with a broad type of sending behaviour.

For a packet to receive L4S treatment as it is forwarded, the sender sets the ECN field in the IP header to the ECT(1) codepoint. See Section 4 for full transport layer behaviour requirements, including feedback and congestion response.

A network node that implements the L4S service always classifies arriving ECT(1) packets for L4S treatment and by default classifies CE packets for L4S treatment unless the heuristics described in Section 5.3 are employed. See Section 5 for full network element behaviour requirements, including classification, ECN-marking and interaction of the L4S identifier with other identifiers and per-hop behaviours.

4. Transport Layer Behaviour (the 'Prague Requirements')

4.1. Codepoint Setting

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

4.2. Prerequisite Transport Feedback

For a transport protocol to provide scalable congestion control (Section 4.3) it MUST provide feedback of the extent of CE marking on the forward path. When ECN was added to TCP [RFC3168], the feedback method reported no more than one CE mark per round trip. Some transport protocols derived from TCP mimic this behaviour while others report the accurate extent of ECN marking. This means that some transport protocols will need to be updated as a prerequisite for scalable congestion control. The position for a few well-known transport protocols is given below.

TCP: Support for the accurate ECN feedback requirements [RFC7560] (such as that provided by AccECN [I-D.ietf-tcpm-accurate-ecn]) by both ends is a prerequisite for scalable congestion control in TCP. Therefore, the presence of ECT(1) in the IP headers even in one direction of a TCP connection will imply that both ends support accurate ECN feedback. However, the converse does not apply. So even if both ends support AccECN, either of the two ends can choose not to use a scalable congestion control, whatever the other end's choice.

SCTP: A suitable ECN feedback mechanism for SCTP could add a chunk to report the number of received CE marks (e.g. [I-D.stewart-tsvwg-sctpecn]), and update the ECN feedback protocol sketched out in Appendix A of the original standards track specification of SCTP [RFC4960].

RTP over UDP: A prerequisite for scalable congestion control is for both (all) ends of one media-level hop to signal ECN support [RFC6679] and use the new generic RTCP feedback format of [RFC8888]. The presence of ECT(1) implies that both (all) ends of that media-level hop support ECN. However, the converse does not apply. So each end of a media-level hop can independently choose not to use a scalable congestion control, even if both ends support ECN.

QUIC: Support for sufficiently fine-grained ECN feedback is provided by the v1 IETF QUIC transport [RFC9000].

DCCP: The ACK vector in DCCP [RFC4340] is already sufficient to

report the extent of CE marking as needed by a scalable congestion control.

4.3. Prerequisite Congestion Response

As a condition for a host to send packets with the L4S identifier (ECT(1)), it SHOULD implement a congestion control behaviour that ensures that, in steady state, the average duration between induced ECN marks does not increase as flow rate scales up, all other factors being equal. This is termed a scalable congestion control. This invariant duration ensures that, as flow rate scales, the average period with no feedback information about capacity does not become excessive. It also ensures that queue variations remain small, without having to sacrifice utilization.

With a congestion control that sawtooths to probe capacity, this duration is called the recovery time, because each time the sawtooth yields, on average it takes this time to recover to its previous high point. A scalable congestion control does not have to sawtooth, but it has to coexist with scalable congestion controls that do.

For instance, for DCTCP [RFC8257], TCP Prague [I-D.briscoe-iccrp-prague-congestion-control], [PragueLinux] and the L4S variant of SCReAM [RFC8298], the average recovery time is always half a round trip (or half a reference round trip), whatever the flow rate.

As with all transport behaviours, a detailed specification (probably an experimental RFC) is expected for each congestion control, following the guidelines for specifying new congestion control algorithms in [RFC5033]. In addition it is expected to document these L4S-specific matters, specifically the timescale over which the proportionality is averaged, and control of burstiness. The recovery time requirement above is worded as a 'SHOULD' rather than a 'MUST' to allow reasonable flexibility for such implementations.

The condition 'all other factors being equal', allows the recovery time to be different for different round trip times, as long as it does not increase with flow rate for any particular RTT.

Saying that the recovery time remains roughly invariant is equivalent to saying that the number of ECN CE marks per round trip remains invariant as flow rate scales, all other factors being equal. For instance, an average recovery time of half of 1 RTT is equivalent to 2 ECN marks per round trip. For those familiar with steady-state congestion response functions, it is also equivalent to say that the congestion window is inversely proportional to the proportion of bytes in packets marked with the CE codepoint (see section 2 of [PI2]).

In order to coexist safely with other Internet traffic, a scalable congestion control MUST NOT tag its packets with the ECT(1) codepoint unless it complies with the following bulleted requirements:

1. A scalable congestion control MUST be capable of being replaced by a Classic congestion control (by application and/or by administrative control). If a Classic congestion control is activated, it will not tag its packets with the ECT(1) codepoint (see Appendix A.1.3 for rationale).
2. As well as responding to ECN markings, a scalable congestion control MUST react to packet loss in a way that will coexist safely with Classic congestion controls such as standard Reno [RFC5681], as required by [RFC5033] (see Appendix A.1.4 for rationale).
3. In uncontrolled environments, monitoring MUST be implemented to support detection of problems with an ECN-capable AQM at the path bottleneck that appears not to support L4S and might be in a shared queue. Such monitoring SHOULD be applied to live traffic that is using Scalable congestion control. Alternatively, monitoring need not be applied to live traffic, if monitoring has been arranged to cover the paths that live traffic takes through uncontrolled environments.

A function to detect the above problems with an ECN-capable AQM MUST also be implemented and used. The detection function SHOULD be capable of making the congestion control adapt its ECN-marking response in real-time to coexist safely with Classic congestion controls such as standard Reno [RFC5681], as required by [RFC5033]. This could be complemented by more detailed offline detection of potential problems. If only offline detection is used and potential problems with such an AQM are detected on certain paths, the scalable congestion control MUST be replaced by a Classic congestion control, at least for the problem paths.

See Section 4.3.1, Appendix A.1.5 and the L4S operational guidance [I-D.ietf-tsvwg-l4sops] for rationale.

Note that a scalable congestion control is not expected to change to setting ECT(0) while it transiently adapts to coexist with Classic congestion controls, whereas a replacement congestion control that solely behaves in the Classic way will set ECT(0).

4. In the range between the minimum likely RTT and typical RTTs expected in the intended deployment scenario, a scalable congestion control MUST converge towards a rate that is as independent of RTT as is possible without compromising stability or efficiency (see Appendix A.1.6 for rationale).
5. A scalable congestion control SHOULD remain responsive to congestion when typical RTTs over the public Internet are significantly smaller because they are no longer inflated by queuing delay. It would be preferable for the minimum window of a scalable congestion control to be lower than 1 segment rather than use the timeout approach described for TCP in S.6.1.2 of the ECN spec [RFC3168] (or an equivalent for other transports). However, a lower minimum is not set as a formal requirement for L4S experiments (see Appendix A.1.7 for rationale).
6. A scalable congestion control's loss detection SHOULD be resilient to reordering over an adaptive time interval that scales with throughput and adapts to reordering (as in RACK [RFC8985]), as opposed to counting only in fixed units of packets (as in the 3 DupACK rule of New Reno [RFC5681] and [RFC6675], which is not scalable). As data rates increase (e.g., due to new and/or improved technology), congestion controls that detect loss by counting in units of packets become more likely to incorrectly treat reordering events as congestion-caused loss events (see Appendix A.1.8 for further rationale). This requirement does not apply to congestion controls that are solely used in controlled environments where the network introduces hardly any reordering.
7. A scalable congestion control is expected to limit the queue caused by bursts of packets. It would not seem necessary to set the limit any lower than 10% of the minimum RTT expected in a typical deployment (e.g. additional queuing of roughly 250 us for the public Internet). This would be converted to a number of packets under the worst-case assumption that the bottleneck link capacity equals the current flow rate. No normative requirement to limit bursts is given here and, until there is more industry experience from the L4S experiment, it is not even known whether one is needed - it seems to be in an L4S sender's self-interest to limit bursts.

Each sender in a session can use a scalable congestion control independently of the congestion control used by the receiver(s) when they send data. Therefore there might be ECT(1) packets in one direction and ECT(0) or Not-ECT in the other.

Later (Section 5.4.1.1) this document discusses the conditions for mixing other "'Safe' Unresponsive Traffic" (e.g. DNS, LDAP, NTP, voice, game sync packets) with L4S traffic. To be clear, although such traffic can share the same queue as L4S traffic, it is not appropriate for the sender to tag it as ECT(1), except in the (unlikely) case that it satisfies the above conditions.

4.3.1. Guidance on Congestion Response in the RFC Series

RFC 3168 requires the congestion responses to a CE-marked packet and a dropped packet to be the same. RFC 8311 is a standards-track update to RFC 3168 intended to enable experimentation with ECN, including the L4S experiment. RFC 8311 allows an experimental congestion control's response to a CE-marked packet to differ from the response to a dropped packet, provided that the differences are documented in an experimental RFC, such as the present document.

BCP 124 [RFC4774] gives guidance to protocol designers, when specifying alternative semantics for the ECN field. RFC 8311 explained that it did not need to update the best current practice in BCP 124 in order to relax the 'equivalence with drop' requirement because, although BCP 124 quotes the same requirement from RFC 3168, the BCP does not impose requirements based on it. BCP 124 describes three options for incremental deployment, with Option 3 (in Section 4.3 of BCP 124) best matching the L4S case. Option 3's requirement for end-nodes is that they respond to CE marks "in a way that is friendly to flows using IETF-conformant congestion control." This echoes other general congestion control requirements in the RFC series, for example [RFC5033], which says "...congestion controllers that have a significantly negative impact on traffic using standard congestion control may be suspect", or [RFC8085] concerning UDP congestion control says "Bulk-transfer applications that choose not to implement TFRC or TCP-like windowing SHOULD implement a congestion control scheme that results in bandwidth (capacity) use that competes fairly with TCP within an order of magnitude."

The third normative bullet in Section 4.3 above (which concerns L4S response to congestion from a Classic ECN AQM) aims to ensure that these 'coexistence' requirements are satisfied, but it makes some compromises. This subsection highlights and justifies those compromises and Appendix A.1.5 and the L4S operational guidance [I-D.ietf-tsvwg-l4sops] give detailed analysis, examples and references (the normative text in that bullet takes precedence if any

informative elaboration leads to ambiguity). The approach is based on an assessment of the risk of harm, which is a combination of the prevalence of the conditions necessary for harm to occur, and the potential severity of the harm if they do.

Prevalence: There are three cases:

- * Drop Tail: Coexistence between L4S and Classic flows is not in doubt where the bottleneck does not support any form of ECN, which has remained by far the most prevalent case since the ECN RFC was published in 2001.
- * L4S: Coexistence is not in doubt if the bottleneck supports L4S.
- * Classic ECN [RFC3168]: The compromises centre around cases where the bottleneck supports Classic ECN but not L4S. But it depends on which sub-case:
 - Shared Queue with Classic ECN: The members of the Transport Working group are not aware of any current deployments of single-queue Classic ECN bottlenecks in the Internet. Nonetheless, at the scale of the Internet, rarity need not imply small numbers, nor that there will be rarity in future.
 - Per-Flow-queues with Classic ECN: Most AQMs with per-flow-queuing (FQ) deployed from 2012 onwards had Classic ECN enabled by default, specifically FQ-CoDel [RFC8290] and COBALT [COBALT]. But the compromises only apply to the second of two further sub-cases:
 - o With per-flow-queuing, co-existence between Classic and L4S flows is not normally a problem, because different flows are not meant to be in the same queue (BCP 124 [RFC4774] did not foresee the introduction of per-flow-queuing, which appeared as a potential isolation technique some eight years after the BCP was published).
 - o However, the isolation between L4S and Classic flows is not perfect in cases where the hashes of flow IDs collide or where multiple flows within a layer-3 VPN are encapsulated within one flow ID.

To summarize, the coexistence problem is confined to cases of imperfect flow isolation in an FQ, or in potential cases where a Classic ECN AQM has been deployed in a shared queue (see the L4S operational guidance [I-D.ietf-tsvwg-l4sops] for further details

including recent surveys attempting to quantify prevalence). Further, if one of these cases does occur, the coexistence problem does not arise unless sources of Classic and L4S flows are simultaneously sharing the same bottleneck queue (e.g. different applications in the same household) and flows of each type have to be large enough to coincide for long enough for any throughput imbalance to have developed.

Severity: Where long-running L4S and Classic flows coincide in a shared queue, testing of one L4S congestion control (TCP Prague) has found that the imbalance in average throughput between an L4S and a Classic flow can reach 25:1 in favour of L4S in the worst case [ecn-fallback]. However, when capacity is most scarce, the Classic flow gets a higher proportion of the link, for instance over a 4 Mb/s link the throughput ratio is below ~10:1 over paths with a base RTT below 100 ms, and falls below ~5:1 for base RTTs below 20ms.

These throughput ratios can clearly fall well outside current RFC guidance on coexistence. However, the tendency towards leaving a greater share for Classic flows at lower link rate and the very limited prevalence of the conditions necessary for harm to occur led to the possibility of allowing the RFC requirements to be compromised, albeit briefly:

- * The recommended approach is still to detect and adapt to a Classic ECN AQM in real-time, which is fully consistent with all the RFCs on coexistence. In other words, the "SHOULD"s in the third bullet of Section 4.3 above expect the sender to implement something similar to the proof of concept code that detects the presence of a Classic ECN AQM and falls back to a Classic congestion response within a few round trips [ecn-fallback]. However, although this code reliably detects a Classic ECN AQM, the current code can also wrongly categorize an L4S AQM as Classic, most often in cases when link rate is low or RTT is high. Although this is the safe way round, and although implementers are expected to be able to improve on this proof of concept, concerns have been raised that implementers might lose faith in such detection and disable it.
- * Therefore the third bullet in Section 4.3 above allows a compromise where coexistence could diverge from the requirements in the RFC Series briefly, but mandatory monitoring is required, in order to detect such cases and trigger remedial action. This approach tolerates a brief divergence from the RFCs given the likely low prevalence and given harm here means a flow progresses more slowly than otherwise, but it does progress. The L4S operational guidance [I-D.ietf-tsvwg-l4sops] outlines a range of example remedial actions that include alterations either to the

sender or to the network. However, the final normative requirement in the third bullet of Section 4.3 above places ultimate responsibility for remedial action on the sender. If coexistence problems with a Classic ECN AQM are detected (implying they have not been resolved by the network), it says the sender "MUST" revert to a Classic congestion control."

[I-D.ietf-tsvwg-l4sops] also gives example ways in which L4S congestion controls can be rolled out initially in lower risk scenarios.

4.4. Filtering or Smoothing of ECN Feedback

Section 5.2 below specifies that an L4S AQM is expected to signal L4S ECN immediately, to avoid introducing delay due to filtering or smoothing. This contrasts with a Classic AQM, which filters out variations in the queue before signalling ECN marking or drop. In the L4S architecture [I-D.ietf-tsvwg-l4s-arch], responsibility for smoothing out these variations shifts to the sender's congestion control.

This shift of responsibility has the advantage that each sender can smooth variations over a timescale proportionate to its own RTT. Whereas, in the Classic approach, the network doesn't know the RTTs of any of the flows, so it has to smooth out variations for a worst-case RTT to ensure stability. For all the typical flows with shorter RTT than the worst-case, this makes congestion control unnecessarily sluggish.

This also gives an L4S sender the choice not to smooth, depending on its context (start-up, congestion avoidance, etc). Therefore, this document places no requirement on an L4S congestion control to smooth out variations in any particular way. Implementers are encouraged to openly publish the approach they take to smoothing, and the results and experience they gain during the L4S experiment.

5. Network Node Behaviour

5.1. Classification and Re-Marking Behaviour

A network node that implements the L4S service:

- * MUST classify arriving ECT(1) packets for L4S treatment, unless overridden by another classifier (e.g., see Section 5.4.1.2);

- * MUST classify arriving CE packets for L4S treatment as well, unless overridden by a another classifier or unless the exception referred to next applies;

CE packets might have originated as ECT(1) or ECT(0), but the above rule to classify them as if they originated as ECT(1) is the safe choice (see Appendix B for rationale). The exception is where some flow-aware in-network mechanism happens to be available for distinguishing CE packets that originated as ECT(0), as described in Section 5.3, but there is no implication that such a mechanism is necessary.

An L4S AQM treatment follows similar codepoint transition rules to those in RFC 3168. Specifically, the ECT(1) codepoint MUST NOT be changed to any other codepoint than CE, and CE MUST NOT be changed to any other codepoint. An ECT(1) packet is classified as ECN-capable and, if congestion increases, an L4S AQM algorithm will increasingly mark the ECN field as CE, otherwise forwarding packets unchanged as ECT(1). Necessary conditions for an L4S marking treatment are defined in Section 5.2.

Under persistent overload an L4S marking treatment MUST begin applying drop to L4S traffic until the overload episode has subsided, as recommended for all AQM methods in [RFC7567] (Section 4.2.1), which follows the similar advice in RFC 3168 (Section 7). During overload, it MUST apply the same drop probability to L4S traffic as it would to Classic traffic.

Where an L4S AQM is transport-aware, this requirement could be satisfied by using drop in only the most overloaded individual per-flow AQMs. In a DualQ with flow-aware queue protection (e.g. [I-D.briscoe-docsis-q-protection]), this could be achieved by redirecting packets in those flows contributing most to the overload out of the L4S queue so that they are subjected to drop in the Classic queue.

For backward compatibility in uncontrolled environments, a network node that implements the L4S treatment MUST also implement an AQM treatment for the Classic service as defined in Section 1.2. This Classic AQM treatment need not mark ECT(0) packets, but if it does, see Section 5.2 for the strengths of the markings relative to drop. It MUST classify arriving ECT(0) and Not-ECT packets for treatment by this Classic AQM (for the DualQ Coupled AQM, see the extensive discussion on classification in Sections 2.3 and 2.5.1.1 of [I-D.ietf-tsvwg-aqm-dualq-coupled]).

In case unforeseen problems arise with the L4S experiment, it MUST be possible to configure an L4S implementation to disable the L4S treatment. Once disabled, all packets of all ECN codepoints will receive Classic treatment and ECT(1) packets MUST be treated as if they were Not-ECT.

5.2. The Strength of L4S CE Marking Relative to Drop

The relative strengths of L4S CE and drop are irrelevant where AQMs are implemented in separate queues per-application-flow, which are then explicitly scheduled (e.g. with an FQ scheduler as in FQ-CoDel [RFC8290]). Nonetheless, the relationship between them needs to be defined for the coupling between L4S and Classic congestion signals in a DualQ Coupled AQM [I-D.ietf-tsvwg-aqm-dualq-coupled], as below.

Unless an AQM node schedules application flows explicitly, the likelihood that the AQM drops a Not-ECT Classic packet (p_C) MUST be roughly proportional to the square of the likelihood that it would have marked it if it had been an L4S packet (p_L). That is

$$p_C \sim (p_L / k)^2$$

The constant of proportionality (k) does not have to be standardised for interoperability, but a value of 2 is RECOMMENDED. The term 'likelihood' is used above to allow for marking and dropping to be either probabilistic or deterministic.

This formula ensures that Scalable and Classic flows will converge to roughly equal congestion windows, for the worst case of Reno congestion control. This is because the congestion windows of Scalable and Classic congestion controls are inversely proportional to p_L and $\sqrt{p_C}$ respectively. So squaring p_C in the above formula counterbalances the square root that characterizes Reno-friendly flows.

Note that, contrary to RFC 3168, an AQM implementing the L4S and Classic treatments does not mark an ECT(1) packet under the same conditions that it would have dropped a Not-ECT packet, as allowed by [RFC8311], which updates RFC 3168. However, if it marks ECT(0) packets, it does so under the same conditions that it would have dropped a Not-ECT packet [RFC3168].

Also, In the L4S architecture [I-D.ietf-tsvwg-l4s-arch], the sender, not the network, is responsible for smoothing out variations in the queue. So, an L4S AQM MUST signal congestion as soon as possible. Then, an L4S sender generally interprets CE marking as an unsmoothed signal.

This requirement does not prevent an L4S AQM from mixing in additional congestion signals that are smoothed, such as the signals from a Classic smoothed AQM that are coupled with unsmoothed L4S signals in the coupled DualQ [I-D.ietf-tsvwg-aqm-dualq-coupled]. But only as long as the onset of congestion can be signalled immediately, and can be interpreted by the sender as if it has been signalled immediately, which is important for interoperability

5.3. Exception for L4S Packet Identification by Network Nodes with Transport-Layer Awareness

To implement L4S packet classification, a network node does not need to identify transport-layer flows. Nonetheless, if an L4S network node classifies packets by their transport-layer flow ID and their ECN field, and if all the ECT packets in a flow have been ECT(0), the node MAY classify any CE packets in the same flow as if they were Classic ECT(0) packets. In all other cases, a network node MUST classify all CE packets as if they were ECT(1) packets. Examples of such other cases are: i) if no ECT packets have yet been identified in a flow; ii) if it is not desirable for a network node to identify transport-layer flows; or iii) if some ECT packets in a flow have been ECT(1) (this advice will need to be verified as part of L4S experiments).

5.4. Interaction of the L4S Identifier with other Identifiers

The examples in this section concern how additional identifiers might complement the L4S identifier to classify packets between class-based queues. Firstly Section 5.4.1 considers two queues, L4S and Classic, as in the Coupled DualQ AQM [I-D.ietf-tsvwg-aqm-dualq-coupled], either alone (Section 5.4.1.1) or within a larger queuing hierarchy (Section 5.4.1.2). Then Section 5.4.2 considers schemes that might combine per-flow 5-tuples with other identifiers.

5.4.1. DualQ Examples of Other Identifiers Complementing L4S Identifiers

5.4.1.1. Inclusion of Additional Traffic with L4S

In a typical case for the public Internet a network element that implements L4S in a shared queue might want to classify some low-rate but unresponsive traffic (e.g. DNS, LDAP, NTP, voice, game sync packets) into the low latency queue to mix with L4S traffic. In this case it would not be appropriate to call the queue an L4S queue, because it is shared by L4S and non-L4S traffic. Instead it will be called the low latency or L queue. The L queue then offers two different treatments:

- * The L4S treatment, which is a combination of the L4S AQM treatment and a priority scheduling treatment;
- * The low latency treatment, which is solely the priority scheduling treatment, without ECN-marking by the AQM.

To identify packets for just the scheduling treatment, it would be inappropriate to use the L4S ECT(1) identifier, because such traffic is unresponsive to ECN marking. Examples of relevant non-ECN identifiers are:

- * address ranges of specific applications or hosts configured to be, or known to be, safe, e.g. hard-coded IoT devices sending low intensity traffic;
- * certain low data-volume applications or protocols (e.g. ARP, DNS);
- * specific Diffserv codepoints that indicate traffic with limited burstiness such as the EF (Expedited Forwarding [RFC3246]), Voice-Admit [RFC5865] or proposed NQB (Non-Queue-Building [I-D.ietf-tsvwg-nqb]) service classes or equivalent local-use DSCPs (see [I-D.briscoe-tsvwg-l4s-diffserv]).

In summary, a network element that implements L4S in a shared queue MAY classify additional types of packets into the L queue based on identifiers other than the ECN field, but the types SHOULD be 'safe' to mix with L4S traffic, where 'safe' is explained in Section 5.4.1.1.1.

A packet that carries one of these non-ECN identifiers to classify it into the L queue would not be subject to the L4S ECN marking treatment, unless it also carried an ECT(1) or CE codepoint. The specification of an L4S AQM MUST define the behaviour for packets with unexpected combinations of codepoints, e.g. a non-ECN-based classifier for the L queue, but ECT(0) in the ECN field (for examples see section 2.5.1.1 of the DualQ spec [I-D.ietf-tsvwg-aqm-dualq-coupled]).

For clarity, non-ECN identifiers, such as the examples itemized above, might be used by some network operators who believe they identify non-L4S traffic that would be safe to mix with L4S traffic. They are not alternative ways for a host to indicate that it is sending L4S packets. Only the ECT(1) ECN codepoint indicates to a network element that a host is sending L4S packets (and CE indicates that it could have originated as ECT(1)). Specifically ECT(1) indicates that the host claims its behaviour satisfies the prerequisite transport requirements in Section 4.

In order to include non-L4S packets in the L queue, a network node MUST NOT alter Not-ECT or ECT(0) in the IP-ECN field to an L4S identifier. This ensures that these codepoints survive for any potential use later on the network path.

5.4.1.1.1. 'Safe' Unresponsive Traffic

The above section requires unresponsive traffic to be 'safe' to mix with L4S traffic. Ideally this means that the sender never sends any sequence of packets at a rate that exceeds the available capacity of the bottleneck link. However, typically an unresponsive transport does not even know the bottleneck capacity of the path, let alone its available capacity. Nonetheless, an application can be considered safe enough if it paces packets out (not necessarily completely regularly) such that its maximum instantaneous rate from packet to packet stays well below a typical broadband access rate.

This is a vague but useful definition, because many low latency applications of interest, such as DNS, voice, game sync packets, RPC, ACKs, keep-alives, could match this description.

Low rate streams such as voice and game sync packets, might not use continuously adapting ECN-based congestion control, but they ought to at least use a 'circuit-breaker' style of congestion response [RFC8083]. If the volume of traffic from unresponsive applications is high enough to overload the link, this will at least protect the capacity available to responsive applications. However, queuing delay in the L queue will probably rise to that controlled by the Classic (drop-based) AQM. If a network operator considers that such self-restraint is not enough, it might want to police the L queue (see Section 8.2 of the L4S architecture [I-D.ietf-tsvwg-l4s-arch]).

5.4.1.2. Exclusion of Traffic From L4S Treatment

To extend the above example, an operator might want to exclude some traffic from the L4S treatment for a policy reason, e.g. security (traffic from malicious sources) or commercial (e.g. initially the operator may wish to confine the benefits of L4S to business customers).

In this exclusion case, the classifier MUST classify on the relevant locally-used identifiers (e.g. source addresses) before classifying the non-matching traffic on the end-to-end L4S ECN identifier.

A network node MUST NOT alter the end-to-end L4S ECN identifier from L4S to Classic, because an operator decision to exclude certain traffic from L4S treatment is local-only. The end-to-end L4S

identifier then survives for other operators to use, or indeed, they can apply their own policy, independently based on their own choice of locally-used identifiers. This approach also allows any operator to remove its locally-applied exclusions in future, e.g. if it wishes to widen the benefit of the L4S treatment to all its customers.

A network node that supports L4S but excludes certain packets carrying the L4S identifier from L4S treatment **MUST** still apply marking or dropping that is compatible with an L4S congestion response. For instance, it could either drop such packets with the same likelihood as Classic packets or it could ECN-mark them with a likelihood appropriate to L4S traffic (e.g. the coupled probability in a DualQ coupled AQM) but aiming for the Classic delay target. It **MUST NOT** ECN-mark such packets with a Classic marking probability, which could confuse the sender.

5.4.1.3. Generalized Combination of L4S and Other Identifiers

L4S concerns low latency, which it can provide for all traffic without differentiation and without necessarily affecting bandwidth allocation. Diffserv provides for differentiation of both bandwidth and low latency, but its control of latency depends on its control of bandwidth. The two can be combined if a network operator wants to control bandwidth allocation but it also wants to provide low latency - for any amount of traffic within one of these allocations of bandwidth (rather than only providing low latency by limiting bandwidth) [I-D.briscoe-tsvwg-l4s-diffserv].

The DualQ examples so far have been framed in the context of providing the default Best Efforts Per-Hop Behaviour (PHB) using two queues - a Low Latency (L) queue and a Classic (C) Queue. This single DualQ structure is expected to be the most common and useful arrangement. But, more generally, an operator might choose to control bandwidth allocation through a hierarchy of Diffserv PHBs at a node, and to offer one (or more) of these PHBs using a pair of queues for a low latency and a Classic variant of the PHB.

In the first case, if we assume that a network element provides no PHBs except the DualQ, if a packet carries ECT(1) or CE, the network element would classify it for the L4S treatment irrespective of its DSCP. And, if a packet carried (say) the EF DSCP, the network element could classify it into the L queue irrespective of its ECN codepoint. However, where the DualQ is in a hierarchy of other PHBs, the classifier would classify some traffic into other PHBs based on DSCP before classifying between the low latency and Classic queues (based on ECT(1), CE and perhaps also the EF DSCP or other identifiers as in the above example).

[I-D.briscoe-tsvwg-l4s-diffserv] gives a number of examples of such arrangements to address various requirements.

[I-D.briscoe-tsvwg-l4s-diffserv] describes how an operator might use L4S to offer low latency as well as using Diffserv for bandwidth differentiation. It identifies two main types of approach, which can be combined: the operator might split certain Diffserv PHBs between L4S and a corresponding Classic service. Or it might split the L4S and/or the Classic service into multiple Diffserv PHBs. In either of these cases, a packet would have to be classified on its Diffserv and ECN codepoints.

In summary, there are numerous ways in which the L4S ECN identifier (ECT(1) and CE) could be combined with other identifiers to achieve particular objectives. The following categorization articulates those that are valid, but it is not necessarily exhaustive. Those tagged 'Recommended-standard-use' could be set by the sending host or a network. Those tagged 'Local-use' would only be set by a network:

1. Identifiers Complementing the L4S Identifier
 - a. Including More Traffic in the L Queue
(Could use Recommended-standard-use or Local-use identifiers)
 - b. Excluding Certain Traffic from the L Queue
(Local-use only)
2. Identifiers to place L4S classification in a PHB Hierarchy
(Could use Recommended-standard-use or Local-use identifiers)
 - a. PHBs Before L4S ECN Classification
 - b. PHBs After L4S ECN Classification

5.4.2. Per-Flow Queuing Examples of Other Identifiers Complementing L4S Identifiers

At a node with per-flow queueing (e.g. FQ-CoDel [RFC8290]), the L4S identifier could complement the Layer-4 flow ID as a further level of flow granularity (i.e. Not-ECT and ECT(0) queued separately from ECT(1) and CE packets). "Risk of reordering Classic CE packets" in Appendix B discusses the resulting ambiguity if packets originally marked ECT(0) are marked CE by an upstream AQM before they arrive at a node that classifies CE as L4S. It argues that the risk of reordering is vanishingly small and the consequence of such a low level of reordering is minimal.

Alternatively, it could be assumed that it is not in a flow's own interest to mix Classic and L4S identifiers. Then the AQM could use the ECN field to switch itself between a Classic and an L4S AQM behaviour within one per-flow queue. For instance, for ECN-capable packets, the AQM might consist of a simple marking threshold and an L4S ECN identifier might simply select a shallower threshold than a Classic ECN identifier would.

5.5. Limiting Packet Bursts from Links

As well as senders needing to limit packet bursts (Section 4.3), links need to limit the degree of burstiness they introduce. In both cases (senders and links) this is a tradeoff, because batch-handling of packets is done for good reason, e.g. processing efficiency or to make efficient use of medium acquisition delay. Some take the attitude that there is no point reducing burst delay at the sender below that introduced by links (or vice versa). However, delay reduction proceeds by cutting down 'the longest pole in the tent', which turns the spotlight on the next longest, and so on.

This document does not set any quantified requirements for links to limit burst delay, primarily because link technologies are outside the remit of L4S specifications. Nonetheless, the following two subsections outline opportunities for addressing bursty links in the process of L4S implementation and deployment.

5.5.1. Limiting Packet Bursts from Links Fed by an L4S AQM

It would not make sense to implement an L4S AQM that feeds into a particular link technology without also reviewing opportunities to reduce any form of burst delay introduced by that link technology. This would at least limit the bursts that the link would otherwise introduce into the onward traffic, which would cause jumpy feedback to the sender as well as potential extra queuing delay downstream. This document does not presume to even give guidance on an

appropriate target for such burst delay until there is more industry experience of L4S. However, as suggested in Section 4.3 it would not seem necessary to limit bursts lower than roughly 10% of the minimum base RTT expected in the typical deployment scenario (e.g. 250 us burst duration for links within the public Internet).

5.5.2. Limiting Packet Bursts from Links Upstream of an L4S AQM

The initial scope of the L4S experiment is to deploy L4S AQMs at bottlenecks and L4S congestion controls at senders. This is expected to highlight interactions with the most bursty upstream links and lead operators to tune down the burstiness of those links in their network that are configurable, or failing that, to have to compromise on the delay target of some L4S AQMs. It might also require specific redesign work relevant to the most problematic link types. Such knock-on effects of initial L4S deployment would all be part of the learning from the L4S experiment.

The details of such link changes are beyond the scope of the present document. Nonetheless, where L4S technology is being implemented on an outgoing interface of a device, it would make sense to consider opportunities for reducing bursts arriving at other incoming interface(s). For instance, where an L4S AQM is implemented to feed into the upstream WAN interface of a home gateway, there would be opportunities to alter the WiFi profiles sent out of any WiFi interfaces from the same device, in order to mitigate incoming bursts of aggregated WiFi frames from other WiFi stations.

6. Behaviour of Tunnels and Encapsulations

6.1. No Change to ECN Tunnels and Encapsulations in General

The L4S identifier is expected to work through and within any tunnel without modification, as long as the tunnel propagates the ECN field in any of the ways that have been defined since the first variant in the year 2001 [RFC3168]. L4S will also work with (but does not rely on) any of the more recent updates to ECN propagation in [RFC4301], [RFC6040] or [I-D.ietf-tsvwg-rfc6040update-shim]. However, it is likely that some tunnels still do not implement ECN propagation at all. In these cases, L4S will work through such tunnels, but within them the outer header of L4S traffic will appear as Classic.

AQMs are typically implemented where an IP-layer buffer feeds into a lower layer, so they are agnostic to link layer encapsulations. Where a bottleneck link is not IP-aware, the L4S identifier is still expected to work within any lower layer encapsulation without modification, as long as it propagates the ECN field as defined for the link technology, for example for MPLS [RFC5129] or

TRILL [I-D.ietf-trill-ecn-support]. In some of these cases, e.g. layer-3 Ethernet switches, the AQM accesses the IP layer header within the outer encapsulation, so again the L4S identifier is expected to work without modification. Nonetheless, the programme to define ECN for other lower layers is still in progress [I-D.ietf-tsvwg-ecn-encap-guidelines].

6.2. VPN Behaviour to Avoid Limitations of Anti-Replay

If a mix of L4S and Classic packets is sent into the same security association (SA) of a virtual private network (VPN), and if the VPN egress is employing the optional anti-replay feature, it could inappropriately discard Classic packets (or discard the records in Classic packets) by mistaking their greater queuing delay for a replay attack (see "Dropped Packets for Tunnels with Replay Protection Enabled" in [Heist21] for the potential performance impact). This known problem is common to both IPsec [RFC4301] and DTLS [RFC6347] VPNs, given they use similar anti-replay window mechanisms. The mechanism used can only check for replay within its window, so if the window is smaller than the degree of reordering, it can only assume there might be a replay attack and discard all the packets behind the trailing edge of the window. The specifications of IPsec AH [RFC4302] and ESP [RFC4303] suggest that an implementer scales the size of the anti-replay window with interface speed, and DTLS 1.3 [I-D.ietf-tls-dtls13] says "The receiver SHOULD pick a window large enough to handle any plausible reordering, which depends on the data rate." However, in practice, the size of a VPN's anti-replay window is not always scaled appropriately.

If a VPN carrying traffic participating in the L4S experiment experiences inappropriate replay detection, the foremost remedy would be to ensure that the egress is configured to comply with the above window-sizing requirements.

If an implementation of a VPN egress does not support a sufficiently large anti-replay window, e.g. due to hardware limitations, one of the temporary alternatives listed in order of preference below might be feasible instead:

- * If the VPN can be configured to classify packets into different SAs indexed by DSCP, apply the appropriate locally defined DSCPs to Classic and L4S packets. The DSCPs could be applied by the network (based on the least significant bit of the ECN field), or by the sending host. Such DSCPs would only need to survive as far as the VPN ingress.
- * If the above is not possible and it is necessary to use L4S, either of the following might be appropriate as a last resort:

- disable anti-replay protection at the VPN egress, after considering the security implications (optional anti-replay is mandatory in both IPsec and DTLS);
- configure the tunnel ingress not to propagate ECN to the outer, which would lose the benefits of L4S and Classic ECN over the VPN.

Modification to VPN implementations is outside the present scope, which is why this section has so far focused on reconfiguration. Although this document does not define any requirements for VPN implementations, determining whether there is a need for such requirements could be one aspect of L4S experimentation.

7. L4S Experiments

This section describes open questions that L4S Experiments ought to focus on. This section also documents outstanding open issues that will need to be investigated as part of L4S experimentation, given they could not be fully resolved during the WG phase. It also lists metrics that will need to be monitored during experiments (summarizing text elsewhere in L4S documents) and finally lists some potential future directions that researchers might wish to investigate.

In addition to this section, the DualQ spec [I-D.ietf-tsvwg-aqm-dualq-coupled] sets operational and management requirements for experiments with DualQ Coupled AQMs; and General operational and management requirements for experiments with L4S congestion controls are given in Section 4 and Section 5 above, e.g. co-existence and scaling requirements, incremental deployment arrangements.

The specification of each scalable congestion control will need to include protocol-specific requirements for configuration and monitoring performance during experiments. Appendix A of the guidelines in [RFC5706] provides a helpful checklist.

7.1. Open Questions

L4S experiments would be expected to answer the following questions:

- * Have all the parts of L4S been deployed, and if so, what proportion of paths support it?
 - What types of L4S AQMs were deployed, e.g. FQ, coupled DualQ, uncoupled DualQ, other? And how prevalent was each?

- Are the signalling patterns emitted by the deployed AQMs in any way different from those expected when the Prague requirements for endpoints were written?
- * Does use of L4S over the Internet result in significantly improved user experience?
- * Has L4S enabled novel interactive applications?
- * Did use of L4S over the Internet result in improvements to the following metrics:
 - queue delay (mean and 99th percentile) under various loads;
 - utilization;
 - starvation / fairness;
 - scaling range of flow rates and RTTs?
- * How dependent was the performance of L4S service on the bottleneck bandwidth or the path RTT?
- * How much do bursty links in the Internet affect L4S performance (see "Underutilization with Bursty Links" in [Heist21]) and how prevalent are they? How much limitation of burstiness from upstream links was needed and/or was realized - both at senders and at links, especially radio links or how much did L4S target delay have to be increased to accommodate the bursts (see bullet #7 in Section 4.3 and Section 5.5.2)?
- * Is the initial experiment with mis-marked bursty traffic at high RTT (see "Underutilization with Bursty Traffic" in [Heist21]) indicative of similar problems at lower RTTs and, if so, how effective is the suggested remedy in Appendix A.1 of the DualQ spec [I-D.ietf-tsvwg-aqm-dualq-coupled] (or possible other remedies)?
- * Was per-flow queue protection typically (un)necessary?
 - How well did overload protection or queue protection work?
- * How well did L4S flows coexist with Classic flows when sharing a bottleneck?
 - How frequently did problems arise?

- What caused any coexistence problems, and were any problems due to single-queue Classic ECN AQMs (this assumes single-queue Classic ECN AQMs can be distinguished from FQ ones)?
- * How prevalent were problems with the L4S service due to tunnels / encapsulations that do not support ECN decapsulation?
- * How easy was it to implement a fully compliant L4S congestion control, over various different transport protocols (TCP, QUIC, RMCAT, etc)?

Monitoring for harm to other traffic, specifically bandwidth starvation or excess queuing delay, will need to be conducted alongside all early L4S experiments. It is hard, if not impossible, for an individual flow to measure its impact on other traffic. So such monitoring will need to be conducted using bespoke monitoring across flows and/or across classes of traffic.

7.2. Open Issues

- * What is the best way forward to deal with L4S over single-queue Classic ECN AQM bottlenecks, given current problems with misdetecting L4S AQMs as Classic ECN AQMs? See the L4S operational guidance [I-D.ietf-tsvwg-l4sops].
- * Fixing the poor Interaction between current L4S congestion controls and CoDel with only Classic ECN support during flow startup. Originally, this was due to a bug in the initialization of the congestion EWMA in the Linux implementation of TCP Prague. That was quickly fixed, which removed the main performance impact, but further improvement would be useful (either by modifying CoDel, Scalable congestion controls, or both).

7.3. Future Potential

Researchers might find that L4S opens up the following interesting areas for investigation:

- * Potential for faster convergence time and tracking of available capacity;
- * Potential for improvements to particular link technologies, and cross-layer interactions with them;
- * Potential for using virtual queues, e.g. to further reduce latency jitter, or to leave headroom for capacity variation in radio networks;

- * Development and specification of reverse path congestion control using L4S building blocks (e.g. AccECN, QUIC);
- * Once queuing delay is cut down, what becomes the 'second longest pole in the tent' (other than the speed of light)?
- * Novel alternatives to the existing set of L4S AQMs;
- * Novel applications enabled by L4S.

8. IANA Considerations

The 01 codepoint of the ECN Field of the IP header is specified by the present Experimental RFC. The process for an experimental RFC to assign this codepoint in the IP header (v4 and v6) is documented in Proposed Standard [RFC8311], which updates the Proposed Standard [RFC3168].

When the present document is published as an RFC, IANA is asked to update the 01 entry in the registry, "ECN Field (Bits 6-7)" to the following (see <https://www.iana.org/assignments/dscp-registry/dscp-registry.xhtml#ecn-field>):

Binary	Keyword	References
01	ECT(1) (ECN-Capable Transport (1)) [1]	[RFC8311] [RFC Errata 5399] [RFCXXXX]

Table 1

[XXXX is the number that the RFC Editor assigns to the present document (this sentence to be removed by the RFC Editor)].

9. Security Considerations

Approaches to assure the integrity of signals using the new identifier are introduced in Appendix C.1. See the security considerations in the L4S architecture [I-D.ietf-tsvwg-l4s-arch] for further discussion of mis-use of the identifier, as well as extensive discussion of policing rate and latency in regard to L4S.

If the anti-replay window of a VPN egress is too small, it will mistake deliberate delay differences as a replay attack, and discard higher delay packets (e.g. Classic) carried within the same security association (SA) as low delay packets (e.g. L4S). Section 6.2 recommends that VPNs used in L4S experiments are configured with a sufficiently large anti-replay window, as required by the relevant specifications. It also discusses other alternatives.

If a user taking part in the L4S experiment sets up a VPN without being aware of the above advice, and if the user allows anyone to send traffic into their VPN, they would open up a DoS vulnerability in which an attacker could induce the VPN's anti-replay mechanism to discard enough of the user's Classic (C) traffic (if they are receiving any) to cause a significant rate reduction. While the user is actively downloading C traffic, the attacker sends C traffic into the VPN to fill the remainder of the bottleneck link, then sends intermittent L4S packets to maximize the chance of exceeding the VPN's replay window. The user can prevent this attack by following the recommendations in Section 6.2.

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

10. Acknowledgements

Thanks to Richard Scheffenegger, John Leslie, David Taeht, Jonathan Morton, Gorrry Fairhurst, Michael Welzl, Mikael Abrahamsson and Andrew McGregor for the discussions that led to this specification. Ing-jyh (Inton) Tsang was a contributor to the early drafts of this document. And thanks to Mikael Abrahamsson, Lloyd Wood, Nicolas Kuhn, Greg White, Tom Henderson, David Black, Gorrry Fairhurst, Brian Carpenter, Jake Holland, Rod Grimes, Richard Scheffenegger, Sebastian Moeller, Neal Cardwell, Praveen Balasubramanian, Reza Marandian Hagh, Pete Heist, Stuart Cheshire, Vidhi Goel, Mirja Kuehlewind and Ermin Sakic for providing help and reviewing this draft and thanks to Ingemar Johansson for reviewing and providing substantial text. Thanks to Sebastian Moeller for identifying the interaction with VPN anti-replay and to Jonathan Morton for identifying the attack based on this. Particular thanks to tsvwg chairs Gorrry Fairhurst, David Black and Wes Eddy for patiently helping this and the other L4S drafts through the IETF process. Appendix A listing the Prague L4S Requirements is based on text authored by Marcelo Bagnulo Braun that was originally an appendix to [I-D.ietf-tsvwg-l4s-arch]. That text was in turn based on the collective output of the attendees listed in the minutes of a 'bar BoF' on DCTCP Evolution during IETF-94 [TCPPrague].

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

11. References

11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC4774] Floyd, S., "Specifying Alternate Semantics for the Explicit Congestion Notification (ECN) Field", BCP 124, RFC 4774, DOI 10.17487/RFC4774, November 2006, <<https://www.rfc-editor.org/info/rfc4774>>.
- [RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", RFC 6679, DOI 10.17487/RFC6679, August 2012, <<https://www.rfc-editor.org/info/rfc6679>>.

11.2. Informative References

- [A2DTCP] Zhang, T., Wang, J., Huang, J., Huang, Y., Chen, J., and Y. Pan, "Adaptive-Acceleration Data Center TCP", IEEE Transactions on Computers 64(6):1522-1533, June 2015, <<http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6871352>>.
- [Ahmed19] Ahmed, A.S., "Extending TCP for Low Round Trip Delay", Masters Thesis, Uni Oslo, August 2019, <<https://www.duo.uio.no/handle/10852/70966>>.
- [Alizadeh-stability] Alizadeh, M., Javanmard, A., and B. Prabhakar, "Analysis of DCTCP: Stability, Convergence, and Fairness", ACM

SIGMETRICS 2011 , June 2011,
<[https://people.csail.mit.edu/alizadeh/papers/
dctcp_analysis-sigmetrics11.pdf](https://people.csail.mit.edu/alizadeh/papers/dctcp_analysis-sigmetrics11.pdf)>.

- [ARED01] Floyd, S., Gummadi, R., and S. Shenker, "Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management", ACIRI Technical Report , August 2001, <<http://www.icir.org/floyd/red.html>>.
- [BBRv2] Cardwell, N., "BRTCP BBR v2 Alpha/Preview Release", github repository; Linux congestion control module, <<https://github.com/google/bbr/blob/v2alpha/README.md>>.
- [COBALT] Palmei, J., Gupta, S., Imputato, P., Morton, J., Tahiliani, M., Avallone, S., and D. Taht, "Design and Evaluation of COBALT Queue Discipline", In Proc. IEEE Int'l Symp. on Local and Metropolitan Area Networks 2019, ppl--6, 2019, <<https://doi.org/10.1109/LANMAN.2019.8847054>>.
- [DCttH19] De Schepper, K., Bondarenko, O., Tilmans, O., and B. Briscoe, "'Data Centre to the Home': Ultra-Low Latency for All", Updated RITE project Technical Report , July 2019, <https://bobbriscoe.net/pubs.html#DCttH_TR>.
- [DualPI2Linux] Albisser, O., De Schepper, K., Briscoe, B., Tilmans, O., and H. Steen, "DUALPI2 - Low Latency, Low Loss and Scalable (L4S) AQM", Proc. Linux Netdev 0x13 , March 2019, <<https://www.netdevconf.org/0x13/session.html?talk-DUALPI2-AQM>>.
- [ecn-fallback] Briscoe, B. and A.S. Ahmed, "TCP Prague Fall-back on Detection of a Classic ECN AQM", bobbriscoe.net Technical Report TR-BB-2019-002, April 2020, <<https://arxiv.org/abs/1911.00710>>.
- [Heist21] Heist, P. and J. Morton, "L4S Tests", github README, May 2021, <<https://github.com/heistp/l4s-tests/>>.
- [I-D.briscoe-docsis-q-protection] Briscoe, B. and G. White, "The DOCSIS(r) Queue Protection Algorithm to Preserve Low Latency", Work in Progress, Internet-Draft, draft-briscoe-docsis-q-protection-02, 31 January 2022, <<https://datatracker.ietf.org/doc/html/draft-briscoe-docsis-q-protection-02>>.

- [I-D.briscoe-iccrp-prague-congestion-control]
Schepper, K. D., Tilmans, O., and B. Briscoe, "Prague Congestion Control", Work in Progress, Internet-Draft, draft-briscoe-iccrp-prague-congestion-control-00, 9 March 2021, <<https://datatracker.ietf.org/doc/html/draft-briscoe-iccrp-prague-congestion-control-00>>.
- [I-D.briscoe-tsvwg-l4s-diffserv]
Briscoe, B., "Interactions between Low Latency, Low Loss, Scalable Throughput (L4S) and Differentiated Services", Work in Progress, Internet-Draft, draft-briscoe-tsvwg-l4s-diffserv-02, 4 November 2018, <<https://datatracker.ietf.org/doc/html/draft-briscoe-tsvwg-l4s-diffserv-02>>.
- [I-D.cardwell-iccrp-bbr-congestion-control]
Cardwell, N., Cheng, Y., Yeganeh, S. H., Swett, I., and V. Jacobson, "BBR Congestion Control", Work in Progress, Internet-Draft, draft-cardwell-iccrp-bbr-congestion-control-01, 7 November 2021, <<https://datatracker.ietf.org/doc/html/draft-cardwell-iccrp-bbr-congestion-control-01>>.
- [I-D.ietf-tcpm-accurate-ecn]
Briscoe, B., Kühlewind, M., and R. Scheffenegger, "More Accurate ECN Feedback in TCP", Work in Progress, Internet-Draft, draft-ietf-tcpm-accurate-ecn-16, 3 February 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-accurate-ecn-16>>.
- [I-D.ietf-tcpm-generalized-ecn]
Bagnulo, M. and B. Briscoe, "ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control Packets", Work in Progress, Internet-Draft, draft-ietf-tcpm-generalized-ecn-09, 31 January 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-generalized-ecn-09>>.
- [I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-43, 30 April 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-tls-dtls13-43>>.

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

Eastlake, D. E. and B. Briscoe, "TRILL (TRansparent Interconnection of Lots of Links): ECN (Explicit Congestion Notification) Support", Work in Progress, Internet-Draft, draft-ietf-trill-ecn-support-07, 25 February 2018, <<https://datatracker.ietf.org/doc/html/draft-ietf-trill-ecn-support-07>>.

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

Schepper, K. D., Briscoe, B., and G. White, "DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)", Work in Progress, Internet-Draft, draft-ietf-tsvwg-aqm-dualq-coupled-22, 4 March 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-aqm-dualq-coupled-22>>.

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

Briscoe, B. and J. Kaippallimalil, "Guidelines for Adding Congestion Notification to Protocols that Encapsulate IP", Work in Progress, Internet-Draft, draft-ietf-tsvwg-ecn-encap-guidelines-16, 25 May 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-ecn-encap-guidelines-16>>.

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

Briscoe, B., Schepper, K. D., Bagnulo, M., and G. White, "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture", Work in Progress, Internet-Draft, draft-ietf-tsvwg-l4s-arch-16, 1 February 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-l4s-arch-16>>.

[I-D.ietf-tsvwg-l4sops]

White, G., "Operational Guidance for Deployment of L4S in the Internet", Work in Progress, Internet-Draft, draft-ietf-tsvwg-l4sops-02, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-l4sops-02>>.

[I-D.ietf-tsvwg-nqb]

White, G. and T. Fossati, "A Non-Queue-Building Per-Hop Behavior (NQB PHB) for Differentiated Services", Work in Progress, Internet-Draft, draft-ietf-tsvwg-nqb-10, 4 March 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-nqb-10>>.

- [I-D.ietf-tsvwg-rfc6040update-shim]
Briscoe, B., "Propagating Explicit Congestion Notification Across IP Tunnel Headers Separated by a Shim", Work in Progress, Internet-Draft, draft-ietf-tsvwg-rfc6040update-shim-14, 25 May 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-rfc6040update-shim-14>>.
- [I-D.sridharan-tcpm-ctcp]
Sridharan, M., Tan, K., Bansal, D., and D. Thaler, "Compound TCP: A New TCP Congestion Control for High-Speed and Long Distance Networks", Work in Progress, Internet-Draft, draft-sridharan-tcpm-ctcp-02, 11 November 2008, <<https://datatracker.ietf.org/doc/html/draft-sridharan-tcpm-ctcp-02>>.
- [I-D.stewart-tsvwg-sctp-ecn]
Stewart, R. R., Tuexen, M., and X. Dong, "ECN for Stream Control Transmission Protocol (SCTP)", Work in Progress, Internet-Draft, draft-stewart-tsvwg-sctp-ecn-05, 15 January 2014, <<https://datatracker.ietf.org/doc/html/draft-stewart-tsvwg-sctp-ecn-05>>.
- [LinuxPacedChirping]
Misund, J. and B. Briscoe, "Paced Chirping - Rethinking TCP start-up", Proc. Linux Netdev 0x13 , March 2019, <<https://www.netdevconf.org/0x13/session.html?talk-chirp>>.
- [Mathis09] Mathis, M., "Relentless Congestion Control", PFLDNet'09 , May 2009, <http://www.hpcc.jp/pfldnet2009/Program_files/1569198525.pdf>.
- [Paced-Chirping]
Misund, J., "Rapid Acceleration in TCP Prague", Masters Thesis , May 2018, <<https://riteproject.files.wordpress.com/2018/07/misundjoakimmastersthesissubmitted180515.pdf>>.
- [PI2] De Schepper, K., Bondarenko, O., Tsang, I., and B. Briscoe, "PI² : A Linearized AQM for both Classic and Scalable TCP", Proc. ACM CoNEXT 2016 pp.105-119, December 2016, <<http://dl.acm.org/citation.cfm?doid=2999572.2999578>>.
- [PragueLinux]
Briscoe, B., De Schepper, K., Albisser, O., Misund, J., Tilmans, O., Kühlewind, M., and A.S. Ahmed, "Implementing the 'TCP Prague' Requirements for Low Latency Low Loss

Scalable Throughput (L4S)", Proc. Linux Netdev 0x13 ,
March 2019, <[https://www.netdevconf.org/0x13/
session.html?talk-tcp-prague-l4s](https://www.netdevconf.org/0x13/session.html?talk-tcp-prague-l4s)>.

- [QV] Briscoe, B. and P. Hurtig, "Up to Speed with Queue View",
RITE Technical Report D2.3; Appendix C.2, August 2015,
<[https://riteproject.files.wordpress.com/2015/12/rite-
deliverable-2-3.pdf](https://riteproject.files.wordpress.com/2015/12/rite-deliverable-2-3.pdf)>.
- [RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering,
S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G.,
Partridge, C., Peterson, L., Ramakrishnan, K., Shenker,
S., Wroclawski, J., and L. Zhang, "Recommendations on
Queue Management and Congestion Avoidance in the
Internet", RFC 2309, DOI 10.17487/RFC2309, April 1998,
<<https://www.rfc-editor.org/info/rfc2309>>.
- [RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black,
"Definition of the Differentiated Services Field (DS
Field) in the IPv4 and IPv6 Headers", RFC 2474,
DOI 10.17487/RFC2474, December 1998,
<<https://www.rfc-editor.org/info/rfc2474>>.
- [RFC3246] Davie, B., Charny, A., Bennet, J.C.R., Benson, K., Le
Boudec, J.Y., Courtney, W., Davari, S., Firoiu, V., and D.
Stiliadis, "An Expedited Forwarding PHB (Per-Hop
Behavior)", RFC 3246, DOI 10.17487/RFC3246, March 2002,
<<https://www.rfc-editor.org/info/rfc3246>>.
- [RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit
Congestion Notification (ECN) Signaling with Nonces",
RFC 3540, DOI 10.17487/RFC3540, June 2003,
<<https://www.rfc-editor.org/info/rfc3540>>.
- [RFC3649] Floyd, S., "HighSpeed TCP for Large Congestion Windows",
RFC 3649, DOI 10.17487/RFC3649, December 2003,
<<https://www.rfc-editor.org/info/rfc3649>>.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the
Internet Protocol", RFC 4301, DOI 10.17487/RFC4301,
December 2005, <<https://www.rfc-editor.org/info/rfc4301>>.
- [RFC4302] Kent, S., "IP Authentication Header", RFC 4302,
DOI 10.17487/RFC4302, December 2005,
<<https://www.rfc-editor.org/info/rfc4302>>.

- [RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <<https://www.rfc-editor.org/info/rfc4303>>.
- [RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI 10.17487/RFC4340, March 2006, <<https://www.rfc-editor.org/info/rfc4340>>.
- [RFC4341] Floyd, S. and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control", RFC 4341, DOI 10.17487/RFC4341, March 2006, <<https://www.rfc-editor.org/info/rfc4341>>.
- [RFC4342] Floyd, S., Kohler, E., and J. Padhye, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC)", RFC 4342, DOI 10.17487/RFC4342, March 2006, <<https://www.rfc-editor.org/info/rfc4342>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.
- [RFC5033] Floyd, S. and M. Allman, "Specifying New Congestion Control Algorithms", BCP 133, RFC 5033, DOI 10.17487/RFC5033, August 2007, <<https://www.rfc-editor.org/info/rfc5033>>.
- [RFC5129] Davie, B., Briscoe, B., and J. Tay, "Explicit Congestion Marking in MPLS", RFC 5129, DOI 10.17487/RFC5129, January 2008, <<https://www.rfc-editor.org/info/rfc5129>>.
- [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <<https://www.rfc-editor.org/info/rfc5348>>.
- [RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", RFC 5562, DOI 10.17487/RFC5562, June 2009, <<https://www.rfc-editor.org/info/rfc5562>>.

- [RFC5622] Floyd, S. and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion ID 4: TCP-Friendly Rate Control for Small Packets (TFRC-SP)", RFC 5622, DOI 10.17487/RFC5622, August 2009, <<https://www.rfc-editor.org/info/rfc5622>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/info/rfc5681>>.
- [RFC5706] Harrington, D., "Guidelines for Considering Operations and Management of New Protocols and Protocol Extensions", RFC 5706, DOI 10.17487/RFC5706, November 2009, <<https://www.rfc-editor.org/info/rfc5706>>.
- [RFC5865] Baker, F., Polk, J., and M. Dolly, "A Differentiated Services Code Point (DSCP) for Capacity-Admitted Traffic", RFC 5865, DOI 10.17487/RFC5865, May 2010, <<https://www.rfc-editor.org/info/rfc5865>>.
- [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <<https://www.rfc-editor.org/info/rfc5925>>.
- [RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <<https://www.rfc-editor.org/info/rfc6040>>.
- [RFC6077] Papadimitriou, D., Ed., Welzl, M., Scharf, M., and B. Briscoe, "Open Research Issues in Internet Congestion Control", RFC 6077, DOI 10.17487/RFC6077, February 2011, <<https://www.rfc-editor.org/info/rfc6077>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6660] Briscoe, B., Moncaster, T., and M. Menth, "Encoding Three Pre-Congestion Notification (PCN) States in the IP Header Using a Single Diffserv Codepoint (DSCP)", RFC 6660, DOI 10.17487/RFC6660, July 2012, <<https://www.rfc-editor.org/info/rfc6660>>.
- [RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, DOI 10.17487/RFC6675, August 2012, <<https://www.rfc-editor.org/info/rfc6675>>.

- [RFC7560] Kuehlewind, M., Ed., Scheffenegger, R., and B. Briscoe, "Problem Statement and Requirements for Increased Accuracy in Explicit Congestion Notification (ECN) Feedback", RFC 7560, DOI 10.17487/RFC7560, August 2015, <<https://www.rfc-editor.org/info/rfc7560>>.
- [RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <<https://www.rfc-editor.org/info/rfc7567>>.
- [RFC7713] Mathis, M. and B. Briscoe, "Congestion Exposure (ConEx) Concepts, Abstract Mechanism, and Requirements", RFC 7713, DOI 10.17487/RFC7713, December 2015, <<https://www.rfc-editor.org/info/rfc7713>>.
- [RFC8033] Pan, R., Natarajan, P., Baker, F., and G. White, "Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem", RFC 8033, DOI 10.17487/RFC8033, February 2017, <<https://www.rfc-editor.org/info/rfc8033>>.
- [RFC8083] Perkins, C. and V. Singh, "Multimedia Congestion Control: Circuit Breakers for Unicast RTP Sessions", RFC 8083, DOI 10.17487/RFC8083, March 2017, <<https://www.rfc-editor.org/info/rfc8083>>.
- [RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <<https://www.rfc-editor.org/info/rfc8085>>.
- [RFC8257] Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <<https://www.rfc-editor.org/info/rfc8257>>.
- [RFC8290] Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm", RFC 8290, DOI 10.17487/RFC8290, January 2018, <<https://www.rfc-editor.org/info/rfc8290>>.
- [RFC8298] Johansson, I. and Z. Sarker, "Self-Clocked Rate Adaptation for Multimedia", RFC 8298, DOI 10.17487/RFC8298, December 2017, <<https://www.rfc-editor.org/info/rfc8298>>.

- [RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <<https://www.rfc-editor.org/info/rfc8311>>.
- [RFC8312] Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", RFC 8312, DOI 10.17487/RFC8312, February 2018, <<https://www.rfc-editor.org/info/rfc8312>>.
- [RFC8511] Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst, "TCP Alternative Backoff with ECN (ABE)", RFC 8511, DOI 10.17487/RFC8511, December 2018, <<https://www.rfc-editor.org/info/rfc8511>>.
- [RFC8888] Sarker, Z., Perkins, C., Singh, V., and M. Ramalho, "RTP Control Protocol (RTCP) Feedback for Congestion Control", RFC 8888, DOI 10.17487/RFC8888, January 2021, <<https://www.rfc-editor.org/info/rfc8888>>.
- [RFC8985] Cheng, Y., Cardwell, N., Dukkkipati, N., and P. Jha, "The RACK-TLP Loss Detection Algorithm for TCP", RFC 8985, DOI 10.17487/RFC8985, February 2021, <<https://www.rfc-editor.org/info/rfc8985>>.
- [RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <<https://www.rfc-editor.org/info/rfc9000>>.
- [Savage-TCP] Savage, S., Cardwell, N., Wetherall, D., and T. Anderson, "TCP Congestion Control with a Misbehaving Receiver", ACM SIGCOMM Computer Communication Review 29(5):71--78, October 1999.
- [SCReAM] Johansson, I., "SCReAM", github repository; , <<https://github.com/EricssonResearch/scream/blob/master/README.md>>.
- [sub-mss-prob] Briscoe, B. and K. De Schepper, "Scaling TCP's Congestion Window for Small Round Trip Times", BT Technical Report TR-TUB8-2015-002, May 2015, <<https://arxiv.org/abs/1904.07598>>.

- [TCP-CA] Jacobson, V. and M.J. Karels, "Congestion Avoidance and Control", Laurence Berkeley Labs Technical Report , November 1988, <<http://ee.lbl.gov/papers/congavoid.pdf>>.
- [TCPPrague] Briscoe, B., "Notes: DCTCP evolution 'bar BoF': Tue 21 Jul 2015, 17:40, Prague", tcpprague mailing list archive , July 2015, <<https://www.ietf.org/mail-archive/web/tcpprague/current/msg00001.html>>.
- [VCP] Xia, Y., Subramanian, L., Stoica, I., and S. Kalyanaraman, "One more bit is enough", Proc. SIGCOMM'05, ACM CCR 35(4)37--48, 2005, <<http://doi.acm.org/10.1145/1080091.1080098>>.

Appendix A. Rationale for the 'Prague L4S Requirements'

This appendix is informative, not normative. It gives a list of modifications to current scalable congestion controls so that they can be deployed over the public Internet and coexist safely with existing traffic. The list complements the normative requirements in Section 4 that a sender has to comply with before it can set the L4S identifier in packets it sends into the Internet. As well as rationale for safety improvements (the requirements in Section 4) this appendix also includes preferable performance improvements (optimizations).

The requirements and recommendations in Section 4) have become known as the Prague L4S Requirements, because they were originally identified at an ad hoc meeting during IETF-94 in Prague [TCPPrague]. They were originally called the 'TCP Prague Requirements', but they are not solely applicable to TCP, so the name and wording has been generalized for all transport protocols, and the name 'TCP Prague' is now used for a specific implementation of the requirements.

At the time of writing, DCTCP [RFC8257] is the most widely used scalable transport protocol. In its current form, DCTCP is specified to be deployable only in controlled environments. Deploying it in the public Internet would lead to a number of issues, both from the safety and the performance perspective. The modifications and additional mechanisms listed in this section will be necessary for its deployment over the global Internet. Where an example is needed, DCTCP is used as a base, but the requirements in Section 4 apply equally to other scalable congestion controls, covering adaptive real-time media, etc., not just capacity-seeking behaviours.

A.1. Rationale for the Requirements for Scalable Transport Protocols

A.1.1. Use of L4S Packet Identifier

Description: A scalable congestion control needs to distinguish the packets it sends from those sent by Classic congestion controls (see the precise normative requirement wording in Section 4.1).

Motivation: It needs to be possible for a network node to classify L4S packets without flow state into a queue that applies an L4S ECN marking behaviour and isolates L4S packets from the queuing delay of Classic packets.

A.1.2. Accurate ECN Feedback

Description: The transport protocol for a scalable congestion control needs to provide timely, accurate feedback about the extent of ECN marking experienced by all packets (see the precise normative requirement wording in Section 4.2).

Motivation: Classic congestion controls only need feedback about the existence of a congestion episode within a round trip, not precisely how many packets were marked with ECN or dropped. Therefore, in 2001, when ECN feedback was added to TCP [RFC3168], it could not inform the sender of more than one ECN mark per RTT. Since then, requirements for more accurate ECN feedback in TCP have been defined in [RFC7560] and [I-D.ietf-tcpm-accurate-ecn] specifies a change to the TCP protocol to satisfy these requirements. Most other transport protocols already satisfy this requirement (see Section 4.2).

A.1.3. Capable of Replacement by Classic Congestion Control

Description: It needs to be possible to replace the implementation of a scalable congestion control with a Classic control (see the precise normative requirement wording in Section 4.3).

Motivation: L4S is an experimental protocol, therefore it seems prudent to be able to disable it at source in case of insurmountable problems, perhaps due to some unexpected interaction on a particular sender; over a particular path or network; with a particular receiver or even ultimately an insurmountable problem with the experiment as a whole.

A.1.4. Fall back to Classic Congestion Control on Packet Loss

Description: As well as responding to ECN markings in a scalable way, a scalable congestion control needs to react to packet loss in a way that will coexist safely with a Reno congestion control [RFC5681] (see the precise normative requirement wording in Section 4.3).

Motivation: Part of the safety conditions for deploying a scalable congestion control on the public Internet is to make sure that it behaves properly when it builds a queue at a network bottleneck that has not been upgraded to support L4S. Packet loss can have many causes, but it usually has to be conservatively assumed that it is a sign of congestion. Therefore, on detecting packet loss, a scalable congestion control will need to fall back to Classic congestion control behaviour. If it does not comply, it could starve Classic traffic.

A scalable congestion control can be used for different types of transport, e.g. for real-time media or for reliable transport like TCP. Therefore, the particular Classic congestion control behaviour to fall back on will need to be dependent on the specific congestion control implementation. In the particular case of DCTCP, the DCTCP specification [RFC8257] states that "It is RECOMMENDED that an implementation deal with loss episodes in the same way as conventional TCP." For safe deployment, Section 4.3 requires any specification of a scalable congestion control for the public Internet to define the above requirement as a "MUST".

Even though a bottleneck is L4S capable, it might still become overloaded and have to drop packets. In this case, the sender may receive a high proportion of packets marked with the CE bit set and also experience loss. Current DCTCP implementations each react differently to this situation. At least one implementation reacts only to the drop signal (e.g. by halving the CWND) and at least another DCTCP implementation reacts to both signals (e.g. by halving the CWND due to the drop and also further reducing the CWND based on the proportion of marked packet). A third approach for the public Internet has been proposed that adjusts the loss response to result in a halving when combined with the ECN response. We believe that further experimentation is needed to understand what is the best behaviour for the public Internet, which may or not be one of these existing approaches.

A.1.1.5. Coexistence with Classic Congestion Control at Classic ECN bottlenecks

Description: Monitoring has to be in place so that a non-L4S but ECN-capable AQM can be detected at path bottlenecks. This is in case such an AQM has been implemented in a shared queue, in which case any long-running scalable flow would predominate over any simultaneous long-running Classic flow sharing the queue. The precise requirement wording in Section 4.3 is written so that such a problem could either be resolved in real-time, or via administrative intervention.

Motivation: Similarly to the discussion in Appendix A.1.4, this requirement in Section 4.3 is a safety condition to ensure an L4S congestion control coexists well with Classic flows when it builds a queue at a shared network bottleneck that has not been upgraded to support L4S. Nonetheless, if necessary, it is considered reasonable to resolve such problems over management timescales (possibly involving human intervention) because:

- * although a Classic flow can considerably reduce its throughput in the face of a competing scalable flow, it still makes progress and does not starve;
- * implementations of a Classic ECN AQM in a queue that is intended to be shared are believed to be rare;
- * detection of such AQMs is not always clear-cut; so focused out-of-band testing (or even contacting the relevant network operator) would improve certainty.

Therefore, the relevant normative requirement (Section 4.3) is divided into three stages: monitoring, detection and action:

Monitoring: Monitoring involves collection of the measurement data to be analysed. Monitoring is expressed as a 'MUST' for uncontrolled environments, although the placement of the monitoring function is left open. Whether monitoring has to be applied in real-time is expressed as a 'SHOULD'. This allows for the possibility that the operator of an L4S sender (e.g. a CDN) might prefer to test out-of-band for signs of Classic ECN AQMs, perhaps to avoid continually consuming resources to monitor live traffic.

Detection: Detection involves analysis of the monitored data to detect the likelihood of a Classic ECN AQM. Detection can either directly detect actual coexistence problems between flows, or it can aim to identify AQM technologies that are likely to present coexistence problems, based on knowledge of AQMs deployed at the

time. The requirements recommend that detection occurs live in real-time. However, detection is allowed to be deferred (e.g. it might involve further testing targeted at candidate AQMs);

Action: This involves the act of switching the sender to a Classic congestion control. This might occur in real-time within the congestion control for the subsequent duration of a flow, or it might involve administrative action to switch to Classic congestion control for a specific interface or for a certain set of destination addresses.

Instead of the sender taking action itself, the operator of the sender (e.g. a CDN) might prefer to ask the network operator to modify the Classic AQM's treatment of L4S packets; or to ensure L4S packets bypass the AQM; or to upgrade the AQM to support L4S (see the L4S operational guidance [I-D.ietf-tsvwg-l4sops]). Once L4S flows no longer shared the Classic ECN AQM they would obviously no longer detect it, and the requirement to act on it would no longer apply.

The whole set of normative requirements concerning Classic ECN AQMs in Section 4.3 is worded so that it does not apply in controlled environments, such as private networks or data centre networks. CDN servers placed within an access ISP's network can be considered as a single controlled environment, but any onward networks served by the access network, including all the attached customer networks, would be unlikely to fall under the same degree of coordinated control. Monitoring is expressed as a 'MUST' for these uncontrolled segments of paths (e.g. beyond the access ISP in a home network), because there is a possibility that there might be a shared queue Classic ECN AQM in that segment. Nonetheless, the intent of the wording is to only require occasional monitoring of these uncontrolled regions, and not to burden CDN operators if monitoring never uncovers any potential problems.

More detailed discussion of all the above options and alternatives can be found in the L4S operational guidance [I-D.ietf-tsvwg-l4sops].

Having said all the above, the approach recommended in Section 4.3 is to monitor, detect and act in real-time on live traffic. A passive monitoring algorithm to detect a Classic ECN AQM at the bottleneck and fall back to Classic congestion control is described in an extensive technical report [ecn-fallback], which also provides a link to Linux source code, and a large online visualization of its evaluation results. Very briefly, the algorithm primarily monitors RTT variation using the same algorithm that maintains the mean deviation of TCP's smoothed RTT, but it smooths over a duration of the order of a Classic sawtooth. The outcome is also conditioned on

other metrics such as the presence of CE marking and congestion avoidance phase having stabilized. The report also identifies further work to improve the approach, for instance improvements with low capacity links and combining the measurements with a cache of what had been learned about a path in previous connections. The report also suggests alternative approaches.

Although using passive measurements within live traffic (as above) can detect a Classic ECN AQM, it is much harder (perhaps impossible) to determine whether or not the AQM is in a shared queue. Nonetheless, this is much easier using active test traffic out-of-band, because two flows can be used. Section 4 of the same report [ecn-fallback] describes a simple technique to detect a Classic ECN AQM and determine whether it is in a shared queue, summarized here.

An L4S-enabled test server could be set up so that, when a test client accesses it, it serves a script that gets the client to open two parallel long-running flows. It could serve one with a Classic congestion control (C, that sets ECT(0)) and one with a scalable CC (L, that sets ECT(1)). If neither flow induces any ECN marks, it can be presumed the path does not contain a Classic ECN AQM. If either flow induces some ECN marks, the server could measure the relative flow rates and round trip times of the two flows. Table 2 shows the AQM that can be inferred for various cases (presuming the AQM behaviours known at the time of writing).

Rate	RTT	Inferred AQM
L > C	L = C	Classic ECN AQM (FIFO)
L = C	L = C	Classic ECN AQM (FQ)
L = C	L < C	FQ-L4S AQM
L ~ C	L < C	Coupled DualQ AQM

Table 2: Out-of-band testing with two parallel flows. L:=L4S, C:=Classic.

Finally, we motivate the recommendation in Section 4.3 that a scalable congestion control is not expected to change to setting ECT(0) while it adapts its behaviour to coexist with Classic flows. This is because the sender needs to continue to check whether it made the right decision – and switch back if it was wrong, or if a different link becomes the bottleneck:

- * If, as recommended, the sender changes only its behaviour but not its codepoint to Classic, its codepoint will still be compatible with either an L4S or a Classic AQM. If the bottleneck does actually support both, it will still classify ECT(1) into the same L4S queue, where the sender can measure that switching to Classic behaviour was wrong, so that it can switch back.
- * In contrast, if the sender changes both its behaviour and its codepoint to Classic, even if the bottleneck supports both, it will classify ECT(0) into the Classic queue, reinforcing the sender's incorrect decision so that it never switches back.
- * Also, not changing codepoint avoids the risk of being flipped to a different path by a load balancer or multipath routing that hashes on the whole of the ex-ToS byte (unfortunately still a common pathology).

Note that if a flow is configured to only use a Classic congestion control, it is then entirely appropriate not to use ECT(1).

A.1.6. Reduce RTT dependence

Description: A scalable congestion control needs to reduce RTT bias as much as possible at least over the low to typical range of RTTs that will interact in the intended deployment scenario (see the precise normative requirement wording in Section 4.3).

Motivation: The throughput of Classic congestion controls is known to be inversely proportional to RTT, so one would expect flows over very low RTT paths to nearly starve flows over larger RTTs. However, Classic congestion controls have never allowed a very low RTT path to exist because they induce a large queue. For instance, consider two paths with base RTT 1 ms and 100 ms. If a Classic congestion control induces a 100 ms queue, it turns these RTTs into 101 ms and 200 ms leading to a throughput ratio of about 2:1. Whereas if a scalable congestion control induces only a 1 ms queue, the ratio is 2:101, leading to a throughput ratio of about 50:1.

Therefore, with very small queues, long RTT flows will essentially starve, unless scalable congestion controls comply with this requirement in Section 4.3.

The RTT bias in current Classic congestion controls works satisfactorily when the RTT is higher than typical, and L4S does not change that. So, there is no additional requirement in Section 4.3 for high RTT L4S flows to remove RTT bias - they can but they don't have to.

A.1.7. Scaling down to fractional congestion windows

Description: A scalable congestion control needs to remain responsive to congestion when typical RTTs over the public Internet are significantly smaller because they are no longer inflated by queuing delay (see the precise normative requirement wording in Section 4.3).

Motivation: As currently specified, the minimum congestion window of ECN-capable TCP (and its derivatives) is expected to be 2 sender maximum segment sizes (SMSS), or 1 SMSS after a retransmission timeout. Once the congestion window reaches this minimum, if there is further ECN-marking, TCP is meant to wait for a retransmission timeout before sending another segment (see section 6.1.2 of the ECN spec [RFC3168]). In practice, most known window-based congestion control algorithms become unresponsive to ECN congestion signals at this point. No matter how much ECN marking, the congestion window no longer reduces. Instead, the sender's lack of any further congestion response forces the queue to grow, overriding any AQM and increasing queuing delay (making the window large enough to become responsive again). This can result in a stable but deeper queue, or it might drive the queue to loss, then the retransmission timeout mechanism acts as a backstop.

Most window-based congestion controls for other transport protocols have a similar minimum window, albeit when measured in bytes for those that use smaller packets.

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

Unless scalable congestion controls address the requirement in Section 4.3 from the start, they will frequently become unresponsive to ECN, negating the low latency benefit of L4S, for themselves and for others.

That would seem to imply that scalable congestion controllers ought to be required to be able work with a congestion window less than 1 SMSS. For instance, if an ECN-capable TCP gets an ECN-mark when it is already sitting at a window of 1 SMSS, RFC 3168 requires it to defer sending for a retransmission timeout. A less drastic but more

complex mechanism can maintain a congestion window less than 1 SMSS (significantly less if necessary), as described in [Ahmed19]. Other approaches are likely to be feasible.

However, the requirement in Section 4.3 is worded as a "SHOULD" because it is believed that the existence of a minimum window is not all bad. When competing with an unresponsive flow, a minimum window naturally protects the flow from starvation by at least keeping some data flowing.

By stating the requirement to go lower than 1 SMSS as a "SHOULD", while the requirement in RFC 3168 still stands as well, we shall be able to watch the choices of minimum window evolve in different scalable congestion controllers.

A.1.8. Measuring Reordering Tolerance in Time Units

Description: When detecting loss, a scalable congestion control needs to be tolerant to reordering over an adaptive time interval, which scales with throughput, rather than counting only in fixed units of packets, which does not scale (see the precise normative requirement wording in Section 4.3).

Motivation: A primary purpose of L4S is scalable throughput (it's in the name). Scalability in all dimensions is, of course, also a goal of all IETF technology. The inverse linear congestion response in Section 4.3 is necessary, but not sufficient, to solve the congestion control scalability problem identified in [RFC3649]. As well as maintaining frequent ECN signals as rate scales, it is also important to ensure that a potentially false perception of loss does not limit throughput scaling.

End-systems cannot know whether a missing packet is due to loss or reordering, except in hindsight - if it appears later. So they can only deem that there has been a loss if a gap in the sequence space has not been filled, either after a certain number of subsequent packets has arrived (e.g. the 3 DupACK rule of standard TCP congestion control [RFC5681]) or after a certain amount of time (e.g. the RACK approach [RFC8985]).

As we attempt to scale packet rate over the years:

- * Even if only some sending hosts still deem that loss has occurred by counting reordered packets, all networks will have to keep reducing the time over which they keep packets in order. If some link technologies keep the time within which reordering occurs roughly unchanged, then loss over these links, as perceived by these hosts, will appear to continually rise over the years.

- * In contrast, if all senders detect loss in units of time, the time over which the network has to keep packets in order stays roughly invariant.

Therefore hosts have an incentive to detect loss in time units (so as not to fool themselves too often into detecting losses when there are none). And for hosts that are changing their congestion control implementation to L4S, there is no downside to including time-based loss detection code in the change (loss recovery implemented in hardware is an exception, covered later). Therefore requiring L4S hosts to detect loss in time-based units would not be a burden.

If the requirement in Section 4.3 were not placed on L4S hosts, even though it would be no burden on hosts to comply, all networks would face unnecessary uncertainty over whether some L4S hosts might be detecting loss by counting packets. Then all link technologies will have to unnecessarily keep reducing the time within which reordering occurs. That is not a problem for some link technologies, but it becomes increasingly challenging for other link technologies to continue to scale, particularly those relying on channel bonding for scaling, such as LTE, 5G and DOCSIS.

Given Internet paths traverse many link technologies, any scaling limit for these more challenging access link technologies would become a scaling limit for the Internet as a whole.

It might be asked how it helps to place this loss detection requirement only on L4S hosts, because networks will still face uncertainty over whether non-L4S flows are detecting loss by counting DupACKs. The answer is that those link technologies for which it is challenging to keep squeezing the reordering time will only need to do so for non-L4S traffic (which they can do because the L4S identifier is visible at the IP layer). Therefore, they can focus their processing and memory resources into scaling non-L4S (Classic) traffic. Then, the higher the proportion of L4S traffic, the less of a scaling challenge they will have.

To summarize, there is no reason for L4S hosts not to be part of the solution instead of part of the problem.

Requirement ("MUST") or recommendation ("SHOULD")? As explained above, this is a subtle interoperability issue between hosts and networks, which seems to need a "MUST". Unless networks can be certain that all L4S hosts follow the time-based approach, they still have to cater for the worst case - continually squeeze reordering into a smaller and smaller duration - just for hosts that might be using the counting approach. However, it was decided to express this as a recommendation, using "SHOULD". The main justification was that networks can still be fairly certain that L4S hosts will follow this recommendation, because following it offers only gain and no pain.

Details:

The speed of loss recovery is much more significant for short flows than long, therefore a good compromise is to adapt the reordering window; from a small fraction of the RTT at the start of a flow, to a larger fraction of the RTT for flows that continue for many round trips.

This is broadly the approach adopted by TCP RACK (Recent ACKnowledgements) [RFC8985]. However, RACK starts with the 3 DupACK approach, because the RTT estimate is not necessarily stable. As long as the initial window is paced, such initial use of 3 DupACK counting would amount to time-based loss detection and therefore would satisfy the time-based loss detection recommendation of Section 4.3. This is because pacing of the initial window would ensure that 3 DupACKs early in the connection would be spread over a small fraction of the round trip.

As mentioned above, hardware implementations of loss recovery using DupACK counting exist (e.g. some implementations of RoCEv2 for RDMA). For low latency, these implementations can change their congestion control to implement L4S, because the congestion control (as distinct from loss recovery) is implemented in software. But they cannot easily satisfy this loss recovery requirement. However, it is believed they do not need to, because such implementations are believed to solely exist in controlled environments, where the network technology keeps reordering extremely low anyway. This is why controlled environments with hardly any reordering are excluded from the scope of the normative recommendation in Section 4.3.

Detecting loss in time units also prevents the ACK-splitting attacks described in [Savage-TCP].

A.2. Scalable Transport Protocol Optimizations

A.2.1. Setting ECT in Control Packets and Retransmissions

Description: This item concerns TCP and its derivatives (e.g. SCTP) as well as RTP/RTCP [RFC6679]. The original specification of ECN for TCP precluded the use of ECN on control packets and retransmissions. Similarly RFC 6679 precludes the use of ECT on RTCP datagrams, in case the path changes after it has been checked for ECN traversal. To improve performance, scalable transport protocols ought to enable ECN at the IP layer in TCP control packets (SYN, SYN-ACK, pure ACKs, etc.) and in retransmitted packets. The same is true for other transports, e.g. SCTP, RTCP.

Motivation (TCP): RFC 3168 prohibits the use of ECN on these types of TCP packet, based on a number of arguments. This means these packets are not protected from congestion loss by ECN, which considerably harms performance, particularly for short flows. ECN++ [I-D.ietf-tcpm-generalized-ecn] proposes experimental use of ECN on all types of TCP packet as long as AccECN feedback [I-D.ietf-tcpm-accurate-ecn] is available (which itself satisfies the accurate feedback requirement in Section 4.2 for using a scalable congestion control).

Motivation (RTCP): L4S experiments in general will need to observe the rule in the RTP ECN spec [RFC6679] that precludes ECT on RTCP datagrams. Nonetheless, as ECN usage becomes more widespread, it would be useful to conduct specific experiments with ECN-capable RTCP to gather data on whether such caution is necessary.

A.2.2. Faster than Additive Increase

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

Motivation: As currently defined [RFC8257], DCTCP uses the traditional Reno additive increase in congestion avoidance phase. When the available capacity suddenly increases (e.g. when another flow finishes, or if radio capacity increases) it can take very many round trips to take advantage of the new capacity. TCP Cubic [RFC8312] was designed to solve this problem, but as flow rates have continued to increase, the delay accelerating into available capacity has become prohibitive. See, for instance, the examples in

Section 5.1 of the L4S architecture [I-D.ietf-tsvwg-l4s-arch]. Even when out of its Reno-compatibility mode, every 8x scaling of Cubic's flow rate leads to 2x more acceleration delay.

In the steady state, DCTCP induces about 2 ECN marks per round trip, so it is possible to quickly detect when these signals have disappeared and seek available capacity more rapidly, while minimizing the impact on other flows (Classic and scalable) [LinuxPacedChirping]. Alternatively, approaches such as Adaptive Acceleration (A2DTCP [A2DTCP]) have been proposed to address this problem in data centres, which might be deployable over the public Internet.

A.2.3. Faster Convergence at Flow Start

Description: It would improve performance if scalable congestion controls converged (reached their steady-state share of the capacity) faster than Classic congestion controls or at least no slower. This affects the flow start behaviour of any L4S congestion control derived from a Classic transport that uses TCP slow start, including those for real-time media.

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

Appendix B. Compromises in the Choice of L4S Identifier

This appendix is informative, not normative. As explained in Section 2, there is insufficient space in the IP header (v4 or v6) to fully accommodate every requirement. So the choice of L4S identifier involves tradeoffs. This appendix records the pros and cons of the choice that was made.

Non-normative recap of the chosen codepoint scheme:

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

- The ECT(1) codepoint signifies that the packet was sent by an L4S-capable sender.
- Given shortage of codepoints, both L4S and Classic ECN sides of an AQM have to use the same CE codepoint to indicate that a packet has experienced congestion. If a packet that had already been marked CE in an upstream buffer arrived at a subsequent AQM, this AQM would then have to guess whether to classify CE packets as L4S or Classic ECN. Choosing the L4S treatment is a safer choice, because then a few Classic packets might arrive early, rather than a few L4S packets arriving late.
- Additional information might be available if the classifier were transport-aware. Then it could classify a CE packet for Classic ECN treatment if the most recent ECT packet in the same flow had been marked ECT(0). However, the L4S service ought not to need transport-layer awareness.

Cons:

Consumes the last ECN codepoint: The L4S service could potentially supersede the service provided by Classic ECN, therefore using ECT(1) to identify L4S packets could ultimately mean that the ECT(0) codepoint was 'wasted' purely to distinguish one form of ECN from its successor.

ECN hard in some lower layers: It is not always possible to support the equivalent of an IP-ECN field in an AQM acting in a buffer below the IP layer [I-D.ietf-tsvwg-ecn-encap-guidelines]. Then, depending on the lower layer scheme, the L4S service might have to drop rather than mark frames even though they might encapsulate an ECN-capable packet.

Risk of reordering Classic CE packets within a flow: Classifying all CE packets into the L4S queue risks any CE packets that were originally ECT(0) being incorrectly classified as L4S. If there were delay in the Classic queue, these incorrectly classified CE packets would arrive early, which is a form of reordering. Reordering within a microflow can cause TCP senders (and senders of similar transports) to retransmit spuriously. However, the risk of spurious retransmissions would be extremely low for the following reasons:

1. It is quite unusual to experience queuing at more than one bottleneck on the same path (the available capacities have to be identical).
2. In only a subset of these unusual cases would the first bottleneck support Classic ECN marking while the second supported L4S ECN marking, which would be the only scenario where some ECT(0) packets could be CE marked by an AQM supporting Classic ECN then the remainder experienced further delay through the Classic side of a subsequent L4S DualQ AQM.
3. Even then, when a few packets are delivered early, it takes very unusual conditions to cause a spurious retransmission, in contrast to when some packets are delivered late. The first bottleneck has to apply CE-marks to at least N contiguous packets and the second bottleneck has to inject an uninterrupted sequence of at least N of these packets between two packets earlier in the stream (where N is the reordering window that the transport protocol allows before it considers a packet is lost).

For example consider $N=3$, and consider the sequence of packets 100, 101, 102, 103,... and imagine that packets 150,151,152 from later in the flow are injected as follows: 100, 150, 151, 101, 152, 102, 103... If this were late reordering, even one packet arriving out of sequence would trigger a spurious retransmission, but there is no spurious retransmission here with early reordering, because packet 101 moves the cumulative ACK counter forward before 3 packets have arrived out of order. Later, when packets 148, 149, 153... arrive, even though there is a 3-packet hole, there will be no problem, because the packets to fill the hole are already in the receive buffer.

4. Even with the current TCP recommendation of $N=3$ [RFC5681] spurious retransmissions will be unlikely for all the above reasons. As RACK [RFC8985] is becoming widely deployed, it tends to adapt its reordering window to a larger value of N , which will make the chance of a contiguous sequence of N early arrivals vanishingly small.
5. Even a run of 2 CE marks within a Classic ECN flow is unlikely, given FQ-CoDel is the only known widely deployed AQM that supports Classic ECN marking and it takes great care to separate out flows and to space any markings evenly along each flow.

It is extremely unlikely that the above set of 5 eventualities that are each unusual in themselves would all happen simultaneously. But, even if they did, the consequences would hardly be dire: the odd spurious fast retransmission. Whenever the traffic source (a Classic congestion control) mistakes the reordering of a string of CE marks for a loss, one might think that it will reduce its congestion window as well as emitting a spurious retransmission. However, it would have already reduced its congestion window when the CE markings arrived early. If it is using ABE [RFC8511], it might reduce cwnd a little more for a loss than for a CE mark. But it will revert that reduction once it detects that the retransmission was spurious.

In conclusion, the impact of early reordering on spurious retransmissions due to CE being ambiguous will generally be vanishingly small.

Insufficient anti-replay window in some pre-existing VPNs: If delay is reduced for a subset of the flows within a VPN, the anti-replay feature of some VPNs is known to potentially mistake the difference in delay for a replay attack. Section 6.2 recommends that the anti-replay window at the VPN egress is sufficiently sized, as required by the relevant specifications. However, in some VPN implementations the maximum anti-replay window is insufficient to cater for a large delay difference at prevailing packet rates. Section 6.2 suggests alternative work-rounds for such cases, but end-users using L4S over a VPN will need to be able to recognize the symptoms of this problem, in order to seek out these work-rounds.

Hard to distinguish Classic ECN AQM: With this scheme, when a source receives ECN feedback, it is not explicitly clear which type of AQM generated the CE markings. This is not a problem for Classic ECN sources that send ECT(0) packets, because an L4S AQM will recognize the ECT(0) packets as Classic and apply the appropriate Classic ECN marking behaviour.

However, in the absence of explicit disambiguation of the CE markings, an L4S source needs to use heuristic techniques to work out which type of congestion response to apply (see Appendix A.1.5). Otherwise, if long-running Classic flow(s) are sharing a Classic ECN AQM bottleneck with long-running L4S flow(s), which then apply an L4S response to Classic CE signals, the L4S flows would outcompete the Classic flow(s). Experiments have shown that L4S flows can take about 20 times more capacity share than equivalent Classic flows. Nonetheless, as link capacity reduces (e.g. to 4 Mb/s), the inequality reduces. So Classic flows always make progress and are not starved.

When L4S was first proposed (in 2015, 14 years after the Classic ECN spec [RFC3168] was published), it was believed that Classic ECN AQMs had failed to be deployed, because research measurements had found little or no evidence of CE marking. In subsequent years Classic ECN was included in per-flow-queuing (FQ) deployments, however an FQ scheduler stops an L4S flow outcompeting Classic, because it enforces equality between flow rates. It is not known whether there have been any non-FQ deployments of Classic ECN AQMs in the subsequent years, or whether there will be in future.

An algorithm for detecting a Classic ECN AQM as soon as a flow stabilizes after start-up has been proposed [ecn-fallback] (see Appendix A.1.5 for a brief summary). Testbed evaluations of v2 of the algorithm have shown detection is reasonably good for Classic ECN AQMs, in a wide range of circumstances. However, although it can correctly detect an L4S ECN AQM in many circumstances, it is often incorrect at low link capacities and/or high RTTs. Although this is the safe way round, there is a danger that it will discourage use of the algorithm.

Non-L4S service for control packets: Solely for the case of TCP, the Classic ECN RFCs [RFC3168] and [RFC5562] require a sender to clear the ECN field to Not-ECT on retransmissions and on certain control packets specifically pure ACKs, window probes and SYNs. When L4S packets are classified by the ECN field, these TCP control packets would not be classified into an L4S queue, and could therefore be delayed relative to the other packets in the flow. This would not cause reordering (because retransmissions are already out of order, and these control packets typically carry no data). However, it would make critical TCP control packets more vulnerable to loss and delay. To address this problem, ECN++ [I-D.ietf-tcpm-generalized-ecn] proposes an experiment in which all TCP control packets and retransmissions are ECN-capable as long as appropriate ECN feedback is available in each case.

Pros:

Should work e2e: The ECN field generally propagates end-to-end across the Internet without being wiped or mangled, at least over fixed networks. Unlike the DSCP, the setting of the ECN field is at least meant to be forwarded unchanged by networks that do not support ECN.

Should work in tunnels: The L4S identifiers work across and within

any tunnel that propagates the ECN field in any of the variant ways it has been defined since ECN-tunneling was first specified in the year 2001 [RFC3168]. However, it is likely that some tunnels still do not implement ECN propagation at all.

Should work for many link technologies: At most, but not all, path bottlenecks there is IP-awareness, so that L4S AQMs can be located where the IP-ECN field can be manipulated. Bottlenecks at lower layer nodes without IP-awareness either have to use drop to signal congestion or a specific congestion notification facility has to be defined for that link technology, including propagation to and from IP-ECN. The programme to define these is progressing and in each case so far the scheme already defined for ECN inherently supports L4S as well (see Section 6.1).

Could migrate to one codepoint: If all Classic ECN senders eventually evolve to use the L4S service, the ECT(0) codepoint could be reused for some future purpose, but only once use of ECT(0) packets had reduced to zero, or near-zero, which might never happen.

L4 not required: Being based on the ECN field, this scheme does not need the network to access transport layer flow identifiers. Nonetheless, it does not preclude solutions that do.

Appendix C. Potential Competing Uses for the ECT(1) Codepoint

The ECT(1) codepoint of the ECN field has already been assigned once for the ECN nonce [RFC3540], which has now been categorized as historic [RFC8311]. ECN is probably the only remaining field in the Internet Protocol that is common to IPv4 and IPv6 and still has potential to work end-to-end, with tunnels and with lower layers. Therefore, ECT(1) should not be reassigned to a different experimental use (L4S) without carefully assessing competing potential uses. These fall into the following categories:

C.1. Integrity of Congestion Feedback

Receiving hosts can fool a sender into downloading faster by suppressing feedback of ECN marks (or of losses if retransmissions are not necessary or available otherwise).

The historic ECN nonce protocol [RFC3540] proposed that a TCP sender could set either of ECT(0) or ECT(1) in each packet of a flow and remember the sequence it had set. If any packet was lost or congestion marked, the receiver would miss that bit of the sequence. An ECN Nonce receiver had to feed back the least significant bit of the sum, so it could not suppress feedback of a loss or mark without a 50-50 chance of guessing the sum incorrectly.

It is highly unlikely that ECT(1) will be needed for integrity protection in future. The ECN Nonce RFC [RFC3540] has been reclassified as historic, partly because other ways have been developed to protect feedback integrity of TCP and other transports [RFC8311] that do not consume a codepoint in the IP header. For instance:

- * the sender can test the integrity of the receiver's feedback by occasionally setting the IP-ECN field to a value normally only set by the network. Then it can test whether the receiver's feedback faithfully reports what it expects (see para 2 of Section 20.2 of the ECN spec [RFC3168]). This works for loss and it will work for the accurate ECN feedback [RFC7560] intended for L4S.
- * A network can enforce a congestion response to its ECN markings (or packet losses) by auditing congestion exposure (ConEx) [RFC7713]. Whether the receiver or a downstream network is suppressing congestion feedback or the sender is unresponsive to the feedback, or both, ConEx audit can neutralise any advantage that any of these three parties would otherwise gain.
- * The TCP authentication option (TCP-AO [RFC5925]) can be used to detect any tampering with TCP congestion feedback (whether malicious or accidental). TCP's congestion feedback fields are immutable end-to-end, so they are amenable to TCP-AO protection, which covers the main TCP header and TCP options by default. However, TCP-AO is often too brittle to use on many end-to-end paths, where middleboxes can make verification fail in their attempts to improve performance or security, e.g. by resegmentation or shifting the sequence space.

C.2. Notification of Less Severe Congestion than CE

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

Before assigning ECT(1) as an identifier for L4S, we must carefully consider whether it might be better to hold ECT(1) in reserve for future standardisation of rapid flow acceleration, which is an important and enduring problem [RFC6077].

Pre-Congestion Notification (PCN) is another scheme that assigns alternative semantics to the ECN field. It uses ECT(1) to signify a less severe level of pre-congestion notification than CE [RFC6660]. However, the ECN field only takes on the PCN semantics if packets carry a Diffserv codepoint defined to indicate PCN marking within a controlled environment. PCN is required to be applied solely to the outer header of a tunnel across the controlled region in order not to interfere with any end-to-end use of the ECN field. Therefore a PCN region on the path would not interfere with the L4S service identifier defined in Section 3.

Authors' Addresses

Koen De Schepper
Nokia Bell Labs
Antwerp
Belgium
Email: koen.de_schepper@nokia.com
URI: https://www.bell-labs.com/usr/koen.de_schepper

Bob Briscoe (editor)
Independent
United Kingdom
Email: ietf@bobbriscoe.net
URI: <http://bobbriscoe.net/>

TSVWG
Internet-Draft
Updates: 6363 (if approved)
Intended status: Standards Track
Expires: July 15, 2019

V. Roca
INRIA
A. Begen
Networked Media
January 11, 2019

Forward Error Correction (FEC) Framework Extension to Sliding Window
Codes
draft-ietf-tsvwg-fecframe-ext-08

Abstract

RFC 6363 describes a framework for using Forward Error Correction (FEC) codes to provide protection against packet loss. The framework supports applying FEC to arbitrary packet flows over unreliable transport and is primarily intended for real-time, or streaming, media. However, FECFRAME as per RFC 6363 is restricted to block FEC codes. This document updates RFC 6363 to support FEC Codes based on a sliding encoding window, in addition to Block FEC Codes, in a backward-compatible way. During multicast/broadcast real-time content delivery, the use of sliding window codes significantly improves robustness in harsh environments, with less repair traffic and lower FEC-related added latency.

Status of This Memo

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

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

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

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

Copyright Notice

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

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

Table of Contents

1. Introduction	2
2. Definitions and Abbreviations	4
3. Summary of Architecture Overview	7
4. Procedural Overview	10
4.1. General	10
4.2. Sender Operation with Sliding Window FEC Codes	10
4.3. Receiver Operation with Sliding Window FEC Codes	13
5. Protocol Specification	15
5.1. General	15
5.2. FEC Framework Configuration Information	16
5.3. FEC Scheme Requirements	16
6. Feedback	16
7. Transport Protocols	17
8. Congestion Control	17
9. Implementation Status	17
10. Security Considerations	17
11. Operations and Management Considerations	18
12. IANA Considerations	18
13. Acknowledgments	18
14. References	18
14.1. Normative References	18
14.2. Informative References	19
Appendix A. About Sliding Encoding Window Management (informational)	20
Authors' Addresses	21

1. Introduction

Many applications need to transport a continuous stream of packetized data from a source (sender) to one or more destinations (receivers) over networks that do not provide guaranteed packet delivery. In particular packets may be lost, which is strictly the focus of this document: we assume that transmitted packets are either lost (e.g., because of a congested router, of a poor signal-to-noise ratio in a wireless network, or because the number of bit errors exceeds the correction capabilities of the physical-layer error correcting code)

or received by the transport protocol without any corruption (i.e., the bit-errors, if any, have been fixed by the physical-layer error correcting code and therefore are hidden to the upper layers).

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

Then FECFRAME [RFC6363] provides a framework to define Content Delivery Protocols (CDPs) that provide FEC protection for arbitrary packet flows over an unreliable datagram service transport such as UDP. It is primarily intended for real-time or streaming media applications, using broadcast, multicast, or on-demand delivery.

However, [RFC6363] only considers block FEC schemes defined in accordance with the FEC Building Block [RFC5052] (e.g., [RFC6681], [RFC6816] or [RFC6865]). These codes require the input flow(s) to be segmented into a sequence of blocks. Then FEC encoding (at a sender or an encoding middlebox) and decoding (at a receiver or a decoding middlebox) are both performed on a per-block basis. For instance, if the current block encompasses the 100's to 119's source symbols (i.e., a block of size 20 symbols) of an input flow, encoding (and decoding) will be performed on this block independently of other blocks. This approach has major impacts on FEC encoding and decoding delays. The data packets of continuous media flow(s) may be passed to the transport layer immediately, without delay. But the block creation time, that depends on the number of source symbols in this block, impacts both the FEC encoding delay (since encoding requires that all source symbols be known), and mechanically the packet loss recovery delay at a receiver (since no repair symbol for the current block can be generated and therefore received before that time). Therefore a good value for the block size is necessarily a balance between the maximum FEC decoding latency at the receivers (which must be in line with the most stringent real-time requirement of the protected flow(s), hence an incentive to reduce the block size), and the desired robustness against long loss bursts (which increases with the block size, hence an incentive to increase this size).

This document updates [RFC6363] in order to also support FEC codes based on a sliding encoding window (A.K.A. convolutional codes)

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

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

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

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

2. Definitions and Abbreviations

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

Application Data Unit (ADU): The unit of source data provided as payload to the transport layer. For instance, it can be a

payload containing the result of the RTP packetization of a compressed video frame.

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

AL-FEC: Application-layer Forward Error Correction.

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

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

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

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

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

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

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

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

FEC Repair Packet: At a sender (respectively, at a receiver), a payload submitted to (respectively, received from) the transport

protocol containing one or more repair symbols along with a Repair FEC Payload ID and possibly an RTP header.

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

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

Repair Flow: The packet flow carrying FEC data.

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

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

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

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

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

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

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

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

Encoding Symbol: Unit of data generated by the encoding process. With systematic codes, source symbols are part of the encoding symbols.

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

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

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

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

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

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

3. Summary of Architecture Overview

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

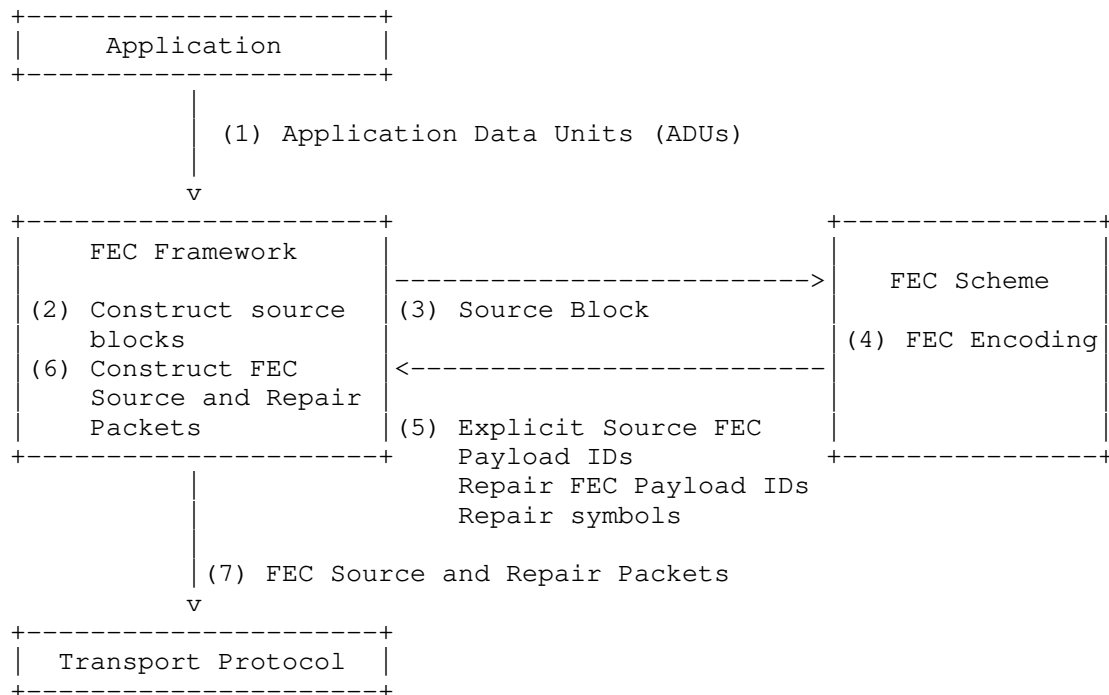


Figure 1: FECFRAME architecture at a sender.

The FECFRAME architecture is illustrated in Figure 1 from the sender's point of view, in case of a block FEC Scheme. It shows an application generating an ADU flow (other flows, from other applications, may co-exist). These ADUs, of variable size, must be somehow mapped to source symbols of fixed size (this fixed size is a requirement of all FEC Schemes that comes from the way mathematical operations are applied to symbols content). This is the goal of an ADU-to-symbols mapping process that is FEC-Scheme specific (see below). Once the source block is built, taking into account both the FEC Scheme constraints (e.g., in terms of maximum source block size) and the application's flow constraints (e.g., in terms of real-time constraints), the associated source symbols are handed to the FEC Scheme in order to produce an appropriate number of repair symbols. FEC Source Packets (containing ADUs) and FEC Repair Packets (containing one or more repair symbols each) are then generated and sent using an appropriate transport protocol (more precisely [RFC6363], Section 7, requires a transport protocol providing an unreliable datagram service, such as UDP). In practice FEC Source Packets may be passed to the transport layer as soon as available, without having to wait for FEC encoding to take place. In that case

a copy of the associated source symbols needs to be kept within FECFRAME for future FEC encoding purposes.

At a receiver (not shown), FECFRAME processing operates in a similar way, taking as input the incoming FEC Source and Repair Packets received. In case of FEC Source Packet losses, the FEC decoding of the associated block may recover all (in case of successful decoding) or a subset potentially empty (otherwise) of the missing source symbols. After source-symbol-to-ADU mapping, when lost ADUs are recovered, they are then assigned to their respective flow (see below). ADUs are returned to the application(s), either in their initial transmission order (in that case ADUs received after an erased one will be delayed until FEC decoding has taken place) or not (in that case each ADU is returned as soon as it is received or recovered), depending on the application requirements.

FECFRAME features two subtle mechanisms:

- o ADUs-to-source-symbols mapping: in order to manage variable size ADUs, FECFRAME and FEC Schemes can use small, fixed size symbols and create a mapping between ADUs and symbols. To each ADU this mechanism prepends a length field (plus a flow identifier, see below) and pads the result to a multiple of the symbol size. A small ADU may be mapped to a single source symbol while a large one may be mapped to multiple symbols. The mapping details are FEC-Scheme-dependent and must be defined in the associated document;
- o Assignment of decoded ADUs to flows in multi-flow configurations: when multiple flows are multiplexed over the same FECFRAME instance, a problem is to assign a decoded ADU to the right flow (UDP port numbers and IP addresses traditionally used to map incoming ADUs to flows are not recovered during FEC decoding). To make it possible, at the FECFRAME sending instance, each ADU is prepended with a flow identifier (1 byte) during the ADU-to-source-symbols mapping (see above). The flow identifiers are also shared between all FECFRAME instances as part of the FEC Framework Configuration Information. This (flow identifier + length + application payload + padding), called ADUI, is then FEC protected. Therefore a decoded ADUI contains enough information to assign the ADU to the right flow.

A few aspects are not covered by FECFRAME, namely:

- o [RFC6363] section 8 does not detail any congestion control mechanism, but only provides high level normative requirements;

- o the possibility of having feedbacks from receiver(s) is considered out of scope, although such a mechanism may exist within the application (e.g., through RTCP control messages);
- o flow adaptation at a FECFRAME sender (e.g., how to set the FEC code rate based on transmission conditions) is not detailed, but it needs to comply with the congestion control normative requirements (see above).

4. Procedural Overview

4.1. General

The general considerations of [RFC6363], Section 4.1, that are specific to block FEC codes are not repeated here.

With a Sliding Window FEC Code, the FEC Source Packet MUST contain information to identify the position occupied by the ADU within the source flow, in terms specific to the FEC Scheme. This information is known as the Source FEC Payload ID, and the FEC Scheme is responsible for defining and interpreting it.

With a Sliding Window FEC Code, the FEC Repair Packets MUST contain information that identifies the relationship between the contained repair payloads and the original source symbols used during encoding. This information is known as the Repair FEC Payload ID, and the FEC Scheme is responsible for defining and interpreting it.

The Sender Operation ([RFC6363], Section 4.2.) and Receiver Operation ([RFC6363], Section 4.3) are both specific to block FEC codes and therefore omitted below. The following two sections detail similar operations for Sliding Window FEC codes.

4.2. Sender Operation with Sliding Window FEC Codes

With a Sliding Window FEC Scheme, the following operations, illustrated in Figure 2 for the generic case (non-RTP repair flows), and in Figure 3 for the case of RTP repair flows, describe a possible way to generate compliant source and repair flows:

1. A new ADU is provided by the application.
2. The FEC Framework communicates this ADU to the FEC Scheme.
3. The sliding encoding window is updated by the FEC Scheme. The ADU-to-source-symbols mapping as well as the encoding window management details are both the responsibility of the FEC Scheme

and MUST be detailed there. Appendix A provides non-normative hints about what FEC Scheme designers need to consider;

4. The Source FEC Payload ID information of the source packet is determined by the FEC Scheme. If required by the FEC Scheme, the Source FEC Payload ID is encoded into the Explicit Source FEC Payload ID field and returned to the FEC Framework.
5. The FEC Framework constructs the FEC Source Packet according to [RFC6363] Figure 6, using the Explicit Source FEC Payload ID provided by the FEC Scheme if applicable.
6. The FEC Source Packet is sent using normal transport-layer procedures. This packet is sent using the same ADU flow identification information as would have been used for the original source packet if the FEC Framework were not present (e.g., the source and destination addresses and UDP port numbers on the IP datagram carrying the source packet will be the same whether or not the FEC Framework is applied).
7. When the FEC Framework needs to send one or several FEC Repair Packets (e.g., according to the target Code Rate), it asks the FEC Scheme to create one or several repair packet payloads from the current sliding encoding window along with their Repair FEC Payload ID.
8. The Repair FEC Payload IDs and repair packet payloads are provided back by the FEC Scheme to the FEC Framework.
9. The FEC Framework constructs FEC Repair Packets according to [RFC6363] Figure 7, using the FEC Payload IDs and repair packet payloads provided by the FEC Scheme.
10. The FEC Repair Packets are sent using normal transport-layer procedures. The port(s) and multicast group(s) to be used for FEC Repair Packets are defined in the FEC Framework Configuration Information.

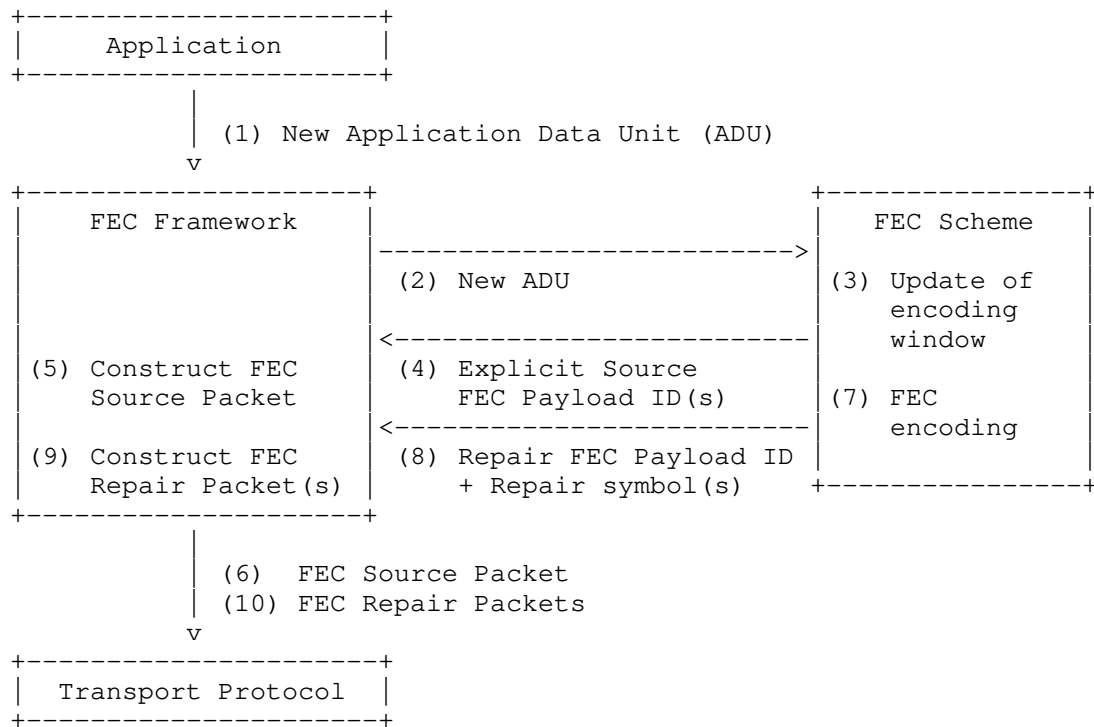


Figure 2: Sender Operation with Sliding Window FEC Codes

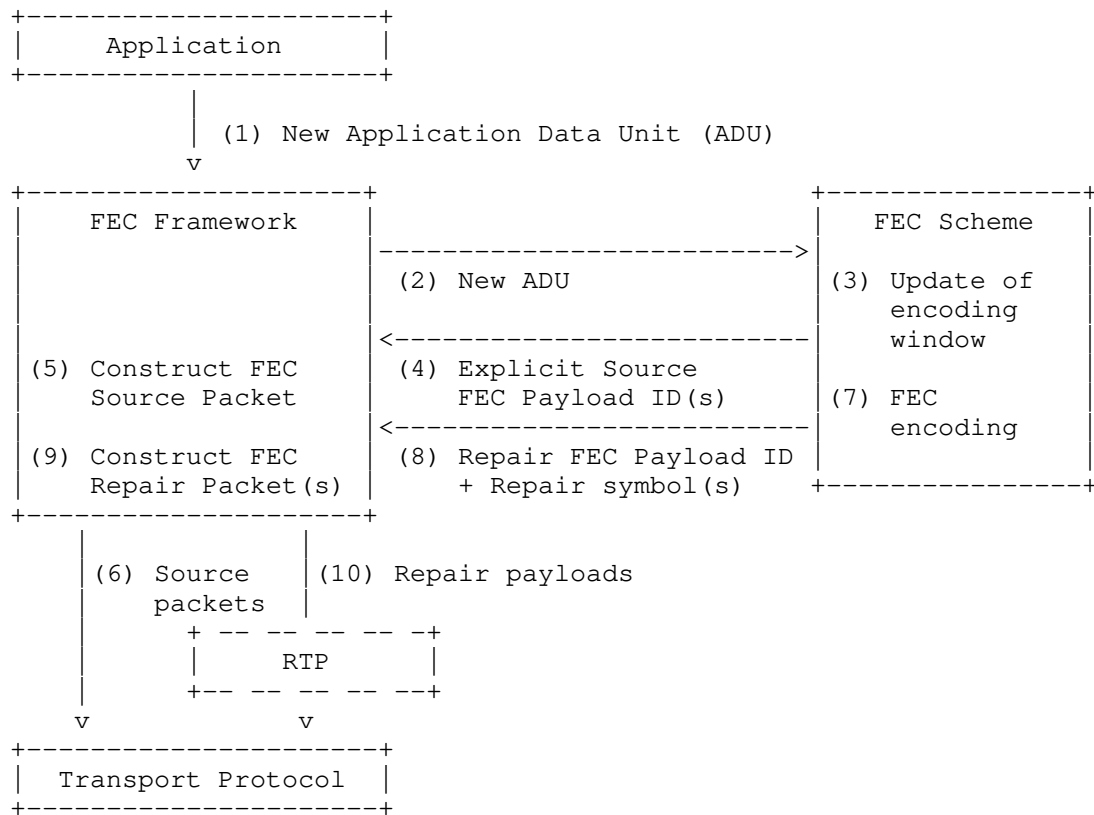


Figure 3: Sender Operation with Sliding Window FEC Codes and RTP Repair Flows

4.3. Receiver Operation with Sliding Window FEC Codes

With a Sliding Window FEC Scheme, the following operations, illustrated in Figure 4 for the generic case (non-RTP repair flows), and in Figure 5 for the case of RTP repair flows. The only differences with respect to block FEC codes lie in steps (4) and (5). Therefore this section does not repeat the other steps of [RFC6363], Section 4.3, "Receiver Operation". The new steps (4) and (5) are:

4. The FEC Scheme uses the received FEC Payload IDs (and derived FEC Source Payload IDs when the Explicit Source FEC Payload ID field is not used) to insert source and repair packets into the decoding window in the right way. If at least one source packet is missing and at least one repair packet has been received, then FEC decoding is attempted to recover missing source payloads. The FEC Scheme determines whether source packets have been lost

and whether enough repair packets have been received to decode any or all of the missing source payloads.

5. The FEC Scheme returns the received and decoded ADUs to the FEC Framework, along with indications of any ADUs that were missing and could not be decoded.

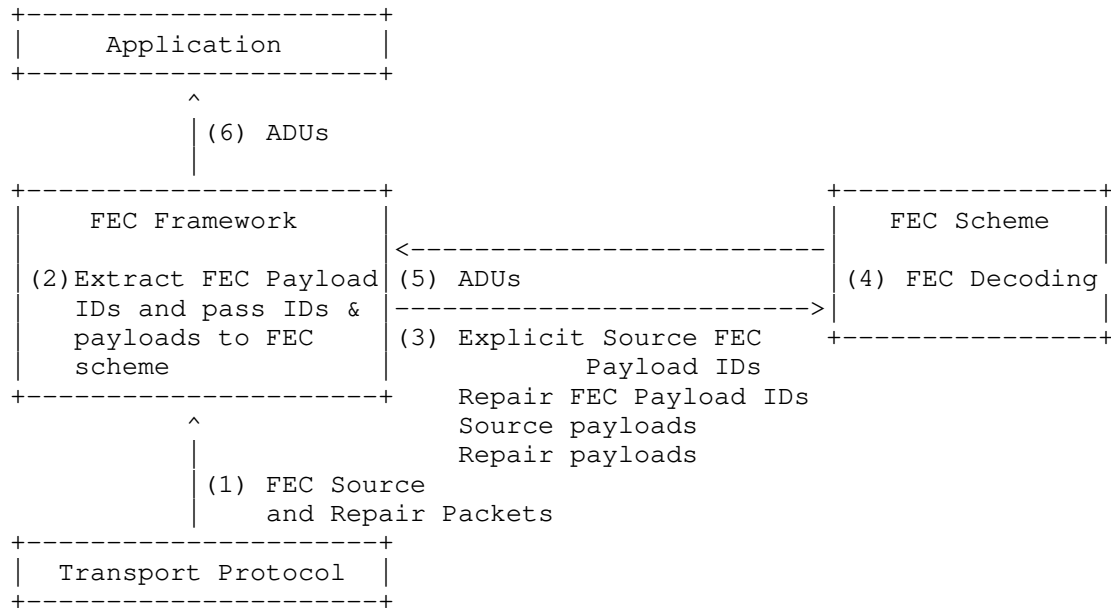


Figure 4: Receiver Operation with Sliding Window FEC Codes

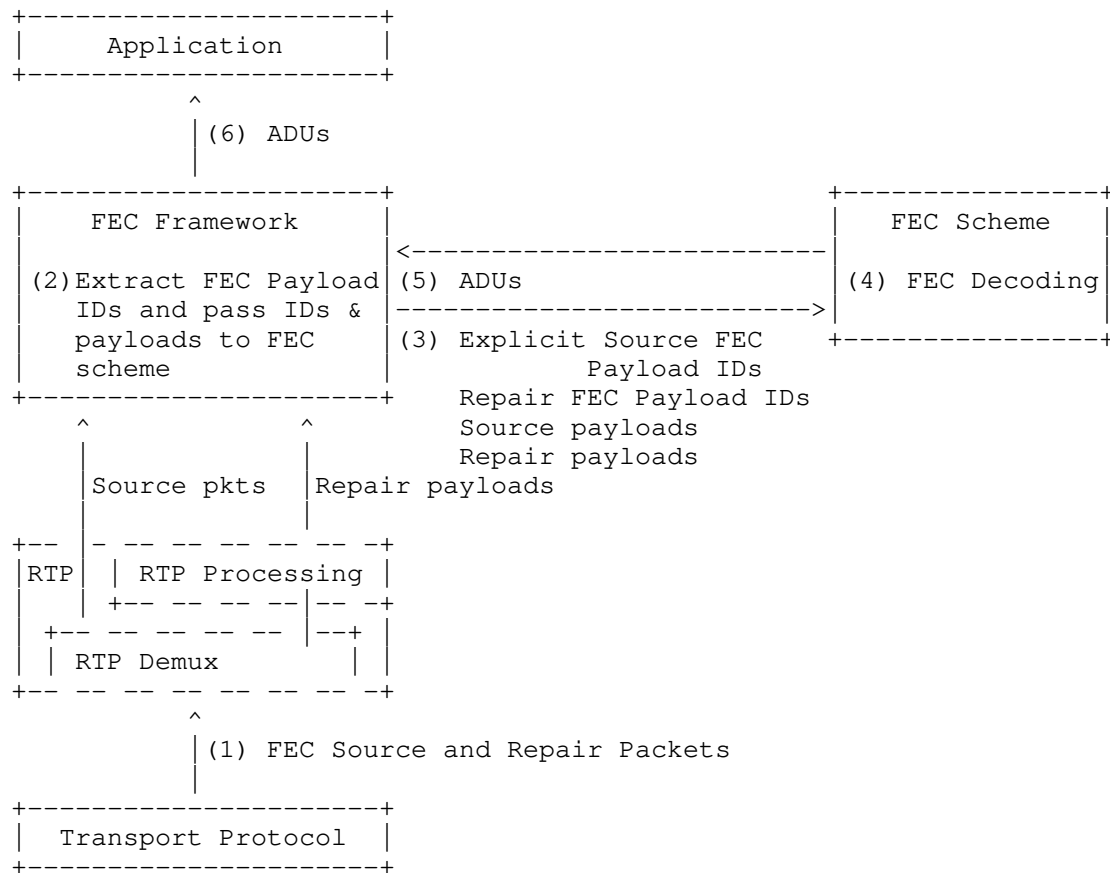


Figure 5: Receiver Operation with Sliding Window FEC Codes and RTP Repair Flows

5. Protocol Specification

5.1. General

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

5.2. FEC Framework Configuration Information

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

5.3. FEC Scheme Requirements

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

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

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

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

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

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

6. Feedback

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

7. Transport Protocols

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

8. Congestion Control

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

9. Implementation Status

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

An implementation of FECFRAME extended to Sliding Window codes exists:

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

10. Security Considerations

This FECFRAME extension does not add any new security consideration. All the considerations of [RFC6363], Section 9, apply to this document as well. However, for the sake of completeness, the

following goal can be added to the list provided in Section 9.1 "Problem Statement" of [RFC6363]:

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

11. Operations and Management Considerations

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

12. IANA Considerations

No IANA actions are required for this document.

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

13. Acknowledgments

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

14. References

14.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC6363] Watson, M., Begen, A., and V. Roca, "Forward Error Correction (FEC) Framework", RFC 6363, DOI 10.17487/RFC6363, October 2011, <<https://www.rfc-editor.org/info/rfc6363>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

14.2. Informative References

- [MBMSTS] 3GPP, "Multimedia Broadcast/Multicast Service (MBMS); Protocols and codecs", 3GPP TS 26.346, March 2009, <<http://ftp.3gpp.org/specs/html-info/26346.htm>>.
- [RFC5052] Watson, M., Luby, M., and L. Vicisano, "Forward Error Correction (FEC) Building Block", RFC 5052, DOI 10.17487/RFC5052, August 2007, <<https://www.rfc-editor.org/info/rfc5052>>.
- [RFC6364] Begen, A., "Session Description Protocol Elements for the Forward Error Correction (FEC) Framework", RFC 6364, DOI 10.17487/RFC6364, October 2011, <<https://www.rfc-editor.org/info/rfc6364>>.
- [RFC6681] Watson, M., Stockhammer, T., and M. Luby, "Raptor Forward Error Correction (FEC) Schemes for FECFRAME", RFC 6681, DOI 10.17487/RFC6681, August 2012, <<https://www.rfc-editor.org/info/rfc6681>>.
- [RFC6816] Roca, V., Cunche, M., and J. Lacan, "Simple Low-Density Parity Check (LDPC) Staircase Forward Error Correction (FEC) Scheme for FECFRAME", RFC 6816, DOI 10.17487/RFC6816, December 2012, <<https://www.rfc-editor.org/info/rfc6816>>.
- [RFC6865] Roca, V., Cunche, M., Lacan, J., Bouabdallah, A., and K. Matsuzono, "Simple Reed-Solomon Forward Error Correction (FEC) Scheme for FECFRAME", RFC 6865, DOI 10.17487/RFC6865, February 2013, <<https://www.rfc-editor.org/info/rfc6865>>.
- [RFC8406] Adamson, B., Adjih, C., Bilbao, J., Firoiu, V., Fitzek, F., Ghanem, S., Lochin, E., Masucci, A., Montpetit, M-J., Pedersen, M., Peralta, G., Roca, V., Ed., Saxena, P., and S. Sivakumar, "Taxonomy of Coding Techniques for Efficient Network Communications", RFC 8406, DOI 10.17487/RFC8406, June 2018, <<https://www.rfc-editor.org/info/rfc8406>>.
- [RLC-ID] Roca, V. and B. Teibi, "Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC) Scheme for FECFRAME", Work in Progress, Transport Area Working Group (TSVWG) draft-ietf-tsvwg-rlc-fec-scheme (Work in Progress), September 2018, <<https://tools.ietf.org/html/draft-ietf-tsvwg-rlc-fec-scheme>>.

Appendix A. About Sliding Encoding Window Management (informational)

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

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

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

- o after a certain delay, when an "old" ADU of a real-time flow times out. The source symbol retention delay in the sliding encoding window should therefore be initialized according to the real-time features of incoming flow(s) when applicable;
- o once the sliding encoding window has reached its maximum size (there is usually an upper limit to the sliding encoding window size). In that case the oldest symbol is removed each time a new source symbol is added.

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

- o at the source flows level: real-time constraints can limit the total time source symbols can remain in the encoding window;
- o at the FEC code level: theoretical or practical limitations (e.g., because of computational complexity) can limit the number of source symbols in the encoding window;
- o at the FEC Scheme level: signaling and window management are intrinsically related. For instance, an encoding window composed of a non-sequential set of source symbols requires an appropriate signaling to inform a receiver of the composition of the encoding window, and the associated transmission overhead can limit the maximum encoding window size. On the opposite, an encoding window always composed of a sequential set of source symbols simplifies signaling: providing the identity of the first source symbol plus their number is sufficient, which creates a fixed and relatively small transmission overhead.

Authors' Addresses

Vincent Roca
INRIA
Univ. Grenoble Alpes
France

EMail: vincent.roca@inria.fr

Ali Begen
Networked Media
Konya
Turkey

EMail: ali.begen@networked.media

Transport Area Working Group
Internet-Draft
Updates: 2474 (if approved)
Intended status: Standards Track
Expires: December 07, 2018

G. Fairhurst
University of Aberdeen
June 07, 2018

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.

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 respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	4
3. The update to RFC2474	4
4. Security Considerations	4
5. IANA Considerations	5
6. Acknowledgments	5
7. References	5
7.1. Normative References	6
7.2. Informative References	6
Appendix A. Revision Notes	6
Author's Address	7

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].

Pool	Codepoint Space
1	xxxxx0
2	xxxx11
3	xxxx01

Figure 1: Format of the field for codepoints allocated in the three IANA pools (where 'x' refers to either '0' or '1').

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.

Pool	Codepoint space	Assignment Policy
----	-----	-----
1	xxxxx0	Standards Action
2	xxxx11	EXP/LU
3	xxxx01	Standards Action

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

security considerations.

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

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2474] Nichols, K., Blake, S., Baker, F. and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <<http://www.rfc-editor.org/info/rfc2474>>.
- [RFC3260] Grossman, D., "New Terminology and Clarifications for Diffserv", RFC 3260, DOI 10.17487/RFC3260, April 2002, <<http://www.rfc-editor.org/info/rfc3260>>.
- [RFC8126] Cotton, M., Leiba, B. and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [Registry] IANA, "Differentiated Services Field Codepoints (DSCP)", <https://www.iana.org/assignments/dscp-registry/dscp-registry.xhtml>, .

7.2. Informative References

- [I-D.ietf-tsvwg-le-phb] Bless, R., "A Lower Effort Per-Hop Behavior (LE PHB)", Internet-Draft draft-ietf-tsvwg-le-phb-02, June 2017.
- [RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z. and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, DOI 10.17487/RFC2475, December 1998, <<http://www.rfc-editor.org/info/rfc2475>>.

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.

- o Thanks to Roland Bless for review comments.

Individual submission as draft -02 (author requests adoption as a TSVWG WG draft).

- o Thanks to David Black for review comments in preparing rev -02.

Working Group submission as draft -00

- o Adopted by the TSVWG working group.

Working Group submission as draft -01

- o Fixed exploded acronyms.

Working Group submission as draft -02

- o Corrections after WGLC.

Working Group submission as draft -03

- o Corrections after TSVWG Shepherd Review.

Working Group submission as draft -04

- o Added RFC 3260 as a necessary downref, with IANA asked to reference this.

Working Group submission as draft -05

- o Corrections following AD review.
- o 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

- o GenART feedback to changed assignment method to assignment policy,.
- o Correction to the IANA reference documents.

Working Group submission as draft -07

- o 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

- o Revised after AD feedback - definition of standards action.

Author's Address

Godred Fairhurst
University of Aberdeen
Department of Engineering
Fraser Noble Building
Aberdeen, AB24 3UE
Scotland

Email: gorry@erg.abdn.ac.uk
URI: <http://www.erg.abdn.ac.uk/>

Transport Area Working Group
Internet-Draft
Intended status: Informational
Expires: 5 September 2022

B. Briscoe, Ed.
Independent
K. De Schepper
Nokia Bell Labs
M. Bagnulo Braun
Universidad Carlos III de Madrid
G. White
CableLabs
4 March 2022

Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service:
Architecture
draft-ietf-tsvwg-l4s-arch-17

Abstract

This document describes the L4S architecture, which enables Internet applications to achieve Low queuing Latency, Low Loss, and Scalable throughput (L4S). The insight on which L4S is based is that the root cause of queuing delay is in the congestion controllers of senders, not in the queue itself. With the L4S architecture all Internet applications could (but do not have to) transition away from congestion control algorithms that cause substantial queuing delay, to a new class of congestion controls that induce very little queuing, aided by explicit congestion signalling from the network. This new class of congestion controls can provide low latency for capacity-seeking flows, so applications can achieve both high bandwidth and low latency.

The architecture primarily concerns incremental deployment. It defines mechanisms that allow the new class of L4S congestion controls to coexist with 'Classic' congestion controls in a shared network. These mechanisms aim to ensure that the latency and throughput performance using an L4S-compliant congestion controller is usually much better (and rarely worse) than performance would have been using a 'Classic' congestion controller, and that competing flows continuing to use 'Classic' controllers are typically not impacted by the presence of L4S. These characteristics are important to encourage adoption of L4S congestion control algorithms and L4S compliant network elements.

The L4S architecture consists of three components: network support to isolate L4S traffic from classic traffic; protocol features that allow network elements to identify L4S traffic; and host support for L4S congestion controls.

Status of This Memo

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

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

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

This Internet-Draft will expire on 5 September 2022.

Copyright Notice

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

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

Table of Contents

1. Introduction	3
1.1. Document Roadmap	5
2. L4S Architecture Overview	5
3. Terminology	7
4. L4S Architecture Components	9
4.1. Protocol Mechanisms	9
4.2. Network Components	10
4.3. Host Mechanisms	13
5. Rationale	15
5.1. Why These Primary Components?	15
5.2. What L4S adds to Existing Approaches	18
6. Applicability	21
6.1. Applications	21
6.2. Use Cases	22
6.3. Applicability with Specific Link Technologies	24

6.4.	Deployment Considerations	24
6.4.1.	Deployment Topology	25
6.4.2.	Deployment Sequences	26
6.4.3.	L4S Flow but Non-ECN Bottleneck	28
6.4.4.	L4S Flow but Classic ECN Bottleneck	29
6.4.5.	L4S AQM Deployment within Tunnels	29
7.	IANA Considerations (to be removed by RFC Editor)	30
8.	Security Considerations	30
8.1.	Traffic Rate (Non-)Policing	30
8.2.	'Latency Friendliness'	31
8.3.	Interaction between Rate Policing and L4S	33
8.4.	ECN Integrity	34
8.5.	Privacy Considerations	34
9.	Acknowledgements	35
10.	Informative References	35
	Authors' Addresses	45

1. Introduction

At any one time, it is increasingly common for all of the traffic in a bottleneck link (e.g. a household's Internet access) to come from applications that prefer low delay: interactive Web, Web services, voice, conversational video, interactive video, interactive remote presence, instant messaging, online gaming, remote desktop, cloud-based applications and video-assisted remote control of machinery and industrial processes. In the last decade or so, much has been done to reduce propagation delay by placing caches or servers closer to users. However, queuing remains a major, albeit intermittent, component of latency. For instance spikes of hundreds of milliseconds are not uncommon, even with state-of-the-art active queue management (AQM) [COBALT], [DOCSIS3AQM]. Queuing in access network bottlenecks is typically configured to cause overall network delay to roughly double during a long-running flow, relative to expected base (unloaded) path delay [BufferSize]. Low loss is also important because, for interactive applications, losses translate into even longer retransmission delays.

It has been demonstrated that, once access network bit rates reach levels now common in the developed world, increasing capacity offers diminishing returns if latency (delay) is not addressed [Dukkipati06], [Rajiullah15]. Therefore, the goal is an Internet service with very Low queueing Latency, very Low Loss and Scalable throughput (L4S). Very low queueing latency means less than 1 millisecond (ms) on average and less than about 2 ms at the 99th percentile. This document describes the L4S architecture for achieving these goals.

Differentiated services (Diffserv) offers Expedited Forwarding (EF [RFC3246]) for some packets at the expense of others, but this makes no difference when all (or most) of the traffic at a bottleneck at any one time requires low latency. In contrast, L4S still works well when all traffic is L4S - a service that gives without taking needs none of the configuration or management baggage (traffic policing, traffic contracts) associated with favouring some traffic flows over others.

Queuing delay degrades performance intermittently [Hohlfeld14]. It occurs when a large enough capacity-seeking (e.g. TCP) flow is running alongside the user's traffic in the bottleneck link, which is typically in the access network. Or when the low latency application is itself a large capacity-seeking or adaptive rate (e.g. interactive video) flow. At these times, the performance improvement from L4S must be sufficient that network operators will be motivated to deploy it.

Active Queue Management (AQM) is part of the solution to queuing under load. AQM improves performance for all traffic, but there is a limit to how much queuing delay can be reduced by solely changing the network; without addressing the root of the problem.

The root of the problem is the presence of standard TCP congestion control (Reno [RFC5681]) or compatible variants (e.g. TCP Cubic [RFC8312]). We shall use the term 'Classic' for these Reno-friendly congestion controls. Classic congestion controls induce relatively large saw-tooth-shaped excursions up the queue and down again, which have been growing as flow rate scales [RFC3649]. So if a network operator naively attempts to reduce queuing delay by configuring an AQM to operate at a shallower queue, a Classic congestion control will significantly underutilize the link at the bottom of every saw-tooth.

It has been demonstrated that if the sending host replaces a Classic congestion control with a 'Scalable' alternative, when a suitable AQM is deployed in the network the performance under load of all the above interactive applications can be significantly improved. For instance, queuing delay under heavy load with the example DCTCP/DualQ solution cited below on a DSL or Ethernet link is roughly 1 to 2 milliseconds at the 99th percentile without losing link utilization [DualPI2Linux], [DCTtH19] (for other link types, see Section 6.3). This compares with 5-20 ms on average with a Classic congestion control and current state-of-the-art AQMs such as FQ-CoDel [RFC8290], PIE [RFC8033] or DOCSIS PIE [RFC8034] and about 20-30 ms at the 99th percentile [DualPI2Linux].

L4S is designed for incremental deployment. It is possible to deploy the L4S service at a bottleneck link alongside the existing best efforts service [DualPI2Linux] so that unmodified applications can start using it as soon as the sender's stack is updated. Access networks are typically designed with one link as the bottleneck for each site (which might be a home, small enterprise or mobile device), so deployment at either or both ends of this link should give nearly all the benefit in the respective direction. With some transport protocols, namely TCP and SCTP, the sender has to check for suitably updated receiver feedback, whereas with more recent transport protocols such as QUIC and DCCP, all receivers have always been suitable.

This document presents the L4S architecture, by describing and justifying the component parts and how they interact to provide the scalable, low latency, low loss Internet service. It also details the approach to incremental deployment, as briefly summarized above.

1.1. Document Roadmap

This document describes the L4S architecture in three passes. First this brief overview gives the very high level idea and states the main components with minimal rationale. This is only intended to give some context for the terminology definitions that follow in Section 3, and to explain the structure of the rest of the document. Then Section 4 goes into more detail on each component with some rationale, but still mostly stating what the architecture is, rather than why. Finally Section 5 justifies why each element of the solution was chosen (Section 5.1) and why these choices were different from other solutions (Section 5.2).

Having described the architecture, Section 6 clarifies its applicability; that is, the applications and use-cases that motivated the design, the challenges applying the architecture to various link technologies, and various incremental deployment models: including the two main deployment topologies, different sequences for incremental deployment and various interactions with pre-existing approaches. The document ends with the usual tail pieces, including extensive discussion of traffic policing and other security considerations Section 8.

2. L4S Architecture Overview

Below we outline the three main components to the L4S architecture; 1) the scalable congestion control on the sending host; 2) the AQM at the network bottleneck; and 3) the protocol between them.

But first, the main point to grasp is that low latency is not provided by the network - low latency results from the careful behaviour of the scalable congestion controllers used by L4S senders. The network does have a role - primarily to isolate the low latency of the carefully behaving L4S traffic from the higher queuing delay needed by traffic with pre-existing Classic behaviour. The network also alters the way it signals queue growth to the transport - It uses the Explicit Congestion Notification (ECN) protocol, but it signals the very start of queue growth - immediately without the smoothing delay typical of Classic AQMs. Because ECN support is essential for L4S, senders use the ECN field as the protocol to identify to the network which packets are L4S and which are Classic.

- 1) Host: Scalable congestion controls already exist. They solve the scaling problem with Classic congestion controls, such as Reno or Cubic. Because flow rate has scaled since TCP congestion control was first designed in 1988, assuming the flow lasts long enough, it now takes hundreds of round trips (and growing) to recover after a congestion signal (whether a loss or an ECN mark) as shown in the examples in Section 5.1 and [RFC3649]. Therefore control of queuing and utilization becomes very slack, and the slightest disturbances (e.g. from new flows starting) prevent a high rate from being attained.

With a scalable congestion control, the average time from one congestion signal to the next (the recovery time) remains invariant as the flow rate scales, all other factors being equal. This maintains the same degree of control over queueing and utilization whatever the flow rate, as well as ensuring that high throughput is more robust to disturbances. The scalable control used most widely (in controlled environments) is Data Center TCP (DCTCP [RFC8257]), which has been implemented and deployed in Windows Server Editions (since 2012), in Linux and in FreeBSD. Although DCTCP as-is functions well over wide-area round trip times, most implementations lack certain safety features that would be necessary for use outside controlled environments like data centres (see Section 6.4.3). So scalable congestion control needs to be implemented in TCP and other transport protocols (QUIC, SCTP, RTP/RTCP, RMCAT, etc.). Indeed, between the present document being drafted and published, the following scalable congestion controls were implemented: TCP Prague [PragueLinux], QUIC Prague, an L4S variant of the RMCAT SCReAM controller [SCReAM] and the L4S ECN part of BBRv2 [BBRv2] intended for TCP and QUIC transports.

- 2) Network: L4S traffic needs to be isolated from the queuing

latency of Classic traffic. One queue per application flow (FQ) is one way to achieve this, e.g. FQ-CoDel [RFC8290]. However, just two queues is sufficient and does not require inspection of transport layer headers in the network, which is not always possible (see Section 5.2). With just two queues, it might seem impossible to know how much capacity to schedule for each queue without inspecting how many flows at any one time are using each. And it would be undesirable to arbitrarily divide access network capacity into two partitions. The Dual Queue Coupled AQM was developed as a minimal complexity solution to this problem. It acts like a 'semi-permeable' membrane that partitions latency but not bandwidth. As such, the two queues are for transition from Classic to L4S behaviour, not bandwidth prioritization.

Section 4 gives a high level explanation of how the per-flow-queue (FQ) and DualQ variants of L4S work, and [I-D.ietf-tsvwg-aqm-dualq-coupled] gives a full explanation of the DualQ Coupled AQM framework. A specific marking algorithm is not mandated for L4S AQMs. Appendices of [I-D.ietf-tsvwg-aqm-dualq-coupled] give non-normative examples that have been implemented and evaluated, and give recommended default parameter settings. It is expected that L4S experiments will improve knowledge of parameter settings and whether the set of marking algorithms needs to be limited.

- 3) Protocol: A host needs to distinguish L4S and Classic packets with an identifier so that the network can classify them into their separate treatments. The L4S identifier spec. [I-D.ietf-tsvwg-ecn-l4s-id] concludes that all alternatives involve compromises, but the ECT(1) and CE codepoints of the ECN field represent a workable solution. As already explained, the network also uses ECN to immediately signal the very start of queue growth to the transport.

3. Terminology

Note: The following definitions are copied from the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id] for convenience. If there are accidental differences, those in [I-D.ietf-tsvwg-ecn-l4s-id] take precedence.

Classic Congestion Control: A congestion control behaviour that can co-exist with standard Reno [RFC5681] without causing significantly negative impact on its flow rate [RFC5033]. The scaling problem with Classic congestion control is explained, with examples, in Section 5.1 and in [RFC3649].

Scalable Congestion Control: A congestion control where the average

time from one congestion signal to the next (the recovery time) remains invariant as the flow rate scales, all other factors being equal. For instance, DCTCP averages 2 congestion signals per round-trip whatever the flow rate, as do other recently developed scalable congestion controls, e.g. Relentless TCP [Mathis09], TCP Prague [I-D.briscoe-iccrp-prague-congestion-control], [PragueLinux], BBRv2 [BBRv2], [I-D.cardwell-iccrp-bbr-congestion-control] and the L4S variant of SCReAM for real-time media [SCReAM], [RFC8298]). See Section 4.3 of [I-D.ietf-tsvwg-ecn-l4s-id] for more explanation.

Classic service: The Classic service is intended for all the congestion control behaviours that co-exist with Reno [RFC5681] (e.g. Reno itself, Cubic [RFC8312], Compound [I-D.sridharan-tcpm-ctcp], TFRC [RFC5348]). The term 'Classic queue' means a queue providing the Classic service.

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

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

Both Classic and L4S services can cope with a proportion of unresponsive or less-responsive traffic as well, but in the L4S case its rate has to be smooth enough or low enough not build a queue (e.g. DNS, VoIP, game sync datagrams, etc).

Reno-friendly: The subset of Classic traffic that is friendly to the standard Reno congestion control defined for TCP in [RFC5681]. The TFRC spec. [RFC5348] indirectly implies that 'friendly' is defined as "generally within a factor of two of the sending rate of a TCP flow under the same conditions". Reno-friendly is used here in place of 'TCP-friendly', given the latter has become imprecise, because the TCP protocol is now used with so many different congestion control behaviours, and Reno is used in non-TCP transports such as QUIC [RFC9000].

Classic ECN: The original Explicit Congestion Notification (ECN)

protocol [RFC3168], which requires ECN signals to be treated as equivalent to drops, both when generated in the network and when responded to by the sender.

L4S uses the ECN field as an identifier [I-D.ietf-tsvwg-ecn-l4s-id] with the names for the four codepoints of the 2-bit IP-ECN field unchanged from those defined in the ECN spec [RFC3168]: Not ECT, ECT(0), ECT(1) and CE, where ECT stands for ECN-Capable Transport and CE stands for Congestion Experienced. A packet marked with the CE codepoint is termed 'ECN-marked' or sometimes just 'marked' where the context makes ECN obvious.

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

4. L4S Architecture Components

The L4S architecture is composed of the elements in the following three subsections.

4.1. Protocol Mechanisms

The L4S architecture involves: a) unassignment of an identifier; b) reassignment of the same identifier; and c) optional further identifiers:

- a. An essential aspect of a scalable congestion control is the use of explicit congestion signals. 'Classic' ECN [RFC3168] requires an ECN signal to be treated as equivalent to drop, both when it is generated in the network and when it is responded to by hosts. L4S needs networks and hosts to support a more fine-grained meaning for each ECN signal that is less severe than a drop, so that the L4S signals:

- * can be much more frequent;
- * can be signalled immediately, without the significant delay required to smooth out fluctuations in the queue.

To enable L4S, the standards track Classic ECN spec. [RFC3168] has had to be updated to allow L4S packets to depart from the 'equivalent to drop' constraint. [RFC8311] is a standards track update to relax specific requirements in RFC 3168 (and certain other standards track RFCs), which clears the way for the experimental changes proposed for L4S. [RFC8311] also reclassifies the original experimental assignment of the ECT(1) codepoint as an ECN nonce [RFC3540] as historic.

- b. [I-D.ietf-tsvwg-ecn-l4s-id] specifies that ECT(1) is used as the identifier to classify L4S packets into a separate treatment from Classic packets. This satisfies the requirement for identifying an alternative ECN treatment in [RFC4774].

The CE codepoint is used to indicate Congestion Experienced by both L4S and Classic treatments. This raises the concern that a Classic AQM earlier on the path might have marked some ECT(0) packets as CE. Then these packets will be erroneously classified into the L4S queue. Appendix B of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id] explains why five unlikely eventualities all have to coincide for this to have any detrimental effect, which even then would only involve a vanishingly small likelihood of a spurious retransmission.

- c. A network operator might wish to include certain unresponsive, non-L4S traffic in the L4S queue if it is deemed to be smoothly enough paced and low enough rate not to build a queue. For instance, VoIP, low rate datagrams to sync online games, relatively low rate application-limited traffic, DNS, LDAP, etc. This traffic would need to be tagged with specific identifiers, e.g. a low latency Diffserv Codepoint such as Expedited Forwarding (EF [RFC3246]), Non-Queue-Building (NQB [I-D.ietf-tsvwg-nqb]), or operator-specific identifiers.

4.2. Network Components

The L4S architecture aims to provide low latency without the need for per-flow operations in network components. Nonetheless, the architecture does not preclude per-flow solutions. The following bullets describe the known arrangements: a) the DualQ Coupled AQM with an L4S AQM in one queue coupled from a Classic AQM in the other; b) Per-Flow Queues with an instance of a Classic and an L4S AQM in each queue; c) Dual queues with per-flow AQMs, but no per-flow queues:

- a. The Dual Queue Coupled AQM (illustrated in Figure 1) achieves the 'semi-permeable' membrane property mentioned earlier as follows:

- * Latency isolation: Two separate queues are used to isolate L4S queuing delay from the larger queue that Classic traffic needs to maintain full utilization.
- * Bandwidth pooling: The two queues act as if they are a single pool of bandwidth in which flows of either type get roughly equal throughput without the scheduler needing to identify any flows. This is achieved by having an AQM in each queue, but the Classic AQM provides a congestion signal to both queues in a manner that ensures a consistent response from the two classes of congestion control. Specifically, the Classic AQM generates a drop/mark probability based on congestion in its own queue, which it uses both to drop/mark packets in its own queue and to affect the marking probability in the L4S queue. The strength of the coupling of the congestion signalling between the two queues is enough to make the L4S flows slow down to leave the right amount of capacity for the Classic flows (as they would if they were the same type of traffic sharing the same queue).

Then the scheduler can serve the L4S queue with priority (denoted by the '1' on the higher priority input), because the L4S traffic isn't offering up enough traffic to use all the priority that it is given. Therefore:

- * for latency isolation on short time-scales (sub-round-trip) the prioritization of the L4S queue protects its low latency by allowing bursts to dissipate quickly;
- * but for bandwidth pooling on longer time-scales (round-trip and longer) the Classic queue creates an equal and opposite pressure against the L4S traffic to ensure that neither has priority when it comes to bandwidth - the tension between prioritizing L4S and coupling the marking from the Classic AQM results in approximate per-flow fairness.

To protect against unresponsive traffic taking advantage of the prioritization of the L4S queue and starving the Classic queue, it is advisable for the priority to be conditional, not strict (see Appendix A of the DualQ spec [I-D.ietf-tsvwg-aqm-dualq-coupled]).

When there is no Classic traffic, the L4S queue's own AQM comes into play. It starts congestion marking with a very shallow queue, so L4S traffic maintains very low queuing delay.

If either queue becomes persistently overloaded, drop of ECN-capable packets is introduced, as recommended in Section 7 of the ECN spec [RFC3168] and Section 4.2.1 of the AQM recommendations [RFC7567]. Then both queues introduce the same level of drop (not shown in the figure).

The Dual Queue Coupled AQM has been specified as generically as possible [I-D.ietf-tsvwg-aqm-dualq-coupled] without specifying the particular AQMs to use in the two queues so that designers are free to implement diverse ideas. Informational appendices in that draft give pseudocode examples of two different specific AQM approaches: one called DualPI2 (pronounced Dual PI Squared) [DualPI2Linux] that uses the PI2 variant of PIE, and a zero-config variant of RED called Curvy RED. A DualQ Coupled AQM based on PIE has also been specified and implemented for Low Latency DOCSIS [DOCSIS3.1].

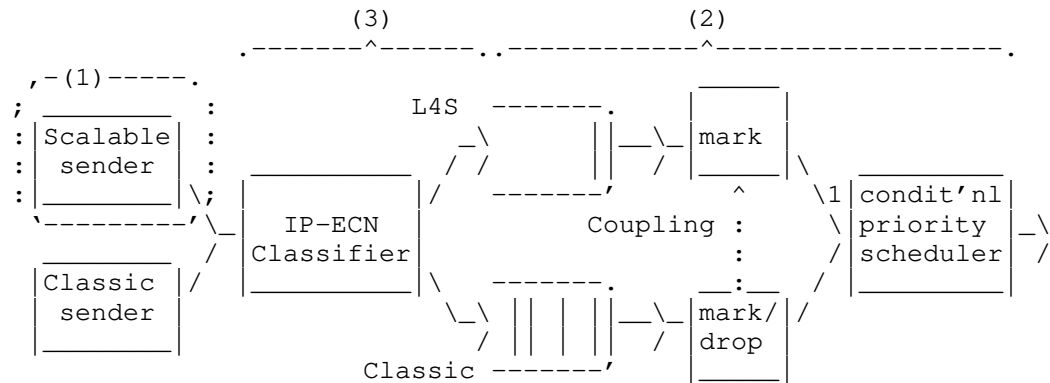


Figure 1: Components of an L4S DualQ Coupled AQM Solution: 1) Scalable Sending Host; 2) Isolation in separate network queues; and 3) Packet Identification Protocol

- b. **Per-Flow Queues and AQMs:** A scheduler with per-flow queues such as FQ-CoDel or FQ-PIE can be used for L4S. For instance within each queue of an FQ-CoDel system, as well as a CoDel AQM, there is typically also the option of ECN marking at an immediate (unsmoothed) shallow threshold to support use in data centres (see Sec.5.2.7 of the FQ-CoDel spec [RFC8290]). In Linux, this has been modified so that the shallow threshold can be solely applied to ECT(1) packets [FQ_CoDel_Thresh]. Then if there is a flow of non-ECN or ECT(0) packets in the per-flow-queue, the Classic AQM (e.g. CoDel) is applied; while if there is a flow of ECT(1) packets in the queue, the shallower (typically sub-millisecond) threshold is applied. In addition, ECT(0) and not-ECT packets could potentially be classified into a separate flow-queue from ECT(1) and CE packets to avoid them mixing if they share a common flow-identifier (e.g. in a VPN).
- c. **Dual-queues, but per-flow AQMs:** It should also be possible to use dual queues for isolation, but with per-flow marking to control flow-rates (instead of the coupled per-queue marking of the Dual Queue Coupled AQM). One of the two queues would be for isolating L4S packets, which would be classified by the ECN codepoint. Flow rates could be controlled by flow-specific marking. The policy goal of the marking could be to differentiate flow rates (e.g. [Nadas20], which requires additional signalling of a per-flow 'value'), or to equalize flow-rates (perhaps in a similar way to Approx Fair CoDel [AFCD], [I-D.morton-tsvwg-codel-approx-fair], but with two queues not one).

Note that whenever the term 'DualQ' is used loosely without saying whether marking is per-queue or per-flow, it means a dual queue AQM with per-queue marking.

4.3. Host Mechanisms

The L4S architecture includes two main mechanisms in the end host that we enumerate next:

- a. **Scalable Congestion Control at the sender:** Section 2 defines a scalable congestion control as one where the average time from one congestion signal to the next (the recovery time) remains invariant as the flow rate scales, all other factors being equal. Data Center TCP is the most widely used example. It has been documented as an informational record of the protocol currently in use in controlled environments [RFC8257]. A draft list of safety and performance improvements for a scalable congestion control to be usable on the public Internet has been drawn up (the so-called 'Prague L4S requirements' in Appendix A of

[I-D.ietf-tsvwg-ecn-l4s-id]). The subset that involve risk of harm to others have been captured as normative requirements in Section 4 of [I-D.ietf-tsvwg-ecn-l4s-id]. TCP Prague [I-D.briscoe-iccrp-prague-congestion-control] has been implemented in Linux as a reference implementation to address these requirements [PragueLinux].

Transport protocols other than TCP use various congestion controls that are designed to be friendly with Reno. Before they can use the L4S service, they will need to be updated to implement a scalable congestion response, which they will have to indicate by using the ECT(1) codepoint. Scalable variants are under consideration for more recent transport protocols, e.g. QUIC, and the L4S ECN part of BBRv2 [BBRv2], [I-D.cardwell-iccrp-bbr-congestion-control] is a scalable congestion control intended for the TCP and QUIC transports, amongst others. Also an L4S variant of the RMCAT SCReAM controller [RFC8298] has been implemented [SCReAM] for media transported over RTP.

Section 4.3 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id] defines scalable congestion control in more detail, and specifies that requirements that an L4S scalable congestion control has to comply with.

- b. The ECN feedback in some transport protocols is already sufficiently fine-grained for L4S (specifically DCCP [RFC4340] and QUIC [RFC9000]). But others either require update or are in the process of being updated:
 - * For the case of TCP, the feedback protocol for ECN embeds the assumption from Classic ECN [RFC3168] that an ECN mark is equivalent to a drop, making it unusable for a scalable TCP. Therefore, the implementation of TCP receivers will have to be upgraded [RFC7560]. Work to standardize and implement more accurate ECN feedback for TCP (AccECN) is in progress [I-D.ietf-tcpm-accurate-ecn], [PragueLinux].
 - * ECN feedback is only roughly sketched in an appendix of the SCTP specification [RFC4960]. A fuller specification has been proposed in a long-expired draft [I-D.stewart-tsvwg-sctpecn], which would need to be implemented and deployed before SCTCP could support L4S.
 - * For RTP, sufficient ECN feedback was defined in [RFC6679], but [RFC8888] defines the latest standards track improvements.

5. Rationale

5.1. Why These Primary Components?

Explicit congestion signalling (protocol): Explicit congestion signalling is a key part of the L4S approach. In contrast, use of drop as a congestion signal creates a tension because drop is both an impairment (less would be better) and a useful signal (more would be better):

- * Explicit congestion signals can be used many times per round trip, to keep tight control, without any impairment. Under heavy load, even more explicit signals can be applied so the queue can be kept short whatever the load. In contrast, Classic AQMs have to introduce very high packet drop at high load to keep the queue short. By using ECN, an L4S congestion control's sawtooth reduction can be smaller and therefore return to the operating point more often, without worrying that more sawteeth will cause more signals. The consequent smaller amplitude sawteeth fit between an empty queue and a very shallow marking threshold (~1 ms in the public Internet), so queue delay variation can be very low, without risk of under-utilization.
- * Explicit congestion signals can be emitted immediately to track fluctuations of the queue. L4S shifts smoothing from the network to the host. The network doesn't know the round trip times of any of the flows. So if the network is responsible for smoothing (as in the Classic approach), it has to assume a worst case RTT, otherwise long RTT flows would become unstable. This delays Classic congestion signals by 100-200 ms. In contrast, each host knows its own round trip time. So, in the L4S approach, the host can smooth each flow over its own RTT, introducing no more soothing delay than strictly necessary (usually only a few milliseconds). A host can also choose not to introduce any smoothing delay if appropriate, e.g. during flow start-up.

Neither of the above are feasible if explicit congestion signalling has to be considered 'equivalent to drop' (as was required with Classic ECN [RFC3168]), because drop is an impairment as well as a signal. So drop cannot be excessively frequent, and drop cannot be immediate, otherwise too many drops would turn out to have been due to only a transient fluctuation in the queue that would not have warranted dropping a packet in hindsight. Therefore, in an L4S AQM, the L4S queue uses a new L4S variant of ECN that is not equivalent to drop (see section 5.2 of

the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id]), while the Classic queue uses either Classic ECN [RFC3168] or drop, which are equivalent to each other.

Before Classic ECN was standardized, there were various proposals to give an ECN mark a different meaning from drop. However, there was no particular reason to agree on any one of the alternative meanings, so 'equivalent to drop' was the only compromise that could be reached. RFC 3168 contains a statement that:

"An environment where all end nodes were ECN-Capable could allow new criteria to be developed for setting the CE codepoint, and new congestion control mechanisms for end-node reaction to CE packets. However, this is a research issue, and as such is not addressed in this document."

Latency isolation (network): L4S congestion controls keep queue delay low whereas Classic congestion controls need a queue of the order of the RTT to avoid under-utilization. One queue cannot have two lengths, therefore L4S traffic needs to be isolated in a separate queue (e.g. DualQ) or queues (e.g. FQ).

Coupled congestion notification: Coupling the congestion notification between two queues as in the DualQ Coupled AQM is not necessarily essential, but it is a simple way to allow senders to determine their rate, packet by packet, rather than be overridden by a network scheduler. An alternative is for a network scheduler to control the rate of each application flow (see discussion in Section 5.2).

L4S packet identifier (protocol): Once there are at least two treatments in the network, hosts need an identifier at the IP layer to distinguish which treatment they intend to use.

Scalable congestion notification: A scalable congestion control in the host keeps the signalling frequency from the network high whatever the flow rate, so that queue delay variations can be small when conditions are stable, and rate can track variations in available capacity as rapidly as possible otherwise.

Low loss: Latency is not the only concern of L4S. The 'Low Loss' part of the name denotes that L4S generally achieves zero congestion loss due to its use of ECN. Otherwise, loss would itself cause delay, particularly for short flows, due to retransmission delay [RFC2884].

Scalable throughput: The "Scalable throughput" part of the name

denotes that the per-flow throughput of scalable congestion controls should scale indefinitely, avoiding the imminent scaling problems with Reno-friendly congestion control algorithms [RFC3649]. It was known when TCP congestion avoidance was first developed in 1988 that it would not scale to high bandwidth-delay products (see footnote 6 in [TCP-CA]). Today, regular broadband flow rates over WAN distances are already beyond the scaling range of Classic Reno congestion control. So 'less unscalable' Cubic [RFC8312] and Compound [I-D.sridharan-tcpm-ctcp] variants of TCP have been successfully deployed. However, these are now approaching their scaling limits.

For instance, we will consider a scenario with a maximum RTT of 30 ms at the peak of each sawtooth. As Reno packet rate scales 8x from 1,250 to 10,000 packet/s (from 15 to 120 Mb/s with 1500 B packets), the time to recover from a congestion event rises proportionately by 8x as well, from 422 ms to 3.38 s. It is clearly problematic for a congestion control to take multiple seconds to recover from each congestion event. Cubic [RFC8312] was developed to be less unscalable, but it is approaching its scaling limit; with the same max RTT of 30 ms, at 120 Mb/s Cubic is still fully in its Reno-friendly mode, so it takes about 4.3 s to recover. However, once the flow rate scales by 8x again to 960 Mb/s it enters true Cubic mode, with a recovery time of 12.2 s. From then on, each further scaling by 8x doubles Cubic's recovery time (because the cube root of 8 is 2), e.g. at 7.68 Gb/s the recovery time is 24.3 s. In contrast a scalable congestion control like DCTCP or TCP Prague induces 2 congestion signals per round trip on average, which remains invariant for any flow rate, keeping dynamic control very tight.

For a feel of where the global average lone-flow download sits on this scale at the time of writing (2021), according to [BDPdata] globally averaged fixed access capacity was 103 Mb/s in 2020 and averaged base RTT to a CDN was 25-34ms in 2019. Averaging of per-country data was weighted by Internet user population (data collected globally is necessarily of variable quality, but the paper does double-check that the outcome compares well against a second source). So a lone CUBIC flow would at best take about 200 round trips (5 s) to recover from each of its sawtooth reductions, if the flow even lasted that long. This is described as 'at best' because it assume everyone uses an AQM, whereas in reality most users still have a (probably bloated) tail-drop buffer. In the tail-drop case, likely average recovery time would be at least 4x 5 s, if not more, because RTT under load would be at least double that of an AQM, and recovery time depends on the square of RTT.

Although work on scaling congestion controls tends to start with TCP as the transport, the above is not intended to exclude other transports (e.g. SCTP, QUIC) or less elastic algorithms (e.g. RMCAT), which all tend to adopt the same or similar developments.

5.2. What L4S adds to Existing Approaches

All the following approaches address some part of the same problem space as L4S. In each case, it is shown that L4S complements them or improves on them, rather than being a mutually exclusive alternative:

Diffserv: Diffserv addresses the problem of bandwidth apportionment for important traffic as well as queuing latency for delay-sensitive traffic. Of these, L4S solely addresses the problem of queuing latency. Diffserv will still be necessary where important traffic requires priority (e.g. for commercial reasons, or for protection of critical infrastructure traffic) - see [I-D.briscoe-tsvwg-l4s-diffserv]. Nonetheless, the L4S approach can provide low latency for all traffic within each Diffserv class (including the case where there is only the one default Diffserv class).

Also, Diffserv can only provide a latency benefit if a small subset of the traffic on a bottleneck link requests low latency. As already explained, it has no effect when all the applications in use at one time at a single site (home, small business or mobile device) require low latency. In contrast, because L4S works for all traffic, it needs none of the management baggage (traffic policing, traffic contracts) associated with favouring some packets over others. This lack of management baggage ought to give L4S a better chance of end-to-end deployment.

In particular, because networks tend not to trust end systems to identify which packets should be favoured over others, where networks assign packets to Diffserv classes they tend to use packet inspection of application flow identifiers or deeper inspection of application signatures. Thus, nowadays, Diffserv doesn't always sit well with encryption of the layers above IP [RFC8404]. So users have to choose between privacy and QoS.

As with Diffserv, the L4S identifier is in the IP header. But, in contrast to Diffserv, the L4S identifier does not convey a want or a need for a certain level of quality. Rather, it promises a certain behaviour (scalable congestion response), which networks can objectively verify if they need to. This is because low delay depends on collective host behaviour, whereas bandwidth priority depends on network behaviour.

State-of-the-art AQMs: AQMs such as PIE and FQ-CoDel give a significant reduction in queuing delay relative to no AQM at all. L4S is intended to complement these AQMs, and should not distract from the need to deploy them as widely as possible. Nonetheless, AQMs alone cannot reduce queuing delay too far without significantly reducing link utilization, because the root cause of the problem is on the host - where Classic congestion controls use large saw-toothed rate variations. The L4S approach resolves this tension between delay and utilization by enabling hosts to minimize the amplitude of their sawteeth. A single-queue Classic AQM is not sufficient to allow hosts to use small sawteeth for two reasons: i) smaller sawteeth would not get lower delay in an AQM designed for larger amplitude Classic sawteeth, because a queue can only have one length at a time; and ii) much smaller sawteeth implies much more frequent sawteeth, so L4S flows would drive a Classic AQM into a high level of ECN-marking, which would appear as heavy congestion to Classic flows, which in turn would greatly reduce their rate as a result (see Section 6.4.4).

Per-flow queuing or marking: Similarly, per-flow approaches such as FQ-CoDel or Approx Fair CoDel [AFCD] are not incompatible with the L4S approach. However, per-flow queuing alone is not enough - it only isolates the queuing of one flow from others; not from itself. Per-flow implementations need to have support for scalable congestion control added, which has already been done for FQ-CoDel in Linux (see Sec.5.2.7 of [RFC8290] and [FQ_CoDel_Thresh]). Without this simple modification, per-flow AQMs like FQ-CoDel would still not be able to support applications that need both very low delay and high bandwidth, e.g. video-based control of remote procedures, or interactive cloud-based video (see Note 1 below).

Although per-flow techniques are not incompatible with L4S, it is important to have the DualQ alternative. This is because handling end-to-end (layer 4) flows in the network (layer 3 or 2) precludes some important end-to-end functions. For instance:

- a. Per-flow forms of L4S like FQ-CoDel are incompatible with full end-to-end encryption of transport layer identifiers for privacy and confidentiality (e.g. IPSec or encrypted VPN tunnels, as opposed to TLS over UDP), because they require packet inspection to access the end-to-end transport flow identifiers.

In contrast, the DualQ form of L4S requires no deeper inspection than the IP layer. So, as long as operators take the DualQ approach, their users can have both very low queuing delay and full end-to-end encryption [RFC8404].

- b. With per-flow forms of L4S, the network takes over control of the relative rates of each application flow. Some see it as an advantage that the network will prevent some flows running faster than others. Others consider it an inherent part of the Internet's appeal that applications can control their rate while taking account of the needs of others via congestion signals. They maintain that this has allowed applications with interesting rate behaviours to evolve, for instance, variable bit-rate video that varies around an equal share rather than being forced to remain equal at every instant, or e2e scavenger behaviours [RFC6817] that use less than an equal share of capacity [LEDBAT_AQM].

The L4S architecture does not require the IETF to commit to one approach over the other, because it supports both, so that the 'market' can decide. Nonetheless, in the spirit of 'Do one thing and do it well' [McIlroy78], the DualQ option provides low delay without prejudging the issue of flow-rate control. Then, flow rate policing can be added separately if desired. This allows application control up to a point, but the network can still choose to set the point at which it intervenes to prevent one flow completely starving another.

Note:

1. It might seem that self-inflicted queuing delay within a per-flow queue should not be counted, because if the delay wasn't in the network it would just shift to the sender. However, modern adaptive applications, e.g. HTTP/2 [RFC7540] or some interactive media applications (see Section 6.1), can keep low latency objects at the front of their local send queue by shuffling priorities of other objects dependent on the progress of other transfers (for example see [lowat]). They cannot shuffle objects once they have released them into the network.

Alternative Back-off ECN (ABE): Here again, L4S is not an alternative to ABE but a complement that introduces much lower queuing delay. ABE [RFC8511] alters the host behaviour in response to ECN marking to utilize a link better and give ECN flows faster throughput. It uses ECT(0) and assumes the network still treats ECN and drop the same. Therefore ABE exploits any lower queuing delay that AQMs can provide. But as explained above, AQMs still cannot reduce queuing delay too far without losing link utilization (to allow for other, non-ABE, flows).

BBR: Bottleneck Bandwidth and Round-trip propagation time

(BBR [I-D.cardwell-iccr-g-bbr-congestion-control]) controls queuing delay end-to-end without needing any special logic in the network, such as an AQM. So it works pretty-much on any path. BBR keeps queuing delay reasonably low, but perhaps not quite as low as with state-of-the-art AQMs such as PIE or FQ-CoDel, and certainly nowhere near as low as with L4S. Queuing delay is also not consistently low, due to BBR's regular bandwidth probing spikes and its aggressive flow start-up phase.

L4S complements BBR. Indeed BBRv2 can use L4S ECN where available and a scalable L4S congestion control behaviour in response to any ECN signalling from the path [BBRv2]. The L4S ECN signal complements the delay based congestion control aspects of BBR with an explicit indication that hosts can use, both to converge on a fair rate and to keep below a shallow queue target set by the network. Without L4S ECN, both these aspects need to be assumed or estimated.

6. Applicability

6.1. Applications

A transport layer that solves the current latency issues will provide new service, product and application opportunities.

With the L4S approach, the following existing applications also experience significantly better quality of experience under load:

- * Gaming, including cloud based gaming;
- * VoIP;
- * Video conferencing;
- * Web browsing;
- * (Adaptive) video streaming;
- * Instant messaging.

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

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

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

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

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

Other low latency high bandwidth applications such as:

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

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

6.2. Use Cases

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

- * Where the bottleneck is one of various types of access network:
e.g. DSL, Passive Optical Networks (PON), DOCSIS cable, mobile, satellite (see Section 6.3 for some technology-specific details)
- * Private networks of heterogeneous data centres, where there is no single administrator that can arrange for all the simultaneous changes to senders, receivers and network needed to deploy DCTCP:
 - a set of private data centres interconnected over a wide area with separate administrations, but within the same company
 - a set of data centres operated by separate companies interconnected by a community of interest network (e.g. for the finance sector)
 - multi-tenant (cloud) data centres where tenants choose their operating system stack (Infrastructure as a Service - IaaS)
- * Different types of transport (or application) congestion control:
 - elastic (TCP/SCTP);
 - real-time (RTP, RMCAT);
 - query (DNS/LDAP).
- * Where low delay quality of service is required, but without inspecting or intervening above the IP layer [RFC8404]:
 - mobile and other networks have tended to inspect higher layers in order to guess application QoS requirements. However, with growing demand for support of privacy and encryption, L4S offers an alternative. There is no need to select which traffic to favour for queuing, when L4S can give favourable queuing to all traffic.
- * If queuing delay is minimized, applications with a fixed delay budget can communicate over longer distances, or via a longer chain of service functions [RFC7665] or onion routers.
- * If delay jitter is minimized, it is possible to reduce the dejitter buffers on the receive end of video streaming, which should improve the interactive experience

6.3. Applicability with Specific Link Technologies

Certain link technologies aggregate data from multiple packets into bursts, and buffer incoming packets while building each burst. WiFi, PON and cable all involve such packet aggregation, whereas fixed Ethernet and DSL do not. No sender, whether L4S or not, can do anything to reduce the buffering needed for packet aggregation. So an AQM should not count this buffering as part of the queue that it controls, given no amount of congestion signals will reduce it.

Certain link technologies also add buffering for other reasons, specifically:

- * Radio links (cellular, WiFi, satellite) that are distant from the source are particularly challenging. The radio link capacity can vary rapidly by orders of magnitude, so it is considered desirable to hold a standing queue that can utilize sudden increases of capacity;
- * Cellular networks are further complicated by a perceived need to buffer in order to make hand-overs imperceptible;

L4S cannot remove the need for all these different forms of buffering. However, by removing 'the longest pole in the tent' (buffering for the large sawteeth of Classic congestion controls), L4S exposes all these 'shorter poles' to greater scrutiny.

Until now, the buffering needed for these additional reasons tended to be over-specified - with the excuse that none were 'the longest pole in the tent'. But having removed the 'longest pole', it becomes worthwhile to minimize them, for instance reducing packet aggregation burst sizes and MAC scheduling intervals.

6.4. Deployment Considerations

L4S AQMs, whether DualQ [I-D.ietf-tsvwg-aqm-dualq-coupled] or FQ, e.g. [RFC8290] are, in themselves, an incremental deployment mechanism for L4S - so that L4S traffic can coexist with existing Classic (Reno-friendly) traffic. Section 6.4.1 explains why only deploying an L4S AQM in one node at each end of the access link will realize nearly all the benefit of L4S.

L4S involves both end systems and the network, so Section 6.4.2 suggests some typical sequences to deploy each part, and why there will be an immediate and significant benefit after deploying just one part.

Section 6.4.3 and Section 6.4.4 describe the converse incremental deployment case where there is no L4S AQM at the network bottleneck, so any L4S flow traversing this bottleneck has to take care in case it is competing with Classic traffic.

6.4.1. Deployment Topology

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

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

Therefore, the full benefit of the L4S service should be available in the downstream direction when an L4S AQM is deployed at the ingress to this bottleneck link. And similarly, the full upstream service will be available once an L4S AQM is deployed at the ingress into the upstream link. (Of course, multi-homed sites would only see the full benefit once all their access links were covered.)

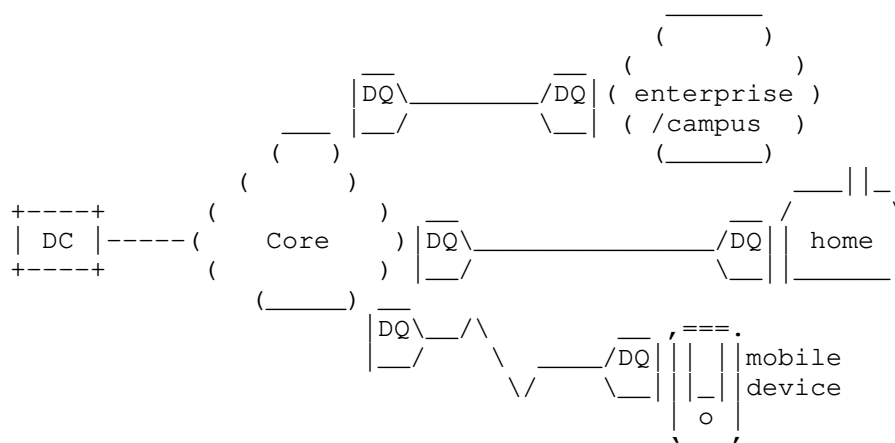


Figure 2: Likely location of DualQ (DQ) Deployments in common access topologies

Deployment in mesh topologies depends on how overbooked the core is. If the core is non-blocking, or at least generously provisioned so that the edges are nearly always the bottlenecks, it would only be necessary to deploy an L4S AQM at the edge bottlenecks. For example, some data-centre networks are designed with the bottleneck in the hypervisor or host NICs, while others bottleneck at the top-of-rack switch (both the output ports facing hosts and those facing the core).

An L4S AQM would often next be needed where the WiFi links in a home sometimes become the bottleneck. And an L4S AQM would eventually also need to be deployed at any other persistent bottlenecks such as network interconnections, e.g. some public Internet exchange points and the ingress and egress to WAN links interconnecting data-centres.

6.4.2. Deployment Sequences

For any one L4S flow to provide benefit, it requires three (or sometimes two) parts to have been deployed: i) the congestion control at the sender; ii) the AQM at the bottleneck; and iii) older transports (namely TCP) need upgraded receiver feedback too. This was the same deployment problem that ECN faced [RFC8170] so we have learned from that experience.

Firstly, L4S deployment exploits the fact that DCTCP already exists on many Internet hosts (Windows, FreeBSD and Linux); both servers and clients. Therefore, an L4S AQM can be deployed at a network bottleneck to immediately give a working deployment of all the L4S parts for testing, as long as the ECT(0) codepoint is switched to ECT(1). DCTCP needs some safety concerns to be fixed for general use over the public Internet (see Section 4.3 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id]), but DCTCP is not on by default, so these issues can be managed within controlled deployments or controlled trials.

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

Thirdly, the L4S identifier is defined so that initially network operators can enable L4S exclusively for certain customers or certain applications. But this is carefully defined so that it does not compromise future evolution towards L4S as an Internet-wide service. This is because the L4S identifier is defined not only as the end-to-

end ECN field, but it can also optionally be combined with any other packet header or some status of a customer or their access link (see section 5.4 of [I-D.ietf-tsvwg-ecn-l4s-id]). Operators could do this anyway, even if it were not blessed by the IETF. However, it is best for the IETF to specify that, if they use their own local identifier, it must be in combination with the IETF's identifier. Then, if an operator has opted for an exclusive local-use approach, later they only have to remove this extra rule to make the service work Internet-wide - it will already traverse middleboxes, peerings, etc.

	Servers or proxies	Access link	Clients
0	DCTCP (existing)		DCTCP (existing)
1	Add L4S AQM downstream WORKS DOWNSTREAM FOR CONTROLLED DEPLOYMENTS/TRIALS		
2	Upgrade DCTCP to TCP Prague	FULLY WORKS DOWNSTREAM	Replace DCTCP feedb'k with AccECN
3		Add L4S AQM upstream	Upgrade DCTCP to TCP Prague
	FULLY WORKS UPSTREAM AND DOWNSTREAM		

Figure 3: Example L4S Deployment Sequence

Figure 3 illustrates some example sequences in which the parts of L4S might be deployed. It consists of the following stages:

1. Here, the immediate benefit of a single AQM deployment can be seen, but limited to a controlled trial or controlled deployment. In this example downstream deployment is first, but in other scenarios the upstream might be deployed first. If no AQM at all was previously deployed for the downstream access, an L4S AQM greatly improves the Classic service (as well as adding the L4S service). If an AQM was already deployed, the Classic service will be unchanged (and L4S will add an improvement on top).
2. In this stage, the name 'TCP Prague' [I-D.briscoe-iccrp-prague-congestion-control] is used to represent a variant of DCTCP that is designed to be used in a production Internet environment (assuming it complies with the requirements in Section 4 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id]). If the application is

primarily unidirectional, 'TCP Prague' at one end will provide all the benefit needed. For TCP transports, Accurate ECN feedback (AccECN) [I-D.ietf-tcpm-accurate-ecn] is needed at the other end, but it is a generic ECN feedback facility that is already planned to be deployed for other purposes, e.g. DCTCP, BBR. The two ends can be deployed in either order, because, in TCP, an L4S congestion control only enables itself if it has negotiated the use of AccECN feedback with the other end during the connection handshake. Thus, deployment of TCP Prague on a server enables L4S trials to move to a production service in one direction, wherever AccECN is deployed at the other end. This stage might be further motivated by the performance improvements of TCP Prague relative to DCTCP (see Appendix A.2 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id]).

Unlike TCP, from the outset, QUIC ECN feedback [RFC9000] has supported L4S. Therefore, if the transport is QUIC, one-ended deployment of a Prague congestion control at this stage is simple and sufficient.

3. This is a two-move stage to enable L4S upstream. An L4S AQM or TCP Prague can be deployed in either order as already explained. To motivate the first of two independent moves, the deferred benefit of enabling new services after the second move has to be worth it to cover the first mover's investment risk. As explained already, the potential for new interactive services provides this motivation. An L4S AQM also improves the upstream Classic service - significantly if no other AQM has already been deployed.

Note that other deployment sequences might occur. For instance: the upstream might be deployed first; a non-TCP protocol might be used end-to-end, e.g. QUIC, RTP; a body such as the 3GPP might require L4S to be implemented in 5G user equipment, or other random acts of kindness.

6.4.3. L4S Flow but Non-ECN Bottleneck

If L4S is enabled between two hosts, the L4S sender is required to coexist safely with Reno in response to any drop (see Section 4.3 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id]).

Unfortunately, as well as protecting Classic traffic, this rule degrades the L4S service whenever there is any loss, even if the cause is not persistent congestion at a bottleneck, e.g.:

- * congestion loss at other transient bottlenecks, e.g. due to bursts in shallower queues;

- * transmission errors, e.g. due to electrical interference;
- * rate policing.

Three complementary approaches are in progress to address this issue, but they are all currently research:

- * In Prague congestion control, ignore certain losses deemed unlikely to be due to congestion (using some ideas from BBR [I-D.cardwell-iccr-g-bbr-congestion-control] regarding isolated losses). This could mask any of the above types of loss while still coexisting with drop-based congestion controls.
- * A combination of RACK, L4S and link retransmission without resequencing could repair transmission errors without the head of line blocking delay usually associated with link-layer retransmission [UnorderedLTE], [I-D.ietf-tsvwg-ecn-l4s-id];
- * Hybrid ECN/drop rate policers (see Section 8.3).

L4S deployment scenarios that minimize these issues (e.g. over wireline networks) can proceed in parallel to this research, in the expectation that research success could continually widen L4S applicability.

6.4.4. L4S Flow but Classic ECN Bottleneck

Classic ECN support is starting to materialize on the Internet as an increased level of CE marking. It is hard to detect whether this is all due to the addition of support for ECN in implementations of FQ-CoDel and/or FQ-COBALT, which is not generally problematic, because flow-queue (FQ) scheduling inherently prevents a flow from exceeding the 'fair' rate irrespective of its aggressiveness. However, some of this Classic ECN marking might be due to single-queue ECN deployment. This case is discussed in Section 4.3 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id].

6.4.5. L4S AQM Deployment within Tunnels

An L4S AQM uses the ECN field to signal congestion. So, in common with Classic ECN, if the AQM is within a tunnel or at a lower layer, correct functioning of ECN signalling requires correct propagation of the ECN field up the layers [RFC6040], [I-D.ietf-tsvwg-rfc6040update-shim], [I-D.ietf-tsvwg-ecn-encap-guidelines].

7. IANA Considerations (to be removed by RFC Editor)

This specification contains no IANA considerations.

8. Security Considerations

8.1. Traffic Rate (Non-)Policing

In the current Internet, scheduling usually enforces separation between 'sites' (e.g. households, businesses or mobile users [RFC0970]) and various techniques like redirection to traffic scrubbing facilities deal with flooding attacks. However, there has never been a universal need to police the rate of individual application flows - the Internet has generally always relied on self-restraint of congestion controls at senders for sharing intra-'site' capacity.

As explained in Section 5.2, the DualQ variant of L4S provides low delay without prejudging the issue of flow-rate control. Then, if flow-rate control is needed, per-flow-queuing (FQ) can be used instead, or flow rate policing can be added as a modular addition to a DualQ.

Because the L4S service reduces delay without increasing the delay of Classic traffic, it should not be necessary to rate-police access to the L4S service. In contrast, Section 5.2 explains how Diffserv only makes a difference if some packets get less favourable treatment than others, which typically requires traffic rate policing, which can, in turn, lead to further complexity such as traffic contracts at trust boundaries. Because L4S avoids this management complexity, it is more likely to work end-to-end.

During early deployment (and perhaps always), some networks will not offer the L4S service. In general, these networks should not need to police L4S traffic. They are required (by both the ECN spec [RFC3168] and the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id]) not to change the L4S identifier, which would interfere with end-to-end congestion control. Instead they can merely treat L4S traffic as Not-ECT, as they might already treat all ECN traffic today. At a bottleneck, such networks will introduce some queuing and dropping. When a scalable congestion control detects a drop it will have to respond safely with respect to Classic congestion controls (as required in Section 4.3 of [I-D.ietf-tsvwg-ecn-l4s-id]). This will degrade the L4S service to be no better (but never worse) than Classic best efforts, whenever a non-ECN bottleneck is encountered on a path (see Section 6.4.3).

In cases that are expected to be rare, networks that solely support Classic ECN [RFC3168] in a single queue bottleneck might opt to police L4S traffic so as to protect competing Classic ECN traffic (for instance, see Section 6.1.3 of the L4S operational guidance [I-D.ietf-tsvwg-l4sops]). However, Section 4.3 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id] recommends that the sender adapts its congestion response to properly coexist with Classic ECN flows, i.e. reverting to the self-restraint approach.

Certain network operators might choose to restrict access to the L4S class, perhaps only to selected premium customers as a value-added service. Their packet classifier (item 2 in Figure 1) could identify such customers against some other field (e.g. source address range) as well as classifying on the ECN field. If only the ECN L4S identifier matched, but not the source address (say), the classifier could direct these packets (from non-premium customers) into the Classic queue. Explaining clearly how operators can use an additional local classifiers (see section 5.4 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id]) is intended to remove any motivation to clear the L4S identifier. Then at least the L4S ECN identifier will be more likely to survive end-to-end even though the service may not be supported at every hop. Such local arrangements would only require simple registered/not-registered packet classification, rather than the managed, application-specific traffic policing against customer-specific traffic contracts that Diffserv uses.

8.2. 'Latency Friendliness'

Like the Classic service, the L4S service relies on self-restraint - limiting rate in response to congestion. In addition, the L4S service requires self-restraint in terms of limiting latency (burstiness). It is hoped that self-interest and guidance on dynamic behaviour (especially flow start-up, which might need to be standardized) will be sufficient to prevent transports from sending excessive bursts of L4S traffic, given the application's own latency will suffer most from such behaviour.

Whether burst policing becomes necessary remains to be seen. Without it, there will be potential for attacks on the low latency of the L4S service.

If needed, various arrangements could be used to address this concern:

Local bottleneck queue protection: A per-flow (5-tuple) queue

protection function [I-D.briscoe-docsis-q-protection] has been developed for the low latency queue in DOCSIS, which has adopted the DualQ L4S architecture. It protects the low latency service from any queue-building flows that accidentally or maliciously classify themselves into the low latency queue. It is designed to score flows based solely on their contribution to queuing (not flow rate in itself). Then, if the shared low latency queue is at risk of exceeding a threshold, the function redirects enough packets of the highest scoring flow(s) into the Classic queue to preserve low latency.

Distributed traffic scrubbing: Rather than policing locally at each bottleneck, it may only be necessary to address problems reactively, e.g. punitively target any deployments of new bursty malware, in a similar way to how traffic from flooding attack sources is rerouted via scrubbing facilities.

Local bottleneck per-flow scheduling: Per-flow scheduling should inherently isolate non-bursty flows from bursty (see Section 5.2 for discussion of the merits of per-flow scheduling relative to per-flow policing).

Distributed access subnet queue protection: Per-flow queue protection could be arranged for a queue structure distributed across a subnet inter-communicating using lower layer control messages (see Section 2.1.4 of [QDyn]). For instance, in a radio access network, user equipment already sends regular buffer status reports to a radio network controller, which could use this information to remotely police individual flows.

Distributed Congestion Exposure to Ingress Policers: The Congestion Exposure (ConEx) architecture [RFC7713] which uses egress audit to motivate senders to truthfully signal path congestion in-band where it can be used by ingress policers. An edge-to-edge variant of this architecture is also possible.

Distributed Domain-edge traffic conditioning: An architecture similar to Diffserv [RFC2475] may be preferred, where traffic is proactively conditioned on entry to a domain, rather than reactively policed only if it leads to queuing once combined with other traffic at a bottleneck.

Distributed core network queue protection: The policing function

could be divided between per-flow mechanisms at the network ingress that characterize the burstiness of each flow into a signal carried with the traffic, and per-class mechanisms at bottlenecks that act on these signals if queuing actually occurs once the traffic converges. This would be somewhat similar to [Nadas20], which is in turn similar to the idea behind core stateless fair queuing.

None of these possible queue protection capabilities are considered a necessary part of the L4S architecture, which works without them (in a similar way to how the Internet works without per-flow rate policing). Indeed, even where latency policers are deployed, under normal circumstances they would not intervene, and if operators found they were not necessary they could disable them. Part of the L4S experiment will be to see whether such a function is necessary, and which arrangements are most appropriate to the size of the problem.

8.3. Interaction between Rate Policing and L4S

As mentioned in Section 5.2, L4S should remove the need for low latency Diffserv classes. However, those Diffserv classes that give certain applications or users priority over capacity, would still be applicable in certain scenarios (e.g. corporate networks). Then, within such Diffserv classes, L4S would often be applicable to give traffic low latency and low loss as well. Within such a Diffserv class, the bandwidth available to a user or application is often limited by a rate policer. Similarly, in the default Diffserv class, rate policers are used to partition shared capacity.

A classic rate policer drops any packets exceeding a set rate, usually also giving a burst allowance (variants exist where the policer re-marks non-compliant traffic to a discard-eligible Diffserv codepoint, so they can be dropped elsewhere during contention). Whenever L4S traffic encounters one of these rate policers, it will experience drops and the source will have to fall back to a Classic congestion control, thus losing the benefits of L4S (Section 6.4.3). So, in networks that already use rate policers and plan to deploy L4S, it will be preferable to redesign these rate policers to be more friendly to the L4S service.

L4S-friendly rate policing is currently a research area (note that this is not the same as latency policing). It might be achieved by setting a threshold where ECN marking is introduced, such that it is just under the policed rate or just under the burst allowance where drop is introduced. For instance the two-rate three-colour marker [RFC2698] or a PCN threshold and excess-rate marker [RFC5670] could mark ECN at the lower rate and drop at the higher. Or an existing rate policer could have congestion-rate policing added,

e.g. using the 'local' (non-ConEx) variant of the ConEx aggregate congestion policer [I-D.briscoe-conex-policing]. It might also be possible to design scalable congestion controls to respond less catastrophically to loss that has not been preceded by a period of increasing delay.

The design of L4S-friendly rate policers will require a separate dedicated document. For further discussion of the interaction between L4S and Diffserv, see [I-D.briscoe-tsvwg-l4s-diffserv].

8.4. ECN Integrity

Receiving hosts can fool a sender into downloading faster by suppressing feedback of ECN marks (or of losses if retransmissions are not necessary or available otherwise). Various ways to protect transport feedback integrity have been developed. For instance:

- * The sender can test the integrity of the receiver's feedback by occasionally setting the IP-ECN field to the congestion experienced (CE) codepoint, which is normally only set by a congested link. Then the sender can test whether the receiver's feedback faithfully reports what it expects (see 2nd para of Section 20.2 of the Classic ECN spec [RFC3168]).
- * A network can enforce a congestion response to its ECN markings (or packet losses) by auditing congestion exposure (ConEx) [RFC7713].
- * Transport layer authentication such as the TCP authentication option (TCP-AO [RFC5925]) or QUIC's use of TLS [RFC9001] can detect any tampering with congestion feedback.
- * The ECN Nonce [RFC3540] was proposed to detect tampering with congestion feedback, but it has been reclassified as historic [RFC8311].

Appendix C.1 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id] gives more details of these techniques including their applicability and pros and cons.

8.5. Privacy Considerations

As discussed in Section 5.2, the L4S architecture does not preclude approaches that inspect end-to-end transport layer identifiers. For instance, L4S support has been added to FQ-CoDel, which classifies by application flow ID in the network. However, the main innovation of L4S is the DualQ AQM framework that does not need to inspect any deeper than the outermost IP header, because the L4S identifier is in

the IP-ECN field.

Thus, the L4S architecture enables very low queuing delay without requiring inspection of information above the IP layer. This means that users who want to encrypt application flow identifiers, e.g. in IPSec or other encrypted VPN tunnels, don't have to sacrifice low delay [RFC8404].

Because L4S can provide low delay for a broad set of applications that choose to use it, there is no need for individual applications or classes within that broad set to be distinguishable in any way while traversing networks. This removes much of the ability to correlate between the delay requirements of traffic and other identifying features [RFC6973]. There may be some types of traffic that prefer not to use L4S, but the coarse binary categorization of traffic reveals very little that could be exploited to compromise privacy.

9. Acknowledgements

Thanks to Richard Scheffenegger, Wes Eddy, Karen Nielsen, David Black, Jake Holland, Vidhi Goel, Ermin Sakic, Praveen Balasubramanian, Gorrry Fairhurst, Mirja Kuehlewind, Philip Eardley, Neal Cardwell and Pete Heist for their useful review comments.

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

10. Informative References

- [AFCD] Xue, L., Kumar, S., Cui, C., Kondikoppa, P., Chiu, C-H., and S-J. Park, "Towards fair and low latency next generation high speed networks: AFCD queuing", Journal of Network and Computer Applications 70:183--193, July 2016, <<https://doi.org/10.1016/j.jnca.2016.03.021>>.
- [BBRv2] Cardwell, N., "TCP BBR v2 Alpha/Preview Release", github repository; Linux congestion control module, <<https://github.com/google/bbr/blob/v2alpha/README.md>>.

- [BDPdata] Briscoe, B., "PI2 Parameters", Technical Report TR-BB-2021-001 arXiv:2107.01003 [cs.NI], July 2021, <<https://arxiv.org/abs/2107.01003>>.
- [BufferSize] Appenzeller, G., Keslassy, I., and N. McKeown, "Sizing Router Buffers", In Proc. SIGCOMM'04 34(4):281--292, September 2004, <<https://doi.org/10.1145/1015467.1015499>>.
- [COBALT] Palmei, J., Gupta, S., Imputato, P., Morton, J., Tahiliani, M. P., Avallone, S., and D. Täht, "Design and Evaluation of COBALT Queue Discipline", In Proc. IEEE Int'l Symp. Local and Metropolitan Area Networks (LANMAN'19) 2019:1-6, July 2019, <<https://ieeexplore.ieee.org/abstract/document/8847054>>.
- [DCttH19] De Schepper, K., Bondarenko, O., Tilmans, O., and B. Briscoe, "'Data Centre to the Home': Ultra-Low Latency for All", Updated RITE project Technical Report , July 2019, <https://bobbbriscoe.net/pubs.html#DCttH_TR>.
- [DOCSIS3.1] CableLabs, "MAC and Upper Layer Protocols Interface (MULPI) Specification, CM-SP-MULPIv3.1", Data-Over-Cable Service Interface Specifications DOCSIS® 3.1 Version i17 or later, 21 January 2019, <<https://specification-search.cablelabs.com/CM-SP-MULPIv3.1>>.
- [DOCSIS3AQM] White, G., "Active Queue Management Algorithms for DOCSIS 3.0; A Simulation Study of CoDel, SFQ-CoDel and PIE in DOCSIS 3.0 Networks", CableLabs Technical Report , April 2013, <http://www.cablelabs.com/wp-content/uploads/2013/11/Active_Queue_Management_Algorithms_DOCSIS_3_0.pdf>.
- [DualPI2Linux] Albisser, O., De Schepper, K., Briscoe, B., Tilmans, O., and H. Steen, "DUALPI2 - Low Latency, Low Loss and Scalable (L4S) AQM", Proc. Linux Netdev 0x13 , March 2019, <<https://www.netdevconf.org/0x13/session.html?talk-DUALPI2-AQM>>.
- [Dukkipati06] Dukkipati, N. and N. McKeown, "Why Flow-Completion Time is the Right Metric for Congestion Control", ACM CCR 36(1):59--62, January 2006, <<https://dl.acm.org/doi/10.1145/1111322.1111336>>.

[FQ_CoDel_Thresh]

Høiland-Jørgensen, T., "fq_codel: generalise ce_threshold marking for subset of traffic", Linux Patch Commit ID: dfcb63celde6b10b, 20 October 2021, <<https://git.kernel.org/pub/scm/linux/kernel/git/netdev/net-next.git/commit/?id=dfcb63celde6b10b>>.

[Hohlfeld14]

Hohlfeld, O., Pujol, E., Ciucu, F., Feldmann, A., and P. Barford, "A QoE Perspective on Sizing Network Buffers", Proc. ACM Internet Measurement Conf (IMC'14) hmm, November 2014, <<http://doi.acm.org/10.1145/2663716.2663730>>.

[I-D.briscoe-conex-policing]

Briscoe, B., "Network Performance Isolation using Congestion Policing", Work in Progress, Internet-Draft, draft-briscoe-conex-policing-01, 14 February 2014, <<https://datatracker.ietf.org/doc/html/draft-briscoe-conex-policing-01>>.

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

Briscoe, B. and G. White, "The DOCSIS(r) Queue Protection Algorithm to Preserve Low Latency", Work in Progress, Internet-Draft, draft-briscoe-docsis-q-protection-02, 31 January 2022, <<https://datatracker.ietf.org/doc/html/draft-briscoe-docsis-q-protection-02>>.

[I-D.briscoe-iccrp-prague-congestion-control]

Schepper, K. D., Tilmans, O., and B. Briscoe, "Prague Congestion Control", Work in Progress, Internet-Draft, draft-briscoe-iccrp-prague-congestion-control-00, 9 March 2021, <<https://datatracker.ietf.org/doc/html/draft-briscoe-iccrp-prague-congestion-control-00>>.

[I-D.briscoe-tsvwg-l4s-diffserv]

Briscoe, B., "Interactions between Low Latency, Low Loss, Scalable Throughput (L4S) and Differentiated Services", Work in Progress, Internet-Draft, draft-briscoe-tsvwg-l4s-diffserv-02, 4 November 2018, <<https://datatracker.ietf.org/doc/html/draft-briscoe-tsvwg-l4s-diffserv-02>>.

- [I-D.cardwell-iccr-g-bbr-congestion-control]
Cardwell, N., Cheng, Y., Yeganeh, S. H., Swett, I., and V. Jacobson, "BBR Congestion Control", Work in Progress, Internet-Draft, draft-cardwell-iccr-g-bbr-congestion-control-01, 7 November 2021, <<https://datatracker.ietf.org/doc/html/draft-cardwell-iccr-g-bbr-congestion-control-01>>.
- [I-D.ietf-tcpm-accurate-ecn]
Briscoe, B., Kühlewind, M., and R. Scheffenegger, "More Accurate ECN Feedback in TCP", Work in Progress, Internet-Draft, draft-ietf-tcpm-accurate-ecn-16, 3 February 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-accurate-ecn-16>>.
- [I-D.ietf-tsvwg-aqm-dualq-coupled]
Schepper, K. D., Briscoe, B., and G. White, "DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)", Work in Progress, Internet-Draft, draft-ietf-tsvwg-aqm-dualq-coupled-22, 4 March 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-aqm-dualq-coupled-22>>.
- [I-D.ietf-tsvwg-ecn-encap-guidelines]
Briscoe, B. and J. Kaippallimalil, "Guidelines for Adding Congestion Notification to Protocols that Encapsulate IP", Work in Progress, Internet-Draft, draft-ietf-tsvwg-ecn-encap-guidelines-16, 25 May 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-ecn-encap-guidelines-16>>.
- [I-D.ietf-tsvwg-ecn-l4s-id]
Schepper, K. D. and B. Briscoe, "Explicit Congestion Notification (ECN) Protocol for Very Low Queuing Delay (L4S)", Work in Progress, Internet-Draft, draft-ietf-tsvwg-ecn-l4s-id-24, 1 February 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-ecn-l4s-id-24>>.
- [I-D.ietf-tsvwg-l4sops]
White, G., "Operational Guidance for Deployment of L4S in the Internet", Work in Progress, Internet-Draft, draft-ietf-tsvwg-l4sops-02, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-l4sops-02>>.

[I-D.ietf-tsvwg-nqb]

White, G. and T. Fossati, "A Non-Queue-Building Per-Hop Behavior (NQB PHB) for Differentiated Services", Work in Progress, Internet-Draft, draft-ietf-tsvwg-nqb-10, 4 March 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-nqb-10>>.

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

Briscoe, B., "Propagating Explicit Congestion Notification Across IP Tunnel Headers Separated by a Shim", Work in Progress, Internet-Draft, draft-ietf-tsvwg-rfc6040update-shim-14, 25 May 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-rfc6040update-shim-14>>.

[I-D.morton-tsvwg-codel-approx-fair]

Morton, J. and P. G. Heist, "Controlled Delay Approximate Fairness AQM", Work in Progress, Internet-Draft, draft-morton-tsvwg-codel-approx-fair-01, 9 March 2020, <<https://datatracker.ietf.org/doc/html/draft-morton-tsvwg-codel-approx-fair-01>>.

[I-D.sridharan-tcpm-ctcp]

Sridharan, M., Tan, K., Bansal, D., and D. Thaler, "Compound TCP: A New TCP Congestion Control for High-Speed and Long Distance Networks", Work in Progress, Internet-Draft, draft-sridharan-tcpm-ctcp-02, 11 November 2008, <<https://datatracker.ietf.org/doc/html/draft-sridharan-tcpm-ctcp-02>>.

[I-D.stewart-tsvwg-sctpecn]

Stewart, R. R., Tuexen, M., and X. Dong, "ECN for Stream Control Transmission Protocol (SCTP)", Work in Progress, Internet-Draft, draft-stewart-tsvwg-sctpecn-05, 15 January 2014, <<https://datatracker.ietf.org/doc/html/draft-stewart-tsvwg-sctpecn-05>>.

[L4Sdemo16]

Bondarenko, O., De Schepper, K., Tsang, I., and B. Briscoe, "Ultra-Low Delay for All: Live Experience, Live Analysis", Proc. MMSYS'16 pp33:1--33:4, May 2016, <<http://dl.acm.org/citation.cfm?doid=2910017.2910633> (videos of demos: <https://riteproject.eu/dctth/#1511dispatchwg>)>.

[LEDBAT_AQM]

Al-Saadi, R., Armitage, G., and J. But, "Characterising LEDBAT Performance Through Bottlenecks Using PIE, FQ-CoDel

- and FQ-PIE Active Queue Management", Proc. IEEE 42nd Conference on Local Computer Networks (LCN) 278--285, 2017, <<https://ieeexplore.ieee.org/document/8109367>>.
- [lowat] Meenan, P., "Optimizing HTTP/2 prioritization with BBR and tcp_notsent_lowat", Cloudflare Blog , 12 October 2018, <<https://blog.cloudflare.com/http-2-prioritization-with-nginx/>>.
- [Mathis09] Mathis, M., "Relentless Congestion Control", PFLDNeT'09 , May 2009, <<https://www.gdt.id.au/~gdt/presentations/2010-07-06-questnet-tcp/reference-materials/papers/mathis-relentless-congestion-control.pdf>>.
- [McIlroy78] McIlroy, M.D., Pinson, E. N., and B. A. Tague, "UNIX Time-Sharing System: Foreword", The Bell System Technical Journal 57:6(1902--1903), July 1978, <<https://archive.org/details/bstj57-6-1899>>.
- [Nadas20] Nádas, S., Gombos, G., Fejes, F., and S. Laki, "A Congestion Control Independent L4S Scheduler", Proc. Applied Networking Research Workshop (ANRW '20) 45--51, July 2020, <<https://doi.org/10.1145/3404868.3406669>>.
- [PragueLinux] Briscoe, B., De Schepper, K., Albisser, O., Misund, J., Tilmans, O., Kühlewind, M., and A.S. Ahmed, "Implementing the 'TCP Prague' Requirements for Low Latency Low Loss Scalable Throughput (L4S)", Proc. Linux Netdev 0x13 , March 2019, <<https://www.netdevconf.org/0x13/session.html?talk-tcp-prague-l4s>>.
- [QDyn] Briscoe, B., "Rapid Signalling of Queue Dynamics", bobbriscoe.net Technical Report TR-BB-2017-001; arXiv:1904.07044 [cs.NI], September 2017, <<https://arxiv.org/abs/1904.07044>>.
- [Rajiullah15] Rajiullah, M., "Towards a Low Latency Internet: Understanding and Solutions", Masters Thesis; Karlstad Uni, Dept of Maths & CS 2015:41, 2015, <<https://www.diva-portal.org/smash/get/diva2:846109/FULLTEXT01.pdf>>.
- [RFC0970] Nagle, J., "On Packet Switches With Infinite Storage", RFC 970, DOI 10.17487/RFC0970, December 1985, <<https://www.rfc-editor.org/info/rfc970>>.

- [RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, DOI 10.17487/RFC2475, December 1998, <<https://www.rfc-editor.org/info/rfc2475>>.
- [RFC2698] Heinanen, J. and R. Guerin, "A Two Rate Three Color Marker", RFC 2698, DOI 10.17487/RFC2698, September 1999, <<https://www.rfc-editor.org/info/rfc2698>>.
- [RFC2884] Hadi Salim, J. and U. Ahmed, "Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks", RFC 2884, DOI 10.17487/RFC2884, July 2000, <<https://www.rfc-editor.org/info/rfc2884>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC3246] Davie, B., Charny, A., Bennet, J.C.R., Benson, K., Le Boudec, J.Y., Courtney, W., Davari, S., Firoiu, V., and D. Stiliadis, "An Expedited Forwarding PHB (Per-Hop Behavior)", RFC 3246, DOI 10.17487/RFC3246, March 2002, <<https://www.rfc-editor.org/info/rfc3246>>.
- [RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, DOI 10.17487/RFC3540, June 2003, <<https://www.rfc-editor.org/info/rfc3540>>.
- [RFC3649] Floyd, S., "HighSpeed TCP for Large Congestion Windows", RFC 3649, DOI 10.17487/RFC3649, December 2003, <<https://www.rfc-editor.org/info/rfc3649>>.
- [RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI 10.17487/RFC4340, March 2006, <<https://www.rfc-editor.org/info/rfc4340>>.
- [RFC4774] Floyd, S., "Specifying Alternate Semantics for the Explicit Congestion Notification (ECN) Field", BCP 124, RFC 4774, DOI 10.17487/RFC4774, November 2006, <<https://www.rfc-editor.org/info/rfc4774>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.

- [RFC5033] Floyd, S. and M. Allman, "Specifying New Congestion Control Algorithms", BCP 133, RFC 5033, DOI 10.17487/RFC5033, August 2007, <<https://www.rfc-editor.org/info/rfc5033>>.
- [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <<https://www.rfc-editor.org/info/rfc5348>>.
- [RFC5670] Eardley, P., Ed., "Metering and Marking Behaviour of PCN-Nodes", RFC 5670, DOI 10.17487/RFC5670, November 2009, <<https://www.rfc-editor.org/info/rfc5670>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/info/rfc5681>>.
- [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <<https://www.rfc-editor.org/info/rfc5925>>.
- [RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <<https://www.rfc-editor.org/info/rfc6040>>.
- [RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", RFC 6679, DOI 10.17487/RFC6679, August 2012, <<https://www.rfc-editor.org/info/rfc6679>>.
- [RFC6817] Shalunov, S., Hazel, G., Iyengar, J., and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)", RFC 6817, DOI 10.17487/RFC6817, December 2012, <<https://www.rfc-editor.org/info/rfc6817>>.
- [RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <<https://www.rfc-editor.org/info/rfc6973>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<https://www.rfc-editor.org/info/rfc7540>>.

- [RFC7560] Kuehlewind, M., Ed., Scheffenegger, R., and B. Briscoe, "Problem Statement and Requirements for Increased Accuracy in Explicit Congestion Notification (ECN) Feedback", RFC 7560, DOI 10.17487/RFC7560, August 2015, <<https://www.rfc-editor.org/info/rfc7560>>.
- [RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <<https://www.rfc-editor.org/info/rfc7567>>.
- [RFC7665] Halpern, J., Ed. and C. Pignataro, Ed., "Service Function Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <<https://www.rfc-editor.org/info/rfc7665>>.
- [RFC7713] Mathis, M. and B. Briscoe, "Congestion Exposure (ConEx) Concepts, Abstract Mechanism, and Requirements", RFC 7713, DOI 10.17487/RFC7713, December 2015, <<https://www.rfc-editor.org/info/rfc7713>>.
- [RFC8033] Pan, R., Natarajan, P., Baker, F., and G. White, "Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem", RFC 8033, DOI 10.17487/RFC8033, February 2017, <<https://www.rfc-editor.org/info/rfc8033>>.
- [RFC8034] White, G. and R. Pan, "Active Queue Management (AQM) Based on Proportional Integral Controller Enhanced (PIE) for Data-Over-Cable Service Interface Specifications (DOCSIS) Cable Modems", RFC 8034, DOI 10.17487/RFC8034, February 2017, <<https://www.rfc-editor.org/info/rfc8034>>.
- [RFC8170] Thaler, D., Ed., "Planning for Protocol Adoption and Subsequent Transitions", RFC 8170, DOI 10.17487/RFC8170, May 2017, <<https://www.rfc-editor.org/info/rfc8170>>.
- [RFC8257] Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <<https://www.rfc-editor.org/info/rfc8257>>.
- [RFC8290] Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm", RFC 8290, DOI 10.17487/RFC8290, January 2018, <<https://www.rfc-editor.org/info/rfc8290>>.

- [RFC8298] Johansson, I. and Z. Sarker, "Self-Clocked Rate Adaptation for Multimedia", RFC 8298, DOI 10.17487/RFC8298, December 2017, <<https://www.rfc-editor.org/info/rfc8298>>.
- [RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <<https://www.rfc-editor.org/info/rfc8311>>.
- [RFC8312] Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", RFC 8312, DOI 10.17487/RFC8312, February 2018, <<https://www.rfc-editor.org/info/rfc8312>>.
- [RFC8404] Moriarty, K., Ed. and A. Morton, Ed., "Effects of Pervasive Encryption on Operators", RFC 8404, DOI 10.17487/RFC8404, July 2018, <<https://www.rfc-editor.org/info/rfc8404>>.
- [RFC8511] Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst, "TCP Alternative Backoff with ECN (ABE)", RFC 8511, DOI 10.17487/RFC8511, December 2018, <<https://www.rfc-editor.org/info/rfc8511>>.
- [RFC8888] Sarker, Z., Perkins, C., Singh, V., and M. Ramalho, "RTP Control Protocol (RTCP) Feedback for Congestion Control", RFC 8888, DOI 10.17487/RFC8888, January 2021, <<https://www.rfc-editor.org/info/rfc8888>>.
- [RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <<https://www.rfc-editor.org/info/rfc9000>>.
- [RFC9001] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <<https://www.rfc-editor.org/info/rfc9001>>.
- [SCReAM] Johansson, I., "SCReAM", github repository; , <<https://github.com/EricssonResearch/scream/blob/master/README.md>>.
- [TCP-CA] Jacobson, V. and M.J. Karels, "Congestion Avoidance and Control", Laurence Berkeley Labs Technical Report , November 1988, <<http://ee.lbl.gov/papers/congavoid.pdf>>.

[UnorderedLTE]

Austrheim, M.V., "Implementing immediate forwarding for 4G in a network simulator", Masters Thesis, Uni Oslo , June 2019.

Authors' Addresses

Bob Briscoe (editor)
Independent
United Kingdom
Email: ietf@bobbriscoe.net
URI: <http://bobbriscoe.net/>

Koen De Schepper
Nokia Bell Labs
Antwerp
Belgium
Email: koen.de_schepper@nokia.com
URI: https://www.bell-labs.com/usr/koen.de_schepper

Marcelo Bagnulo
Universidad Carlos III de Madrid
Av. Universidad 30
Leganes, Madrid 28911
Spain
Phone: 34 91 6249500
Email: marcelo@it.uc3m.es
URI: <http://www.it.uc3m.es>

Greg White
CableLabs
United States of America
Email: G.White@CableLabs.com

Internet Engineering Task Force
Internet-Draft
Obsoletes: 3662 (if approved)
Updates: 4594,8325 (if approved)
Intended status: Standards Track
Expires: September 12, 2019

R. Bless
Karlsruhe Institute of Technology (KIT)
March 11, 2019

A Lower Effort Per-Hop Behavior (LE PHB) for Differentiated Services
draft-ietf-tsvwg-le-phb-10

Abstract

This document specifies properties and characteristics of a Lower Effort (LE) per-hop behavior (PHB). The primary objective of this LE PHB is to protect best-effort (BE) traffic (packets forwarded with the default PHB) from LE traffic in congestion situations, i.e., when resources become scarce, best-effort traffic has precedence over LE traffic and may preempt it. Alternatively, packets forwarded by the LE PHB can be associated with a scavenger service class, i.e., they scavenge otherwise unused resources only. There are numerous uses for this PHB, e.g., for background traffic of low precedence, such as bulk data transfers with low priority in time, non time-critical backups, larger software updates, web search engines while gathering information from web servers and so on. This document recommends a standard DSCP value for the LE PHB. This specification obsoletes RFC 3662 and updates the DSCP recommended in RFC 4594 and RFC 8325 to use the DSCP assigned in this specification.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 12, 2019.

Copyright Notice

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

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

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	3
2. Requirements Language	3
3. Applicability	3
4. PHB Description	6
5. Traffic Conditioning Actions	7
6. Recommended DS Codepoint	7
7. Deployment Considerations	7
8. Remarking to other DSCPs/PHBs	8
9. Multicast Considerations	9
10. The Update to RFC 4594	10
11. The Update to RFC 8325	12
12. The Update to draft-ietf-tsvwg-rtcweb-qos	12
13. IANA Considerations	14
14. Security Considerations	14
15. References	15
15.1. Normative References	15
15.2. Informative References	15
Appendix A. History of the LE PHB	17
Appendix B. Acknowledgments	18

Appendix C. Change History	18
Appendix D. Note to RFC Editor	21
Author's Address	21

1. Introduction

This document defines a Differentiated Services per-hop behavior [RFC2474] called "Lower Effort" (LE), which is intended for traffic of sufficiently low urgency that all other traffic takes precedence over the LE traffic in consumption of network link bandwidth. Low urgency traffic has a low priority for timely forwarding, which does not necessarily imply that it is generally of minor importance. From this viewpoint, it can be considered as a network equivalent to a background priority for processes in an operating system. There may or may not be memory (buffer) resources allocated for this type of traffic.

Some networks carry packets that ought to consume network resources only when no other traffic is demanding them. In this point of view, packets forwarded by the LE PHB scavenge otherwise unused resources only, which led to the name "scavenger service" in early Internet2 deployments (see Appendix A). Other commonly used names for LE PHB type services are "Lower-than-best-effort" or "Less-than-best-effort". In summary, with the mentioned feature above, the LE PHB has two important properties: it should scavenge residual capacity and it must be preemptable by the default PHB (or other elevated PHBs) in case they need more resources. Consequently, the effect of this type of traffic on all other network traffic is strictly limited ("no harm" property). This is distinct from "best-effort" (BE) traffic since the network makes no commitment to deliver LE packets. In contrast, BE traffic receives an implied "good faith" commitment of at least some available network resources. This document proposes a Lower Effort Differentiated Services per-hop behavior (LE PHB) for handling this "optional" traffic in a differentiated services node.

2. Requirements Language

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

3. Applicability

A Lower Effort PHB is applicable for many applications that otherwise use best-effort delivery. More specifically, it is suitable for traffic and services that can tolerate strongly varying throughput

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

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

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

is expected to have applicability in networks that have at least some unused capacity at certain periods.

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

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

Example uses for the LE PHB:

- o For traffic caused by world-wide web search engines while they gather information from web servers.
- o For software updates or dissemination of new releases of operating systems.
- o For reporting errors or telemetry data from operating systems or applications.

- o For backup traffic or non-time critical synchronization or mirroring traffic.
- o For content distribution transfers between caches.
- o For preloading or prefetching objects from web sites.
- o For network news and other "bulk mail" of the Internet.
- o For "downgraded" traffic from some other PHB when this does not violate the operational objectives of the other PHB.
- o For multicast traffic from untrusted (e.g., non-local) sources.

4. PHB Description

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

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

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

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

5. Traffic Conditioning Actions

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

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

6. Recommended DS Codepoint

The RECOMMENDED codepoint for the LE PHB is '000001'.

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

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

7. Deployment Considerations

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

- o A BA classifier (Behavior Aggregate classifier, see [RFC2475]) that classifies packets according to the LE DSCP
- o A dedicated LE queue
- o A suitable scheduling discipline, e.g., simple priority queueing

Alternatively, implementations could use active queue management mechanisms instead of a dedicated LE queue, e.g., dropping all arriving LE packets when certain queue length or sojourn time thresholds are exceeded.

Internet-wide deployment of the LE PHB is eased by the following properties:

- o No harm to other traffic: since the LE PHB has the lowest forwarding priority it does not consume resources from other PHBs. Deployment across different provider domains with LE support causes no trust issues or attack vectors to existing (non LE) traffic. Thus, providers can trust LE markings from end-systems, i.e., there is no need to police or remark incoming LE traffic.
- o No PHB parameters or configuration of traffic profiles: the LE PHB itself possesses no parameters that need to be set or configured. Similarly, since LE traffic requires no admission or policing, it is not necessary to configure traffic profiles.
- o No traffic conditioning mechanisms: the LE PHB requires no traffic meters, droppers, or shapers. See also Section 5 for further discussion.

Operators of DS domains that cannot or do not want to implement the LE PHB (e.g., because there is no separate LE queue available in the corresponding nodes) SHOULD NOT drop packets marked with the LE DSCP. They SHOULD map packets with this DSCP to the default PHB and SHOULD preserve the LE DSCP marking. DS domains operators that do not implement the LE PHB should be aware that they violate the "no harm" property of LE. See also Section 8 for further discussion of forwarding LE traffic with the default PHB instead.

8. Remarking to other DSCPs/PHBs

"DSCP bleaching", i.e., setting the DSCP to '000000' (default PHB) is NOT RECOMMENDED for this PHB. This may cause effects that are in contrast to the original intent in protecting BE traffic from LE traffic (no harm property). In the case that a DS domain does not support the LE PHB, its nodes SHOULD treat LE marked packets with the default PHB instead (by mapping the LE DSCP to the default PHB), but they SHOULD do so without remarking to DSCP '000000'. The reason for this is that later traversed DS domains may then have still the possibility to treat such packets according to the LE PHB.

Operators of DS domains that forward LE traffic within the BE aggregate need to be aware of the implications, i.e., induced congestion situations and quality-of-service degradation of the

original BE traffic. In this case, the LE property of not harming other traffic is no longer fulfilled. To limit the impact in such cases, traffic policing of the LE aggregate MAY be used.

In the case that LE marked packets are effectively carried within the default PHB (i.e., forwarded as best-effort traffic) they get a better forwarding treatment than expected. For some applications and services, it is favorable if the transmission is finished earlier than expected. However, in some cases it may be against the original intention of the LE PHB user to strictly send the traffic only if otherwise unused resources are available. In the case that LE traffic is mapped to the default PHB, LE traffic may compete with BE traffic for the same resources and thus adversely affect the original BE aggregate. Applications that want to ensure the lower precedence compared to BE traffic even in such cases SHOULD use additionally a corresponding Lower-than-Best-Effort transport protocol [RFC6297], e.g., LEDBAT [RFC6817].

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

9. Multicast Considerations

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

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

Superficially viewed, it may sound very attractive to use IP multicast with the LE PHB to build this type of opportunistic reliable distribution in IP networks, but it can only be usefully

deployed with routers that do not experience forwarding/replication resource starvation when a large amount of packets (virtually) need to be replicated to links where the LE queue is full.

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

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

10. The Update to RFC 4594

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

Changes:

- o This update to RFC 4594 removes the following entry from figure 3:

Low-Priority Data	CS1	001000	Any flow that has no BW assurance
----------------------	-----	--------	--------------------------------------

and replaces this by the following entry:

Low-Priority Data	LE	000001	Any flow that has no BW assurance
----------------------	----	--------	--------------------------------------

- o This update to RFC 4594 extends the Notes text below figure 3 that currently states "Notes for Figure 3: Default Forwarding (DF) and Class Selector 0 (CS0) provide equivalent behavior and use the same DS codepoint, '000000'." to state "Notes for Figure 3: Default Forwarding (DF) and Class Selector 0 (CS0) provide equivalent behavior and use the same DS codepoint, '000000'. The prior recommendation to use the CS1 DSCP for Low-Priority Data has been replaced by the current recommendation to use the LE DSCP, '000001'."
- o This update to RFC 4594 removes the following entry from figure 4:

Low-Priority Data	CS1	Not applicable	RFC3662	Rate	Yes
----------------------	-----	----------------	---------	------	-----

and replaces this by the following entry:

Low-Priority Data	LE	Not applicable	RFCXXXX	Rate	Yes
----------------------	----	----------------	---------	------	-----

- o Section 2.3 of [RFC4594] specifies: "In network segments that use IP precedence marking, only one of the two service classes can be supported, High-Throughput Data or Low-Priority Data. We RECOMMEND that the DSCP value(s) of the unsupported service class be changed to 000xx1 on ingress and changed back to original value(s) on egress of the network segment that uses precedence marking. For example, if Low-Priority Data is mapped to Standard service class, then 000001 DSCP marking MAY be used to distinguish it from Standard marked packets on egress." This document removes this recommendation, because by using the herein defined LE DSCP such remarking is not necessary. So even if Low-Priority Data is unsupported (i.e., mapped to the default PHB) the LE DSCP should be kept across the domain as RECOMMENDED in Section 8. That removed text is replaced by: "In network segments that use IP Precedence marking, the Low-Priority Data service class receives the same Diffserv QoS as the Standard service class when the LE DSCP is used for Low-Priority Data traffic. This is acceptable behavior for the Low-Priority Data service class, although it is not the preferred behavior."

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

11. The Update to RFC 8325

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

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

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

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

Low-Priority Data	CS1	RFC 3662	1	AC_BK (Background)
-------------------	-----	----------	---	--------------------

by the following entries:

Low-Priority Data	LE	RFCXXXX	1	AC_BK (Background)
Low-Priority Data (legacy)	CS1	RFC 3662	1	AC_BK (Background)

12. The Update to draft-ietf-tsvwg-rtcweb-qos

Section 5 of [I-D.ietf-tsvwg-rtcweb-qos] describes the Recommended DSCP Values for WebRTC Applications

This update to [I-D.ietf-tsvwg-rtcweb-qos] replaces all occurrences of CS1 with LE in Table 1:

Flow Type	Very Low	Low	Medium	High
Audio	LE (1)	DF (0)	EF (46)	EF (46)
Interactive Video with or without Audio	LE (1)	DF (0)	AF42, AF43 (36, 38)	AF41, AF42 (34, 36)
Non-Interactive Video with or without Audio	LE (1)	DF (0)	AF32, AF33 (28, 30)	AF31, AF32 (26, 28)
Data	LE (1)	DF (0)	AF11	AF21

and updates the following paragraph:

"The above table assumes that packets marked with CS1 are treated as "less than best effort", such as the LE behavior described in [RFC3662]. However, the treatment of CS1 is implementation dependent. If an implementation treats CS1 as other than "less than best effort", then the actual priority (or, more precisely, the per-hop-behavior) of the packets may be changed from what is intended. It is common for CS1 to be treated the same as DF, so applications and browsers using CS1 cannot assume that CS1 will be treated differently than DF [RFC7657]. However, it is also possible per [RFC2474] for CS1 traffic to be given better treatment than DF, thus caution should be exercised when electing to use CS1. This is one of the cases where marking packets using these recommendations can make things worse."

as follows:

"The above table assumes that packets marked with LE are treated as lower effort (i.e., "less than best effort"), such as the LE behavior described in [RFCXXXX]. However, the treatment of LE is implementation dependent. If an implementation treats LE as other than "less than best effort", then the actual priority (or, more precisely, the per-hop-behavior) of the packets may be changed from what is intended. It is common for LE to be treated the same as DF, so applications and browsers using LE cannot assume that LE will be treated differently than DF [RFC7657]. During development of this document, the CS1 DSCP was recommended for "very low" application

priority traffic; implementations that followed that recommendation SHOULD be updated to use the LE DSCP instead of the CS1 DSCP."

13. IANA Considerations

This document assigns the Differentiated Services Field Codepoint (DSCP) '000001' from the Differentiated Services Field Codepoints (DSCP) registry (<https://www.iana.org/assignments/dscp-registry/dscp-registry.xhtml>) (Pool 3, Codepoint Space xxxx01, Standards Action) to the LE PHB. This document suggests to use a DSCP from Pool 3 in order to avoid problems for other PHB marked flows to become accidentally remarked as LE PHB, e.g., due to partial DSCP bleaching. See [RFC8436] for re-classifying Pool 3 for Standards Action.

IANA is requested to update the registry as follows:

- o Name: LE
- o Value (Binary): 000001
- o Value (Decimal): 1
- o Reference: [RFC number of this memo]

14. Security Considerations

There are no specific security exposures for this PHB. Since it defines a new class of low forwarding priority, remarking other traffic as LE traffic may lead to quality-of-service degradation of such traffic. Thus, any attacker that is able to modify the DSCP of a packet to LE may carry out a downgrade attack. See the general security considerations in [RFC2474] and [RFC2475].

With respect to privacy, an attacker could use the information from the DSCP to infer that the transferred (probably even encrypted) content is considered of low priority or low urgency by a user, in case the DSCP was set on the user's request. On the one hand, this disclosed information is useful only if correlation with metadata (such as the user's IP address) and/or other flows reveal user identity. On the other hand, it might help an observer (e.g., a state level actor) who is interested in learning about the user's behavior from observed traffic: LE marked background traffic (such as software downloads, operating system updates, or telemetry data) may be less interesting for surveillance than general web traffic. Therefore, the LE marking may help the observer to focus on potentially more interesting traffic (however, the user may exploit this particular assumption and deliberately hide interesting traffic in the LE aggregate). Apart from such considerations, the impact of

disclosed information by the LE DSCP is likely negligible in most cases given the numerous traffic analysis possibilities and general privacy threats (e.g., see [RFC6973]).

15. References

15.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <<http://www.rfc-editor.org/info/rfc2474>>.
- [RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, DOI 10.17487/RFC2475, December 1998, <<http://www.rfc-editor.org/info/rfc2475>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

15.2. Informative References

- [carlberg-lbe-2001] Carlberg, K., Gevros, P., and J. Crowcroft, "Lower than best effort: a design and implementation", SIGCOMM Computer Communications Review Volume 31, Issue 2 supplement, April 2001, <<https://doi.org/10.1145/844193.844208>>.
- [chown-lbe-2003] Chown, T., Ferrari, T., Leinen, S., Sabatino, R., Simar, N., and S. Venaas, "Less than Best Effort: Application Scenarios and Experimental Results", In Proceedings of the Second International Workshop on Quality of Service in Multiservice IP Networks (QoS-IP 2003), Lecture Notes in Computer Science, vol 2601. Springer, Berlin, Heidelberg Pages 131-144, February 2003, <https://doi.org/10.1007/3-540-36480-3_10>.

- [draft-bless-diffserv-lbe-phb-00]
Bless, R. and K. Wehrle, "A Lower Than Best-Effort Per-Hop Behavior", draft-bless-diffserv-lbe-phb-00 (work in progress), September 1999, <<https://tools.ietf.org/html/draft-bless-diffserv-lbe-phb-00>>.
- [I-D.ietf-tsvwg-rtcweb-qos]
Jones, P., Dhesikan, S., Jennings, C., and D. Druta, "DSCP Packet Markings for WebRTC QoS", draft-ietf-tsvwg-rtcweb-qos-18 (work in progress), August 2016.
- [ietf99-secchi]
Secchi, R., Venne, A., and A. Custura, "Measurements concerning the DSCP for a LE PHB", Presentation held at 99th IETF Meeting, TSVWG, Prague, July 2017, <<https://datatracker.ietf.org/meeting/99/materials/slides-99-tsvwg-sessb-31measurements-concerning-the-dscp-for-a-le-phb-00>>.
- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, DOI 10.17487/RFC2914, September 2000, <<https://www.rfc-editor.org/info/rfc2914>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<http://www.rfc-editor.org/info/rfc3168>>.
- [RFC3439] Bush, R. and D. Meyer, "Some Internet Architectural Guidelines and Philosophy", RFC 3439, DOI 10.17487/RFC3439, December 2002, <<https://www.rfc-editor.org/info/rfc3439>>.
- [RFC3662] Bless, R., Nichols, K., and K. Wehrle, "A Lower Effort Per-Domain Behavior (PDB) for Differentiated Services", RFC 3662, DOI 10.17487/RFC3662, December 2003, <<http://www.rfc-editor.org/info/rfc3662>>.
- [RFC3754] Bless, R. and K. Wehrle, "IP Multicast in Differentiated Services (DS) Networks", RFC 3754, DOI 10.17487/RFC3754, April 2004, <<http://www.rfc-editor.org/info/rfc3754>>.
- [RFC4594] Babiarz, J., Chan, K., and F. Baker, "Configuration Guidelines for DiffServ Service Classes", RFC 4594, DOI 10.17487/RFC4594, August 2006, <<http://www.rfc-editor.org/info/rfc4594>>.

- [RFC5053] Luby, M., Shokrollahi, A., Watson, M., and T. Stockhammer, "Raptor Forward Error Correction Scheme for Object Delivery", RFC 5053, DOI 10.17487/RFC5053, October 2007, <<https://www.rfc-editor.org/info/rfc5053>>.
- [RFC6297] Welzl, M. and D. Ros, "A Survey of Lower-than-Best-Effort Transport Protocols", RFC 6297, DOI 10.17487/RFC6297, June 2011, <<http://www.rfc-editor.org/info/rfc6297>>.
- [RFC6817] Shalunov, S., Hazel, G., Iyengar, J., and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)", RFC 6817, DOI 10.17487/RFC6817, December 2012, <<http://www.rfc-editor.org/info/rfc6817>>.
- [RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <<https://www.rfc-editor.org/info/rfc6973>>.
- [RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <<https://www.rfc-editor.org/info/rfc8085>>.
- [RFC8087] Fairhurst, G. and M. Welzl, "The Benefits of Using Explicit Congestion Notification (ECN)", RFC 8087, DOI 10.17487/RFC8087, March 2017, <<https://www.rfc-editor.org/info/rfc8087>>.
- [RFC8100] Geib, R., Ed. and D. Black, "Diffserv-Interconnection Classes and Practice", RFC 8100, DOI 10.17487/RFC8100, March 2017, <<http://www.rfc-editor.org/info/rfc8100>>.
- [RFC8325] Szigeti, T., Henry, J., and F. Baker, "Mapping Diffserv to IEEE 802.11", RFC 8325, DOI 10.17487/RFC8325, February 2018, <<https://www.rfc-editor.org/info/rfc8325>>.
- [RFC8436] Fairhurst, G., "Update to IANA Registration Procedures for Pool 3 Values in the Differentiated Services Field Codepoints (DSCP) Registry", RFC 8436, DOI 10.17487/RFC8436, August 2018, <<https://www.rfc-editor.org/info/rfc8436>>.

Appendix A. History of the LE PHB

A first version of this PHB was suggested by Roland Bless and Klaus Wehrle in September 1999 [draft-bless-diffserv-lbe-phb-00], named "A Lower Than Best-Effort Per-Hop Behavior". After some discussion in

the Diffserv Working Group Brian Carpenter and Kathie Nichols proposed a "bulk handling" per-domain behavior and believed a PHB was not necessary. Eventually, "Lower Effort" was specified as per-domain behavior and finally became [RFC3662]. More detailed information about its history can be found in Section 10 of [RFC3662].

There are several other names in use for this type of PHB or associated service classes. Well-known is the QBone Scavenger Service (QBSS) that was proposed in March 2001 within the Internet2 QoS Working Group. Alternative names are "Lower-than-best-effort" [carlberg-lbe-2001] or "Less-than-best-effort" [chown-lbe-2003].

Appendix B. Acknowledgments

Since text is partially borrowed from earlier Internet-Drafts and RFCs the co-authors of previous specifications are acknowledged here: Kathie Nichols and Klaus Wehrle. David Black, Olivier Bonaventure, Spencer Dawkins, Toerless Eckert, Gorrry Fairhurst, Ruediger Geib, and Kyle Rose provided helpful comments and (partially also text) suggestions.

Appendix C. Change History

This section briefly lists changes between Internet-Draft versions for convenience.

Changes in Version 10: (incorporated comments from IESG discussion as follows)

- o Appended "for Differentiated Services" to the title as suggested by Alexey.
- o Addressed Deborah Brungard's discuss: changed phrase to "However, non-LE traffic (e.g., BE traffic) SHOULD NOT be remarked to LE." with additional explanation as suggested by Gorrry.
- o Fixed the sentence "An LE PHB SHOULD NOT be used for a customer's "normal Internet" traffic nor should packets be "downgraded" to the LE PHB instead of being dropped, particularly when the packets are unauthorized traffic." according to Alice's and Mirja's comments.
- o Made reference to RFC8174 normative.
- o Added hint for the RFC editor to apply changes from section Section 12 and to delete it afterwards.

- o Incorporated Mirja's and Benjamin's suggestions.
- o Editorial suggested by Gorrry: In case => In the case that

Changes in Version 09:

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

Changes in Version 08:

- o revised two sentences as suggested by Spencer Dawkins

Changes in Version 07:

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

Changes in Version 06:

- o added Multicast Considerations section with input from Toerless Eckert
- o incorporated suggestions by David Black with respect to better reflect legacy CS1 handling

Changes in Version 05:

- o added scavenger service class into abstract
- o added some more history
- o added reference for "Myth of Over-Provisioning" in RFC3439 and references to presentations w.r.t. codepoint choices
- o added text to update draft-ietf-tsvwg-rtcweb-qos
- o revised text on congestion control in case of remarking to BE
- o added reference to DSCP measurement talk @IETF99
- o small typo fixes

Changes in Version 04:

- o Several editorial changes according to review from Gorrry Fairhurst
- o Changed the section structure a bit (moved subsections 1.1 and 1.2 into own sections 3 and 7 respectively)
- o updated section 2 on requirements language
- o added updates to RFC 8325
- o tried to be more explicit what changes are required to RFCs 4594 and 8325

Changes in Version 03:

- o Changed recommended codepoint to 000001
- o Added text to explain the reasons for the DSCP choice
- o Removed LE-min,LE-strict discussion
- o Added one more potential use case: reporting errors or telemetry data from OSs
- o Added privacy considerations to the security section (not worth an own section I think)
- o Changed IANA considerations section

Changes in Version 02:

- o Applied many editorial suggestions from David Black
- o Added Multicast traffic use case
- o Clarified what is required for deployment in section 1.2 (Deployment Considerations)
- o Added text about implementations using AQMs and ECN usage
- o Updated IANA section according to David Black's suggestions
- o Revised text in the security section
- o Changed copyright Notice to pre5378Trust200902

Changes in Version 01:

- o Now obsoletes RFC 3662.
- o Tried to be more precise in section 1.1 (Applicability) according to R. Geib's suggestions, so rephrased several paragraphs. Added text about congestion control
- o Change section 2 (PHB Description) according to R. Geib's suggestions.
- o Added RFC 2119 language to several sentences.
- o Detailed the description of remarking implications and recommendations in Section 8.
- o Added Section 10 to explicitly list changes with respect to RFC 4594, because this document will update it.

Appendix D. Note to RFC Editor

This section lists actions for the RFC editor during final formatting.

- o Apply the suggested changes of section Section 12 and add a normative reference in draft-ietf-tsvwg-rtcweb-qos to this RFC.
- o Delete Section 12.
- o Please replace the occurrences of RFCXXXX in Section 10 and Section 11 with the assigned RFC number for this document.
- o Delete Appendix C.
- o Delete this section.

Author's Address

Roland Bless
Karlsruhe Institute of Technology (KIT)
Kaiserstr. 12
Karlsruhe 76131
Germany

Phone: +49 721 608 46413
Email: roland.bless@kit.edu

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 28 April 2022

R. R. Stewart
Netflix, Inc.
M. Tüxen
I. Rüngeler
Münster Univ. of Appl. Sciences
25 October 2021

Stream Control Transmission Protocol (SCTP) Network Address Translation
Support
draft-ietf-tsvwg-natsupp-23

Abstract

The Stream Control Transmission Protocol (SCTP) provides a reliable communications channel between two end-hosts in many ways similar to the Transmission Control Protocol (TCP). With the widespread deployment of Network Address Translators (NAT), specialized code has been added to NAT functions for TCP that allows multiple hosts to reside behind a NAT function and yet share a single IPv4 address, even when two hosts (behind a NAT function) choose the same port numbers for their connection. This additional code is sometimes classified as Network Address and Port Translation (NAPT).

This document describes the protocol extensions needed for the SCTP endpoints and the mechanisms for NAT functions necessary to provide similar features of NAPT in the single point and multipoint traversal scenario.

Finally, a YANG module for SCTP NAT is defined.

Status of This Memo

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

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

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

This Internet-Draft will expire on 28 April 2022.

Copyright Notice

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

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

Table of Contents

1. Introduction	3
2. Conventions	5
3. Terminology	5
4. Motivation and Overview	6
4.1. SCTP NAT Traversal Scenarios	6
4.1.1. Single Point Traversal	7
4.1.2. Multipoint Traversal	7
4.2. Limitations of Classical NAPT for SCTP	8
4.3. The SCTP-Specific Variant of NAT	8
5. Data Formats	13
5.1. Modified Chunks	13
5.1.1. Extended ABORT Chunk	13
5.1.2. Extended ERROR Chunk	14
5.2. New Error Causes	14
5.2.1. VTag and Port Number Collision Error Cause	14
5.2.2. Missing State Error Cause	15
5.2.3. Port Number Collision Error Cause	15
5.3. New Parameters	16
5.3.1. Disable Restart Parameter	16
5.3.2. VTags Parameter	17
6. Procedures for SCTP Endpoints and NAT Functions	18
6.1. Association Setup Considerations for Endpoints	19
6.2. Handling of Internal Port Number and Verification Tag Collisions	19
6.2.1. NAT Function Considerations	19
6.2.2. Endpoint Considerations	20
6.3. Handling of Internal Port Number Collisions	20
6.3.1. NAT Function Considerations	20
6.3.2. Endpoint Considerations	21
6.4. Handling of Missing State	21
6.4.1. NAT Function Considerations	22
6.4.2. Endpoint Considerations	22

6.5.	Handling of Fragmented SCTP Packets by NAT Functions . .	24
6.6.	Multi Point Traversal Considerations for Endpoints . . .	24
7.	SCTP NAT YANG Module	24
7.1.	Tree Structure	24
7.2.	YANG Module	25
8.	Various Examples of NAT Traversals	27
8.1.	Single-homed Client to Single-homed Server	28
8.2.	Single-homed Client to Multi-homed Server	30
8.3.	Multihomed Client and Server	32
8.4.	NAT Function Loses Its State	35
8.5.	Peer-to-Peer Communications	37
9.	Socket API Considerations	42
9.1.	Get or Set the NAT Friendliness (SCTP_NAT_FRIENDLY) . . .	43
10.	IANA Considerations	43
10.1.	New Chunk Flags for Two Existing Chunk Types	43
10.2.	Three New Error Causes	45
10.3.	Two New Chunk Parameter Types	46
10.4.	One New URI	46
10.5.	One New YANG Module	46
11.	Security Considerations	46
12.	Normative References	47
13.	Informative References	48
	Acknowledgments	51
	Authors' Addresses	51

1. Introduction

Stream Control Transmission Protocol (SCTP) [RFC4960] provides a reliable communications channel between two end-hosts in many ways similar to TCP [RFC0793]. With the widespread deployment of Network Address Translators (NAT), specialized code has been added to NAT functions for TCP that allows multiple hosts to reside behind a NAT function using private-use addresses (see [RFC6890]) and yet share a single IPv4 address, even when two hosts (behind a NAT function) choose the same port numbers for their connection. This additional code is sometimes classified as Network Address and Port Translation (NAPT). Please note that this document focuses on the case where the NAT function maps a single or multiple internal addresses to a single external address and vice versa.

To date, specialized code for SCTP has not yet been added to most NAT functions so that only a translation of IP addresses is supported. The end result of this is that only one SCTP-capable host can successfully operate behind such a NAT function and this host can only be single-homed. The only alternative for supporting legacy NAT functions is to use UDP encapsulation as specified in [RFC6951].

The NAT function in the document refers to NAPT functions described in Section 2.2 of [RFC3022], NAT64 [RFC6146], or DS-Lite AFTR [RFC6333].

This document specifies procedures allowing a NAT function to support SCTP by providing similar features to those provided by a NAPT for TCP (see [RFC5382] and [RFC7857]), UDP (see [RFC4787] and [RFC7857]), and ICMP (see [RFC5508] and [RFC7857]). This document also specifies a set of data formats for SCTP packets and a set of SCTP endpoint procedures to support NAT traversal. An SCTP implementation supporting these procedures can assure that in both single-homed and multi-homed cases a NAT function will maintain the appropriate state without the NAT function needing to change port numbers.

It is possible and desirable to make these changes for a number of reasons:

- * It is desirable for SCTP internal end-hosts on multiple platforms to be able to share a NAT function's external IP address in the same way that a TCP session can use a NAT function.
- * If a NAT function does not need to change any data within an SCTP packet, it will reduce the processing burden of NAT'ing SCTP by not needing to execute the CRC32c checksum used by SCTP.
- * Not having to touch the IP payload makes the processing of ICMP messages by NAT functions easier.

An SCTP-aware NAT function will need to follow these procedures for generating appropriate SCTP packet formats.

When considering SCTP-aware NAT it is possible to have multiple levels of support. At each level, the Internal Host, Remote Host, and NAT function does or does not support the procedures described in this document. The following table illustrates the results of the various combinations of support and if communications can occur between two endpoints.

Internal Host	NAT Function	Remote Host	Communication
Support	Support	Support	Yes
Support	Support	No Support	Limited
Support	No Support	Support	None
Support	No Support	No Support	None
No Support	Support	Support	Limited
No Support	Support	No Support	Limited
No Support	No Support	Support	None
No Support	No Support	No Support	None

Table 1: Communication possibilities

From the table it can be seen that no communication can occur when a NAT function does not support SCTP-aware NAT. This assumes that the NAT function does not handle SCTP packets at all and all SCTP packets sent from behind a NAT function are discarded by the NAT function. In some cases, where the NAT function supports SCTP-aware NAT, but one of the two hosts does not support the feature, communication can possibly occur in a limited way. For example, only one host can have a connection when a collision case occurs.

2. Conventions

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

3. Terminology

This document uses the following terms, which are depicted in Figure 1. Familiarity with the terminology used in [RFC4960] and [RFC5061] is assumed.

Internal-Address (Int-Addr)

An internal address that is known to the internal host.

Internal-Port (Int-Port)

The port number that is in use by the host holding the Internal-Address.

Internal-VTag (Int-VTag)

The SCTP Verification Tag (VTag) (see Section 3.1 of [RFC4960]) that the internal host has chosen for an association. The VTag is a unique 32-bit tag that accompanies any incoming SCTP packet for this association to the Internal-Address.

Remote-Address (Rem-Addr)

The address that an internal host is attempting to contact.

Remote-Port (Rem-Port)

The port number used by the host holding the Remote-Address.

Remote-VTag (Rem-VTag)

The Verification Tag (VTag) (see Section 3.1 of [RFC4960]) that the host holding the Remote-Address has chosen for an association. The VTag is a unique 32-bit tag that accompanies any outgoing SCTP packet for this association to the Remote-Address.

External-Address (Ext-Addr)

An external address assigned to the NAT function, that it uses as a source address when sending packets towards a Remote-Address.

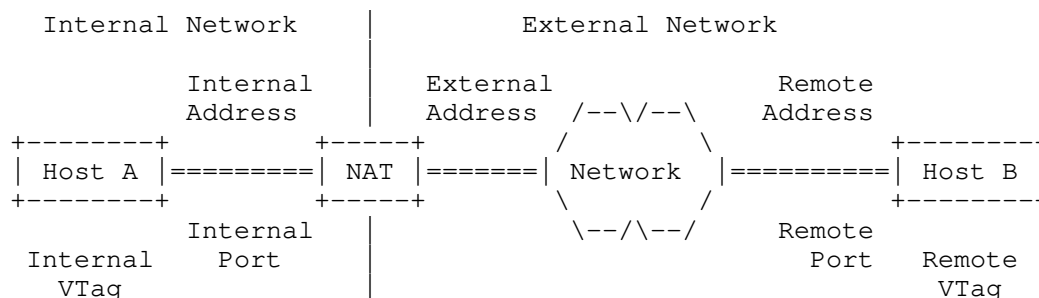


Figure 1: Basic Network Setup

4. Motivation and Overview

4.1. SCTP NAT Traversal Scenarios

This section defines the notion of single and multipoint NAT traversal.

4.1.1. Single Point Traversal

In this case, all packets in the SCTP association go through a single NAT function, as shown in Figure 2.

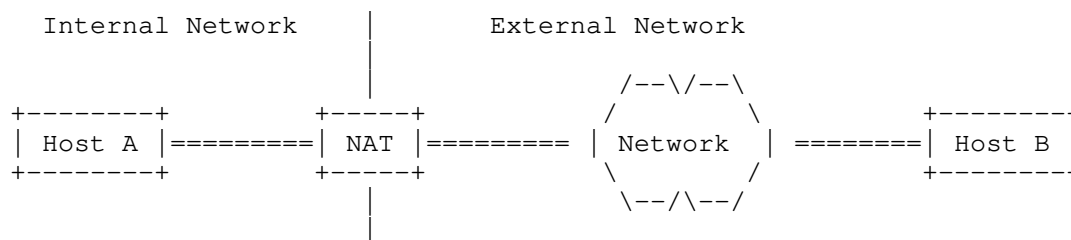


Figure 2: Single NAT Function Scenario

A variation of this case is shown in Figure 3, i.e., multiple NAT functions in the forwarding path between two endpoints.

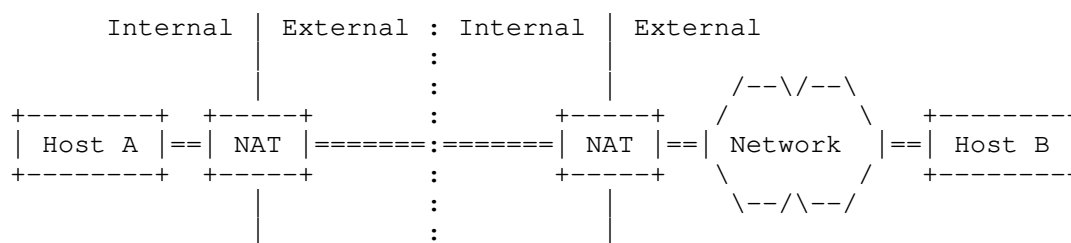


Figure 3: Serial NAT Functions Scenario

Although one of the main benefits of SCTP multi-homing is redundant paths, in the single point traversal scenario the NAT function represents a single point of failure in the path of the SCTP multi-homed association. However, the rest of the path can still benefit from path diversity provided by SCTP multi-homing.

The two SCTP endpoints in this case can be either single-homed or multi-homed. However, the important thing is that the NAT function in this case sees all the packets of the SCTP association.

4.1.2. Multipoint Traversal

This case involves multiple NAT functions and each NAT function only sees some of the packets in the SCTP association. An example is shown in Figure 4.

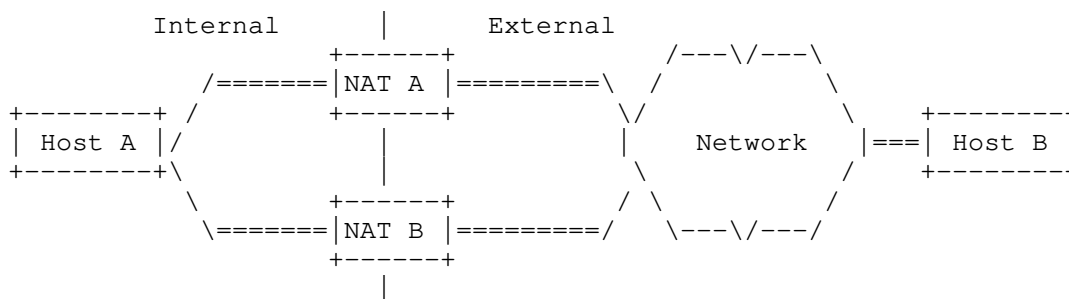


Figure 4: Parallel NAT Functions Scenario

This case does not apply to a single-homed SCTP association (i.e., both endpoints in the association use only one IP address). The advantage here is that the existence of multiple NAT traversal points can preserve the path diversity of a multi-homed association for the entire path. This in turn can improve the robustness of the communication.

4.2. Limitations of Classical NAPT for SCTP

Using classical NAPT possibly results in changing one of the SCTP port numbers during the processing, which requires the recomputation of the transport layer checksum by the NAPT function. Whereas for UDP and TCP this can be done very efficiently, for SCTP the checksum (CRC32c) over the entire packet needs to be recomputed (see Appendix B of [RFC4960] for details of the CRC32c computation). This would considerably add to the NAT computational burden, however hardware support can mitigate this in some implementations.

An SCTP endpoint can have multiple addresses but only has a single port number to use. To make multipoint traversal work, all the NAT functions involved need to recognize the packets they see as belonging to the same SCTP association and perform port number translation in a consistent way. One possible way of doing this is to use a pre-defined table of port numbers and addresses configured within each NAT function. Other mechanisms could make use of NAT to NAT communication. Such mechanisms have not been deployed on a wide scale base and thus are not a preferred solution. Therefore an SCTP variant of NAT function has been developed (see Section 4.3).

4.3. The SCTP-Specific Variant of NAT

In this section it is allowed that there are multiple SCTP capable hosts behind a NAT function that share one External-Address. Furthermore, this section focuses on the single point traversal scenario (see Section 4.1.1).

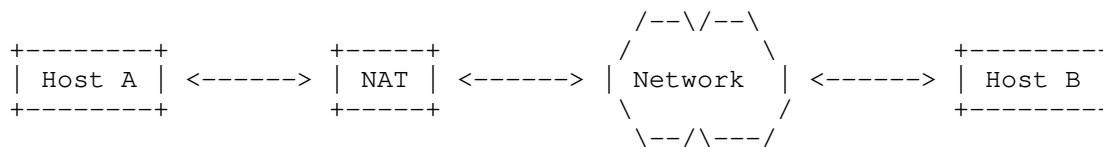
The modification of outgoing SCTP packets sent from an internal host is simple: the source address of the packets has to be replaced with the External-Address. It might also be necessary to establish some state in the NAT function to later handle incoming packets.

Typically, the NAT function has to maintain a NAT binding table of Internal-VTag, Internal-Port, Remote-VTag, Remote-Port, Internal-Address, and whether the restart procedure is disabled or not. An entry in that NAT binding table is called a NAT-State control block. The function Create() obtains the just mentioned parameters and returns a NAT-State control block. A NAT function MAY allow creating NAT-State control blocks via a management interface.

For SCTP packets coming from the external realm of the NAT function the destination address of the packets has to be replaced with the Internal-Address of the host to which the packet has to be delivered, if a NAT state entry is found. The lookup of the Internal-Address is based on the Remote-VTag, Remote-Port, Internal-VTag and the Internal-Port.

The entries in the NAT binding table need to fulfill some uniqueness conditions. There can not be more than one entry NAT binding table with the same pair of Internal-Port and Remote-Port. This rule can be relaxed, if all NAT binding table entries with the same Internal-Port and Remote-Port have the support for the restart procedure disabled (see Section 5.3.1). In this case there can not be no more than one entry with the same Internal-Port, Remote-Port and Remote-VTag and no more than one NAT binding table entry with the same Internal-Port, Remote-Port, and Int-VTag.

The processing of outgoing SCTP packets containing an INIT chunk is illustrated in the following figure. This scenario is valid for all message flows in this section.



```

INIT[Initiate-Tag]
Int-Addr:Int-Port -----> Rem-Addr:Rem-Port
Rem-VTag=0

```

```

Create(Initiate-Tag, Int-Port, 0, Rem-Port, Int-Addr,
      IsRestartDisabled)
Returns(NAT-State control block)

```

Translate To:

```

INIT[Initiate-Tag]
Ext-Addr:Int-Port -----> Rem-Addr:Rem-Port
Rem-VTag=0

```

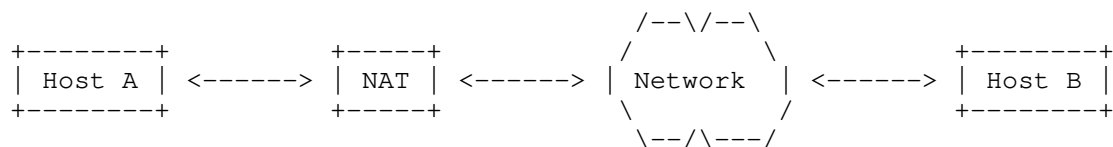
Normally a NAT binding table entry will be created.

However, it is possible that there is already a NAT binding table entry with the same Remote-Port, Internal-Port, and Internal-VTag but different Internal-Address and the restart procedure is disabled. In this case the packet containing the INIT chunk MUST be dropped by the NAT and a packet containing an ABORT chunk SHOULD be sent to the SCTP host that originated the packet with the M bit set and 'VTag and Port Number Collision' error cause (see Section 5.1.1 for the format). The source address of the packet containing the ABORT chunk MUST be the destination address of the packet containing the INIT chunk.

If an outgoing SCTP packet contains an INIT or ASCONF chunk and a matching NAT binding table entry is found, the packet is processed as a normal outgoing packet.

It is also possible that a NAT binding table entry with the same Remote-Port and Internal-Port exists without an Internal-VTag conflict but there exists a NAT binding table entry with the same port numbers but a different Internal-Address and the restart procedure is not disabled. In such a case the packet containing the INIT chunk MUST be dropped by the NAT function and a packet containing an ABORT chunk SHOULD be sent to the SCTP host that originated the packet with the M bit set and 'Port Number Collision' error cause (see Section 5.1.1 for the format).

The processing of outgoing SCTP packets containing no INIT chunks is described in the following figure.

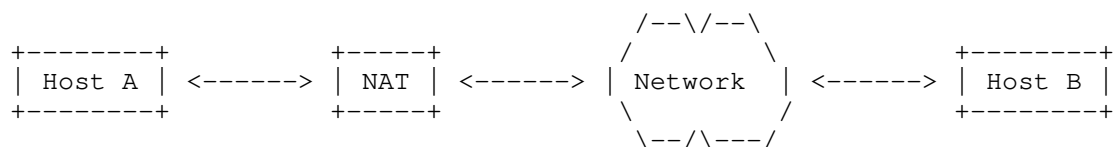


Int-Addr:Int-Port -----> Rem-Addr:Rem-Port
 Rem-VTag

Translate To:

Ext-Addr:Int-Port -----> Rem-Addr:Rem-Port
 Rem-VTag

The processing of incoming SCTP packets containing an INIT ACK chunk is illustrated in the following figure. The Lookup() function has as input the Internal-VTag, Internal-Port, Remote-VTag, and Remote-Port. It returns the corresponding entry of the NAT binding table and updates the Remote-VTag by substituting it with the value of the Initiate-Tag of the INIT ACK chunk. The wildcard character signifies that the parameter's value is not considered in the Lookup() function or changed in the Update() function, respectively.



INIT ACK[Initiate-Tag]
 Ext-Addr:Int-Port <---- Rem-Addr:Rem-Port
 Int-VTag

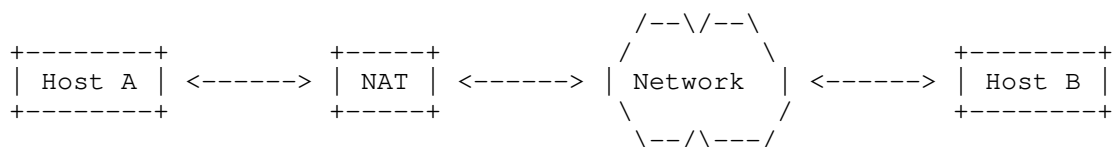
Lookup(Int-VTag, Int-Port, *, Rem-Port)
 Update(*, *, Initiate-Tag, *)

Returns(NAT-State control block containing Int-Addr)

INIT ACK[Initiate-Tag]
 Int-Addr:Int-Port <----- Rem-Addr:Rem-Port
 Int-VTag

In the case where the Lookup function fails because it does not find an entry, the SCTP packet is dropped. If it succeeds, the Update routine inserts the Remote-VTag (the Initiate-Tag of the INIT ACK chunk) in the NAT-State control block.

The processing of incoming SCTP packets containing an ABORT or SHUTDOWN COMPLETE chunk with the T bit set is illustrated in the following figure.



Ext-Addr:Int-Port <----- Rem-Addr:Rem-Port
Rem-VTag

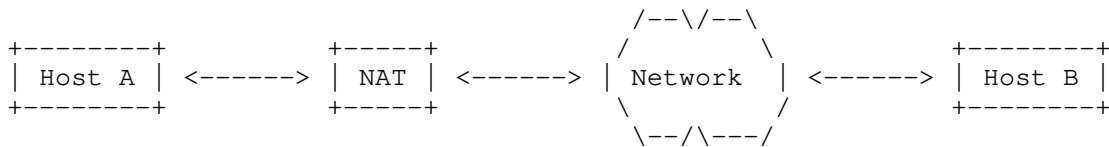
Lookup(*, Int-Port, Rem-VTag, Rem-Port)

Returns (NAT-State control block containing Int-Addr)

Int-Addr:Int-Port <----- Rem-Addr:Rem-Port
Rem-VTag

For an incoming packet containing an INIT chunk a table lookup is made only based on the addresses and port numbers. If an entry with a Remote-VTag of zero is found, it is considered a match and the Remote-VTag is updated. If an entry with a non-matching Remote-VTag is found or no entry is found, the incoming packet is silently dropped. If an entry with a matching Remote-VTag is found, the incoming packet is forwarded. This allows the handling of INIT collision through NAT functions.

The processing of other incoming SCTP packets is described in the following figure.



Ext-Addr: Int-Port <----- Rem-Addr: Rem-Port
Int-VTag

Lookup(Int-VTag, Int-Port, *, Rem-Port)

Returns(NAT-State control block containing Internal-Address)

Int-Addr: Int-Port <----- Rem-Addr: Rem-Port
Int-VTag

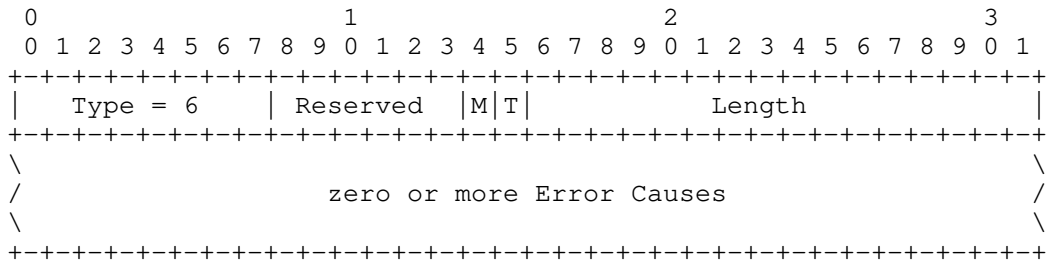
5. Data Formats

This section defines the formats used to support NAT traversal. Section 5.1 and Section 5.2 describe chunks and error causes sent by NAT functions and received by SCTP endpoints. Section 5.3 describes parameters sent by SCTP endpoints and used by NAT functions and SCTP endpoints.

5.1. Modified Chunks

This section presents existing chunks defined in [RFC4960] for which additional flags are specified by this document.

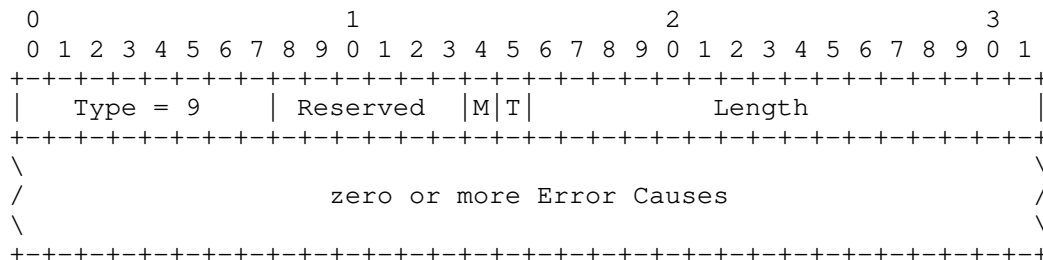
5.1.1. Extended ABORT Chunk



The ABORT chunk is extended to add the new 'M bit'. The M bit indicates to the receiver of the ABORT chunk that the chunk was not generated by the peer SCTP endpoint, but instead by a middle box (e.g., NAT).

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

5.1.2. Extended ERROR Chunk



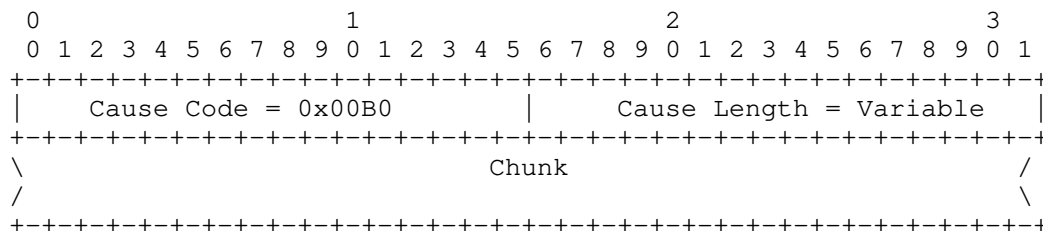
The ERROR chunk defined in [RFC4960] is extended to add the new 'M bit'. The M bit indicates to the receiver of the ERROR chunk that the chunk was not generated by the peer SCTP endpoint, but instead by a middle box.

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

5.2. New Error Causes

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

5.2.1. VTag and Port Number Collision Error Cause



Cause Code: 2 bytes (unsigned integer)

This field holds the IANA defined cause code for the 'VTag and Port Number Collision' Error Cause. IANA is requested to assign the value 0x00B0 for this cause code.

Cause Length: 2 bytes (unsigned integer)

This field holds the length in bytes of the error cause. The value **MUST** be the length of the Cause-Specific Information plus 4.

Chunk: variable length

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

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

5.2.2. Missing State Error Cause

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
Cause Code = 0x00B1										Cause Length = Variable																													
Original Packet																																							

Cause Code: 2 bytes (unsigned integer)

This field holds the IANA defined cause code for the 'Missing State' Error Cause. IANA is requested to assign the value 0x00B1 for this cause code.

Cause Length: 2 bytes (unsigned integer)

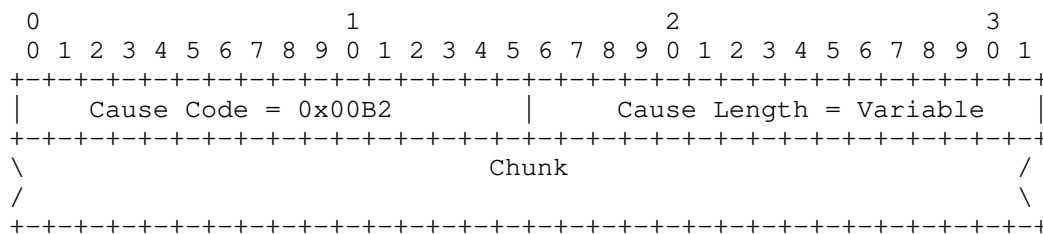
This field holds the length in bytes of the error cause. The value MUST be the length of the Cause-Specific Information plus 4.

Original Packet: variable length

The Cause-Specific Information is filled with the IPv4 or IPv6 packet that caused this error. The IPv4 or IPv6 header MUST be included. Note that if the packet will not fit in the ERROR chunk or ABORT chunk being sent then the bytes that do not fit are truncated.

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

5.2.3. Port Number Collision Error Cause



Cause Code: 2 bytes (unsigned integer)

This field holds the IANA defined cause code for the 'Port Number Collision' Error Cause. IANA is requested to assign the value 0x00B2 for this cause code.

Cause Length: 2 bytes (unsigned integer)

This field holds the length in bytes of the error cause. The value MUST be the length of the Cause-Specific Information plus 4.

Chunk: variable length

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

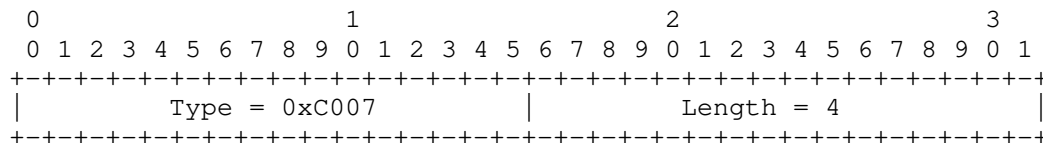
[NOTE to RFC-Editor: Assignment of cause code to be confirmed by IANA.]

5.3. New Parameters

This section defines new parameters and their valid appearance defined by this document.

5.3.1. Disable Restart Parameter

This parameter is used to indicate that the restart procedure is requested to be disabled. Both endpoints of an association MUST include this parameter in the INIT chunk and INIT ACK chunk when establishing an association and MUST include it in the ASCONF chunk when adding an address to successfully disable the restart procedure.



Parameter Type: 2 bytes (unsigned integer)

This field holds the IANA defined parameter type for the Disable Restart Parameter. IANA is requested to assign the value 0xC007 for this parameter type.

Parameter Length: 2 bytes (unsigned integer)

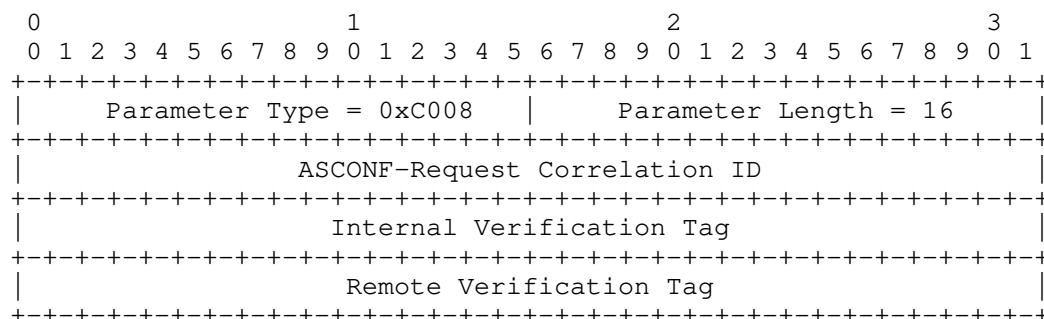
This field holds the length in bytes of the parameter. The value MUST be 4.

[NOTE to RFC-Editor: Assignment of parameter type to be confirmed by IANA.]

The Disable Restart Parameter MAY appear in INIT, INIT ACK and ASCONF chunks and MUST NOT appear in any other chunk.

5.3.2. VTags Parameter

This parameter is used to help a NAT function to recover from state loss.



Parameter Type: 2 bytes (unsigned integer)

This field holds the IANA defined parameter type for the VTags Parameter. IANA is requested to assign the value 0xC008 for this parameter type.

Parameter Length: 2 bytes (unsigned integer)

This field holds the length in bytes of the parameter. The value MUST be 16.

ASCONF-Request Correlation ID: 4 bytes (unsigned integer)

This is an opaque integer assigned by the sender to identify each request parameter. The receiver of the ASCONF Chunk will copy this 32-bit value into the ASCONF Response Correlation ID field of the ASCONF ACK response parameter. The sender of the packet containing the ASCONF chunk can use this same value in the ASCONF ACK chunk to find which request the response is for. The receiver MUST NOT change the value of the ASCONF-Request Correlation ID.

Internal Verification Tag: 4 bytes (unsigned integer)

The Verification Tag that the internal host has chosen for the association. The Verification Tag is a unique 32-bit tag that accompanies any incoming SCTP packet for this association to the Internal-Address.

Remote Verification Tag: 4 bytes (unsigned integer)

The Verification Tag that the host holding the Remote-Address has chosen for the association. The VTag is a unique 32-bit tag that accompanies any outgoing SCTP packet for this association to the Remote-Address.

[NOTE to RFC-Editor: Assignment of parameter type to be confirmed by IANA.]

The VTags Parameter MAY appear in ASCONF chunks and MUST NOT appear in any other chunk.

6. Procedures for SCTP Endpoints and NAT Functions

If an SCTP endpoint is behind an SCTP-aware NAT, a number of problems can arise as it tries to communicate with its peers:

- * IP addresses can not be included in the SCTP packet. This is discussed in Section 6.1.
- * More than one host behind a NAT function could select the same VTag and source port number when communicating with the same peer server. This creates a situation where the NAT function will not be able to tell the two associations apart. This situation is discussed in Section 6.2.
- * If an SCTP endpoint is a server communicating with multiple peers and the peers are behind the same NAT function, then these peers cannot be distinguished by the server. This case is discussed in Section 6.3.
- * A restart of a NAT function during a conversation could cause a loss of its state. This problem and its solution is discussed in Section 6.4.
- * NAT functions need to deal with SCTP packets being fragmented at the IP layer. This is discussed in Section 6.5.
- * An SCTP endpoint can be behind two NAT functions in parallel providing redundancy. The method to set up this scenario is discussed in Section 6.6.

The mechanisms to solve these problems require additional chunks and parameters, defined in this document, and modified handling procedures from those specified in [RFC4960] as described below.

6.1. Association Setup Considerations for Endpoints

The association setup procedure defined in [RFC4960] allows multi-homed SCTP endpoints to exchange its IP-addresses by using IPv4 or IPv6 address parameters in the INIT and INIT ACK chunks. However, this does not work when NAT functions are present.

Every association setup from a host behind a NAT function MUST NOT use multiple internal addresses. The INIT chunk MUST NOT contain an IPv4 Address parameter, IPv6 Address parameter, or Supported Address Types parameter. The INIT ACK chunk MUST NOT contain any IPv4 Address parameter or IPv6 Address parameter using non-global addresses. The INIT chunk and the INIT ACK chunk MUST NOT contain any Host Name parameters.

If the association is intended to be finally multi-homed, the procedure in Section 6.6 MUST be used.

The INIT and INIT ACK chunk SHOULD contain the Disable Restart parameter defined in Section 5.3.1.

6.2. Handling of Internal Port Number and Verification Tag Collisions

Consider the case where two hosts in the Internal-Address space want to set up an SCTP association with the same service provided by some remote hosts. This means that the Remote-Port is the same. If they both choose the same Internal-Port and Internal-VTag, the NAT function cannot distinguish between incoming packets anymore. However, this is unlikely. The Internal-VTags are chosen at random and if the Internal-Ports are also chosen from the ephemeral port range at random (see [RFC6056]) this gives a 46-bit random number that has to match.

The same can happen with the Remote-VTag when a packet containing an INIT ACK chunk or an ASCONF chunk is processed by the NAT function.

6.2.1. NAT Function Considerations

If the NAT function detects a collision of internal port numbers and verification tags, it SHOULD send a packet containing an ABORT chunk with the M bit set if the collision is triggered by a packet containing an INIT or INIT ACK chunk. If such a collision is triggered by a packet containing an ASCONF chunk, it SHOULD send a packet containing an ERROR chunk with the M bit. The M bit is a new

bit defined by this document to express to SCTP that the source of this packet is a "middle" box, not the peer SCTP endpoint (see Section 5.1.1). If a packet containing an INIT ACK chunk triggers the collision, the corresponding packet containing the ABORT chunk MUST contain the same source and destination address and port numbers as the packet containing the INIT ACK chunk. If a packet containing an INIT chunk or an ASCONF chunk, the source and destination address and port numbers MUST be swapped.

The sender of the packet containing an ERROR or ABORT chunk MUST include the error cause with cause code 'VTag and Port Number Collision' (see Section 5.2.1).

6.2.2. Endpoint Considerations

The sender of the packet containing the INIT chunk or the receiver of a packet containing the INIT ACK chunk, upon reception of a packet containing an ABORT chunk with M bit set and the appropriate error cause code for colliding NAT binding table state is included, SHOULD reinitiate the association setup procedure after choosing a new initiate tag, if the association is in COOKIE-WAIT state. In any other state, the SCTP endpoint MUST NOT respond.

The sender of the packet containing the ASCONF chunk, upon reception of a packet containing an ERROR chunk with M bit set, MUST stop adding the path to the association.

6.3. Handling of Internal Port Number Collisions

When two SCTP hosts are behind an SCTP-aware NAT it is possible that two SCTP hosts in the Internal-Address space will want to set up an SCTP association with the same server running on the same remote host. If the two hosts choose the same internal port, this is considered an internal port number collision.

For the NAT function, appropriate tracking can be performed by assuring that the VTags are unique between the two hosts.

6.3.1. NAT Function Considerations

The NAT function, when processing the packet containing the INIT ACK chunk, SHOULD note in its NAT binding table if the association supports the disable restart extension. This note is used when establishing future associations (i.e. when processing a packet containing an INIT chunk from an internal host) to decide if the connection can be allowed. The NAT function does the following when processing a packet containing an INIT chunk:

- * If the packet containing the INIT chunk is originating from an internal port to a remote port for which the NAT function has no matching NAT binding table entry, it MUST allow the packet containing the INIT chunk creating an NAT binding table entry.
- * If the packet containing the INIT chunk matches an existing NAT binding table entry, it MUST validate that the disable restart feature is supported and, if it does, allow the packet containing the INIT chunk to be forwarded.
- * If the disable restart feature is not supported, the NAT function SHOULD send a packet containing an ABORT chunk with the M bit set.

The 'Port Number Collision' error cause (see Section 5.2.3) MUST be included in the ABORT chunk sent in response to the packet containing an INIT chunk.

If the collision is triggered by a packet containing an ASCONF chunk, a packet containing an ERROR chunk with the 'Port Number Collision' error cause SHOULD be sent in response to the packet containing the ASCONF chunk.

6.3.2. Endpoint Considerations

For the remote SCTP server this means that the Remote-Port and the Remote-Address are the same. If they both have chosen the same Internal-Port the server cannot distinguish between both associations based on the address and port numbers. For the server it looks like the association is being restarted. To overcome this limitation the client sends a Disable Restart parameter in the INIT chunk.

When the server receives this parameter it does the following:

- * It MUST include a Disable Restart parameter in the INIT ACK to inform the client that it will support the feature.
- * It MUST disable the restart procedures defined in [RFC4960] for this association.

Servers that support this feature will need to be capable of maintaining multiple connections to what appears to be the same peer (behind the NAT function) differentiated only by the VTags.

6.4. Handling of Missing State

6.4.1. NAT Function Considerations

If the NAT function receives a packet from the internal network for which the lookup procedure does not find an entry in the NAT binding table, a packet containing an ERROR chunk SHOULD be sent back with the M bit set. The source address of the packet containing the ERROR chunk MUST be the destination address of the packet received from the internal network. The verification tag is reflected and the T bit is set. Such a packet containing an ERROR chunk SHOULD NOT be sent if the received packet contains an ASCONF chunk with the VTags parameter or an ABORT, SHUTDOWN COMPLETE or INIT ACK chunk. A packet containing an ERROR chunk MUST NOT be sent if the received packet contains an ERROR chunk with the M bit set. In any case, the packet SHOULD NOT be forwarded to the remote address.

If the NAT function receives a packet from the internal network for which it has no NAT binding table entry and the packet contains an ASCONF chunk with the VTags parameter, the NAT function MUST update its NAT binding table according to the verification tags in the VTags parameter and, if present, the Disable Restart parameter.

When sending a packet containing an ERROR chunk, the error cause 'Missing State' (see Section 5.2.2) MUST be included and the M bit of the ERROR chunk MUST be set (see Section 5.1.2).

6.4.2. Endpoint Considerations

Upon reception of this packet containing the ERROR chunk by an SCTP endpoint the receiver takes the following actions:

- * It SHOULD validate that the verification tag is reflected by looking at the VTag that would have been included in an outgoing packet. If the validation fails, discard the received packet containing the ERROR chunk.
- * It SHOULD validate that the peer of the SCTP association supports the dynamic address extension. If the validation fails, discard the received packet containing the ERROR chunk.
- * It SHOULD generate a packet containing a new ASCONF chunk containing the VTags parameter (see Section 5.3.2) and the Disable Restart parameter (see Section 5.3.1) if the association is using the disable restart feature. By processing this packet the NAT function can recover the appropriate state. The procedures for generating an ASCONF chunk can be found in [RFC5061].

The peer SCTP endpoint receiving such a packet containing an ASCONF chunk SHOULD add the address and respond with an acknowledgment if the address is new to the association (following all procedures defined in [RFC5061]). If the address is already part of the association, the SCTP endpoint MUST NOT respond with an error, but instead SHOULD respond with a packet containing an ASCONF ACK chunk acknowledging the address and take no action (since the address is already in the association).

Note that it is possible that upon receiving a packet containing an ASCONF chunk containing the VTags parameter the NAT function will realize that it has an 'Internal Port Number and Verification Tag collision'. In such a case the NAT function SHOULD send a packet containing an ERROR chunk with the error cause code set to 'VTag and Port Number Collision' (see Section 5.2.1).

If an SCTP endpoint receives a packet containing an ERROR chunk with 'Internal Port Number and Verification Tag collision' as the error cause and the packet in the Error Chunk contains an ASCONF with the VTags parameter, careful examination of the association is necessary. The endpoint does the following:

- * It MUST validate that the verification tag is reflected by looking at the VTag that would have been included in the outgoing packet. If the validation fails, it MUST discard the packet.
- * It MUST validate that the peer of the SCTP association supports the dynamic address extension. If the peer does not support this extension, it MUST discard the received packet containing the ERROR chunk.
- * If the association is attempting to add an address (i.e. following the procedures in Section 6.6) then the endpoint MUST NOT consider the address part of the association and SHOULD make no further attempt to add the address (i.e. cancel any ASCONF timers and remove any record of the path), since the NAT function has a VTag collision and the association cannot easily create a new VTag (as it would if the error occurred when sending a packet containing an INIT chunk).
- * If the endpoint has no other path, i.e. the procedure was executed due to missing a state in the NAT function, then the endpoint MUST abort the association. This would occur only if the local NAT function restarted and accepted a new association before attempting to repair the missing state (Note that this is no different than what happens to all TCP connections when a NAT function loses its state).

6.5. Handling of Fragmented SCTP Packets by NAT Functions

SCTP minimizes the use of IP-level fragmentation. However, it can happen that using IP-level fragmentation is needed to continue an SCTP association. For example, if the path MTU is reduced and there are still some DATA chunk in flight, which require packets larger than the new path MTU. If IP-level fragmentation can not be used, the SCTP association will be terminated in a non-graceful way. See [RFC8900] for more information about IP fragmentation.

Therefore, a NAT function MUST be able to handle IP-level fragmented SCTP packets. The fragments MAY arrive in any order.

When an SCTP packet can not be forwarded by the NAT function due to MTU issues and the IP header forbids fragmentation, the NAT MUST send back a "Fragmentation needed and DF set" ICMPv4 or PTB ICMPv6 message to the internal host. This allows for a faster recovery from this packet drop.

6.6. Multi Point Traversal Considerations for Endpoints

If a multi-homed SCTP endpoint behind a NAT function connects to a peer, it MUST first set up the association single-homed with only one address causing the first NAT function to populate its state. Then it SHOULD add each IP address using packets containing ASCONF chunks sent via their respective NAT functions. The address used in the Add IP address parameter is the wildcard address (0.0.0.0 or ::0) and the address parameter in the ASCONF chunk SHOULD also contain the VTags parameter and optionally the Disable Restart parameter.

7. SCTP NAT YANG Module

This section defines a YANG module for SCTP NAT.

The terminology for describing YANG data models is defined in [RFC7950]. The meaning of the symbols in tree diagrams is defined in [RFC8340].

7.1. Tree Structure

This module augments NAT YANG module [RFC8512] with SCTP specifics. The module supports both classical SCTP NAT (that is, rewrite port numbers) and SCTP-specific variant where the ports numbers are not altered. The YANG "feature" is used to indicate whether SCTP-specific variant is supported.

The tree structure of the SCTP NAT YANG module is provided below:

```

module: ietf-nat-sctp
  augment /nat:nat/nat:instances/nat:instance
    /nat:policy/nat:timers:
      +--rw sctp-timeout?  uint32
  augment /nat:nat/nat:instances/nat:instance
    /nat:mapping-table/nat:mapping-entry:
      +--rw int-VTag?      uint32 {sctp-nat}?
      +--rw rem-VTag?      uint32 {sctp-nat}?

```

Concretely, the SCTP NAT YANG module augments the NAT YANG module (policy, in particular) with the following:

- * The sctp-timeout is used to control the SCTP inactivity timeout. That is, the time an SCTP mapping will stay active without SCTP packets traversing the NAT. This timeout can be set only for SCTP. Hence, `"/nat:nat/nat:instances/nat:instance/nat:policy/nat:transport-protocols/nat:protocol-id"` MUST be set to `'132'` (SCTP).

In addition, the SCTP NAT YANG module augments the mapping entry with the following parameters defined in Section 3. These parameters apply only for SCTP NAT mapping entries (i.e., `"/nat/instances/instance/mapping-table/mapping-entry/transport-protocol"` MUST be set to `'132'`);

- * The Internal Verification Tag (Int-VTag)
- * The Remote Verification Tag (Rem-VTag)

7.2. YANG Module

```

<CODE BEGINS> file "ietf-nat-sctp@2020-11-02.yang"
module ietf-nat-sctp {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-nat-sctp";
  prefix nat-sctp;

  import ietf-nat {
    prefix nat;
    reference
      "RFC 8512: A YANG Module for Network Address Translation
       (NAT) and Network Prefix Translation (NPT)";
  }

  organization
    "IETF TSVWG Working Group";
  contact
    "WG Web:  <https://datatracker.ietf.org/wg/tsvwg/>

```

WG List: <mailto:tsvwg@ietf.org>

Author: Mohamed Boucadair
<mailto:mohamed.boucadair@orange.com>;

description

"This module augments NAT YANG module with Stream Control Transmission Protocol (SCTP) specifics. The extension supports both a classical SCTP NAT (that is, rewrite port numbers) and a, SCTP-specific variant where the ports numbers are not altered.

Copyright (c) 2020 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

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

revision 2019-11-18 {

description

"Initial revision.";

reference

"RFC XXXX: Stream Control Transmission Protocol (SCTP) Network Address Translation Support";

}

feature sctp-nat {

description

"This feature means that SCTP-specific variant of NAT is supported. That is, avoid rewriting port numbers.";

reference

"Section 4.3 of RFC XXXX.";

}

augment "/nat:nat/nat:instances/nat:instance"

+ "/nat:policy/nat:timers" {

when "/nat:nat/nat:instances/nat:instance"

+ "/nat:policy/nat:transport-protocols"

+ "/nat:protocol-id = 132";

description

"Extends NAT policy with a timeout for SCTP mapping entries.";

```
    leaf sctp-timeout {
      type uint32;
      units "seconds";
      description
        "SCTP inactivity timeout. That is, the time an SCTP
        mapping entry will stay active without packets
        traversing the NAT.";
    }
  }

  augment "/nat:nat/nat:instances/nat:instance"
    + "/nat:mapping-table/nat:mapping-entry" {
    when "nat:transport-protocol = 132";
    if-feature "sctp-nat";
    description
      "Extends the mapping entry with SCTP specifics.";

    leaf int-VTag {
      type uint32;
      description
        "The Internal Verification Tag that the internal
        host has chosen for this communication.";
    }
    leaf rem-VTag {
      type uint32;
      description
        "The Remote Verification Tag that the remote
        peer has chosen for this communication.";
    }
  }
}
<CODE ENDS>
```

8. Various Examples of NAT Traversals

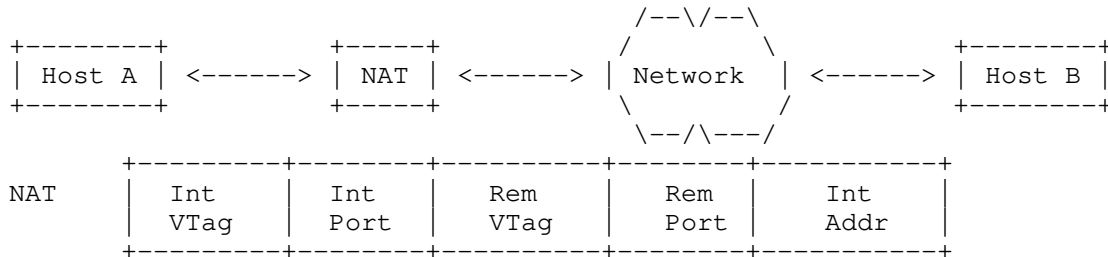
Please note that this section is informational only.

The addresses being used in the following examples are IPv4 addresses for private-use networks and for documentation as specified in [RFC6890]. However, the method described here is not limited to this NAT44 case.

The NAT binding table entries shown in the following examples do not include the flag indicating whether the restart procedure is supported or not. This flag is not relevant for these examples.

8.1. Single-homed Client to Single-homed Server

The internal client starts the association with the remote server via a four-way-handshake. Host A starts by sending a packet containing an INIT chunk.



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

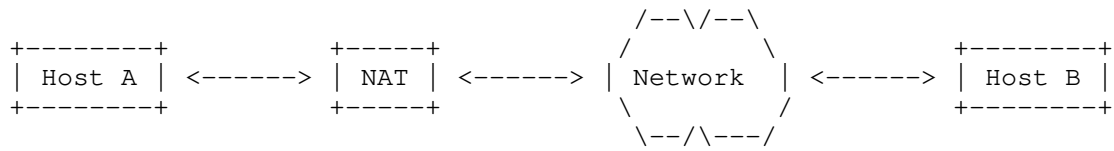
A NAT binding tabled entry is created, the source address is substituted and the packet is sent on:

NAT function creates entry:

NAT	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	0	2	10.0.0.1

```
INIT[Initiate-Tag = 1234]
192.0.2.1:1 -----> 203.0.113.1:2
    Rem-VTtag = 0
```

Host B receives the packet containing an INIT chunk and sends a packet containing an INIT ACK chunk with the NAT's Remote-address as destination address.



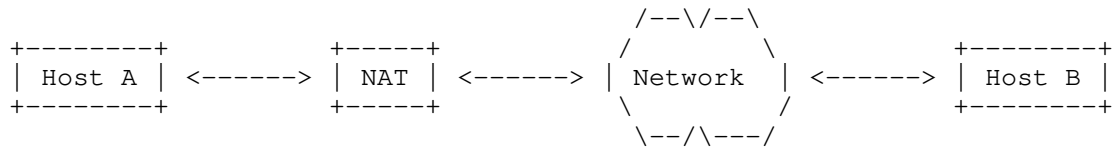
INIT ACK[Initiate-Tag = 5678]
 192.0.2.1:1 <----- 203.0.113.1:2
 Int-VTag = 1234

NAT function updates entry:

NAT	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	5678	2	10.0.0.1

INIT ACK[Initiate-Tag = 5678]
 10.0.0.1:1 <----- 203.0.113.1:2
 Int-VTag = 1234

The handshake finishes with a COOKIE ECHO acknowledged by a COOKIE ACK.



COOKIE ECHO
 10.0.0.1:1 -----> 203.0.113.1:2
 Rem-VTag = 5678

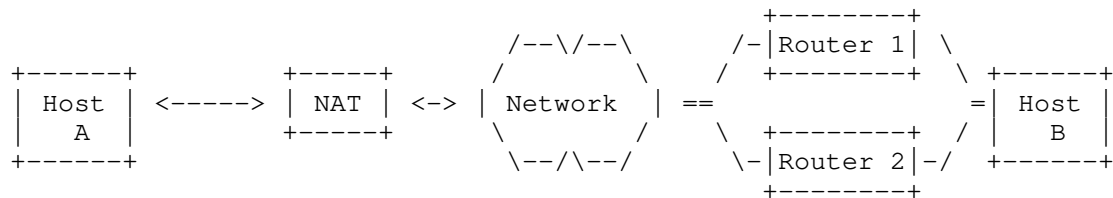
COOKIE ECHO
 192.0.2.1:1 -----> 203.0.113.1:2
 Rem-VTag = 5678

COOKIE ACK
 192.0.2.1:1 <----- 203.0.113.1:2
 Int-VTag = 1234

COOKIE ACK
 10.0.0.1:1 <----- 203.0.113.1:2
 Int-VTag = 1234

8.2. Single-homed Client to Multi-homed Server

The internal client is single-homed whereas the remote server is multi-homed. The client (Host A) sends a packet containing an INIT chunk like in the single-homed case.



NAT					
	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr

```

INIT[Initiate-Tag = 1234]
10.0.0.1:1 ---> 203.0.113.1:2
    Rem-VTag = 0
  
```

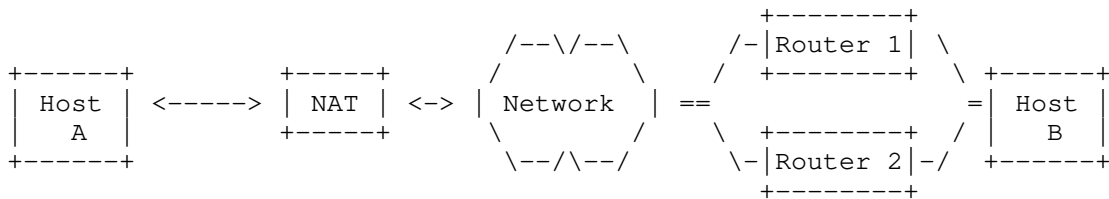
NAT function creates entry:

NAT					
	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	0	2	10.0.0.1

```

                                INIT[Initiate-Tag = 1234]
192.0.2.1:1 -----> 203.0.113.1:2
                        Rem-VTag = 0
  
```

The server (Host B) includes its two addresses in the INIT ACK chunk.



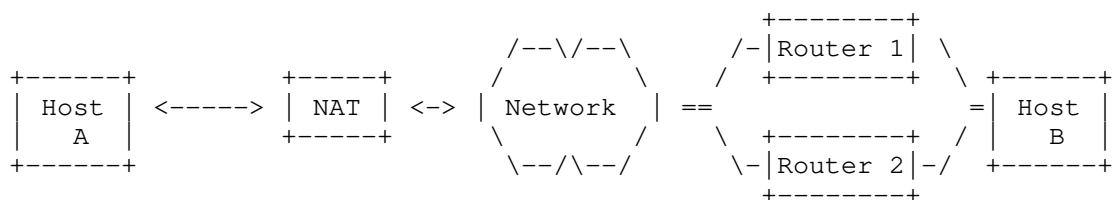
```
INIT ACK[Initiate-tag = 5678, IP-Addr = 203.0.113.129]
192.0.2.1:1 <----- 203.0.113.1:2
Int-VTag = 1234
```

The NAT function does not need to change the NAT binding table for the second address:

NAT	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	5678	2	10.0.0.1

```
INIT ACK[Initiate-Tag = 5678]
10.0.0.1:1 <--- 203.0.113.1:2
Int-VTag = 1234
```

The handshake finishes with a COOKIE ECHO acknowledged by a COOKIE ACK.



COOKIE ECHO
 10.0.0.1:1 ---> 203.0.113.1:2
 Rem-VTag = 5678

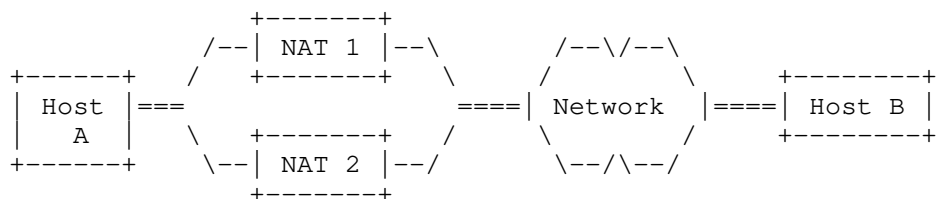
COOKIE ECHO
 192.0.2.1:1 -----> 203.0.113.1:2
 Rem-VTag = 5678

COOKIE ACK
 192.0.2.1:1 <----- 203.0.113.1:2
 Int-VTag = 1234

COOKIE ACK
 10.0.0.1:1 <--- 203.0.113.1:2
 Int-VTag = 1234

8.3. Multihomed Client and Server

The client (Host A) sends a packet containing an INIT chunk to the server (Host B), but does not include the second address.



NAT 1	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr

INIT[Initiate-Tag = 1234]
 10.0.0.1:1 -----> 203.0.113.1:2
 Rem-VTag = 0

NAT function 1 creates entry:

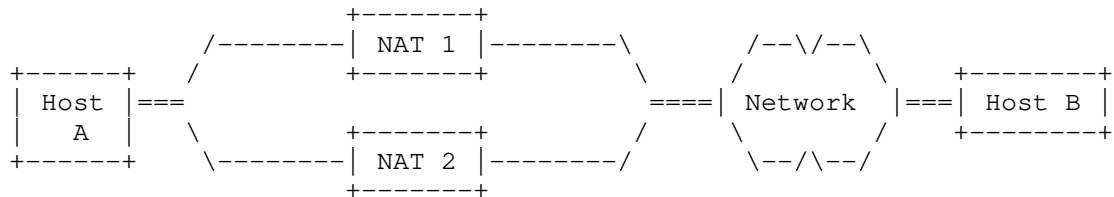
NAT 1	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	0	2	10.0.0.1

```

                                INIT[Initiate-Tag = 1234]
192.0.2.1:1 -----> 203.0.113.1:2
                                Rem-VTag = 0

```

Host B includes its second address in the INIT ACK.



```

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

```

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

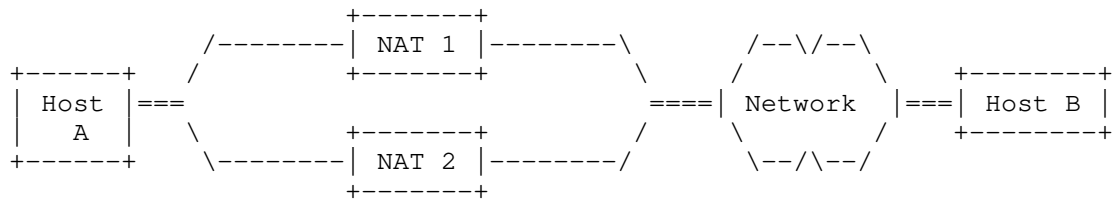
NAT 1	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	5678	2	10.0.0.1

```

INIT ACK[Initiate-Tag = 5678]
10.0.0.1:1 <----- 203.0.113.1:2
                                Int-VTag = 1234

```

The handshake finishes with a COOKIE ECHO acknowledged by a COOKIE ACK.



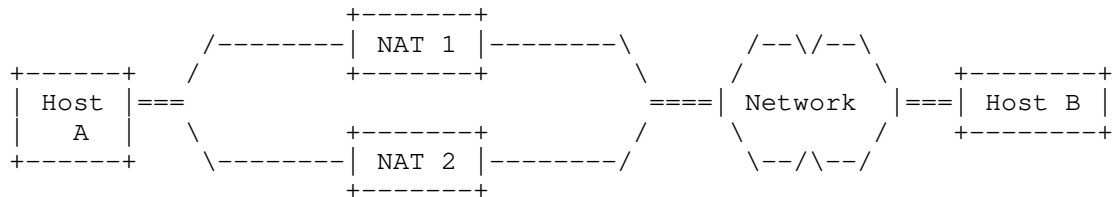
COOKIE ECHO
 10.0.0.1:1 -----> 203.0.113.1:2
 Rem-VTag = 5678

COOKIE ECHO
 192.0.2.1:1 -----> 203.0.113.1:2
 Rem-VTag = 5678

COOKIE ACK
 192.0.2.1:1 <----- 203.0.113.1:2
 Int-VTag = 1234

COOKIE ACK
 10.0.0.1:1 <----- 203.0.113.1:2
 Int-VTag = 1234

Host A announces its second address in an ASCONF chunk. The address parameter contains a wildcard address (0.0.0.0 or ::0) to indicate that the source address has to be added. The address parameter within the ASCONF chunk will also contain the pair of VTags (remote and internal) so that the NAT function can populate its NAT binding table entry completely with this single packet.



ASCONF [ADD-IP=0.0.0.0, INT-VTag=1234, Rem-VTag = 5678]
 10.1.0.1:1 -----> 203.0.113.129:2
 Rem-VTag = 5678

NAT function 2 creates a complete entry:

NAT 2	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	5678	2	10.1.0.1

```

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

```

```

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

```

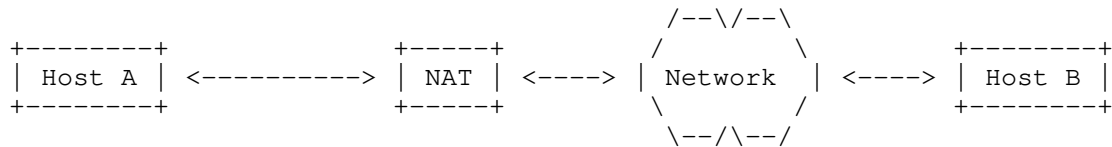
```

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

```

8.4. NAT Function Loses Its State

Association is already established between Host A and Host B, when the NAT function loses its state and obtains a new external address. Host A sends a DATA chunk to Host B.



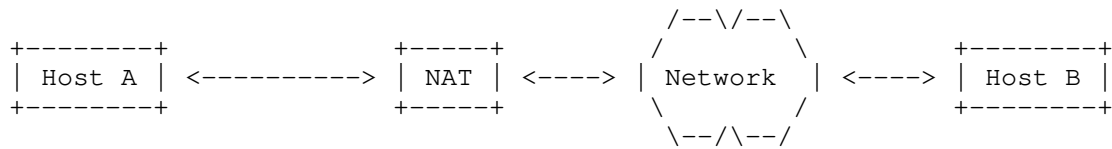
NAT	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr

```

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

```

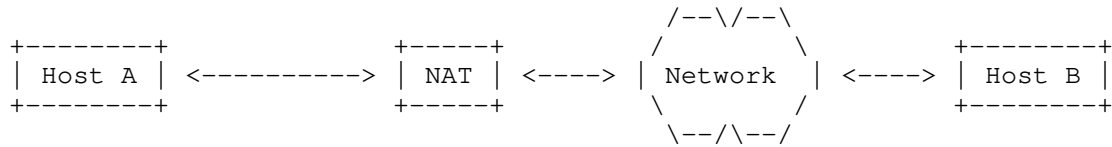
The NAT function cannot find an entry in the NAT binding table for the association. It sends a packet containing an ERROR chunk with the M bit set and the cause "NAT state missing".



```

ERROR [M bit, NAT state missing]
10.0.0.1:1 <----- 203.0.113.1:2
      Rem-VTag = 5678
  
```

On reception of the packet containing the ERROR chunk, Host A sends a packet containing an ASCONF chunk indicating that the former information has to be deleted and the source address of the actual packet added.



```

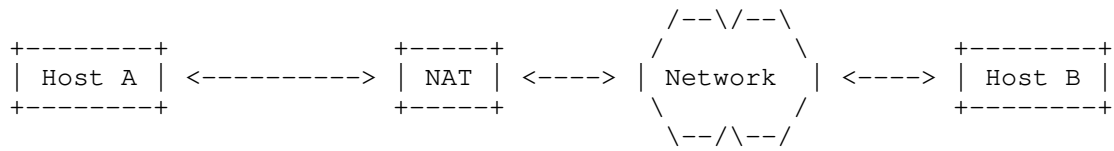
ASCONF [ADD-IP, DELETE-IP, Int-VTag=1234, Rem-VTag = 5678]
10.0.0.1:1 -----> 203.0.113.129:2
      Rem-VTag = 5678
  
```

NAT	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	5678	2	10.0.0.1

```

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

Host B adds the new source address to this association and deletes all other addresses from this association.



ASCONF ACK
 192.0.2.2:1 <----- 203.0.113.129:2
 Int-VTag = 1234

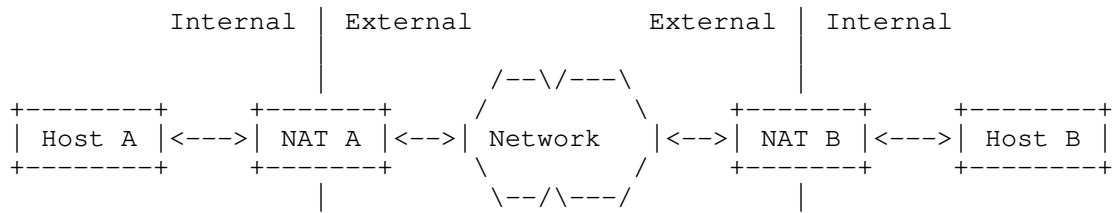
ASCONF ACK
 10.1.0.1:1 <----- 203.0.113.129:2
 Int-VTag = 1234

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

DATA
 192.0.2.2:1 -----> 203.0.113.129:2
 Rem-VTag = 5678

8.5. Peer-to-Peer Communications

If two hosts, each of them behind a NAT function, want to communicate with each other, they have to get knowledge of the peer's external address. This can be achieved with a so-called rendezvous server. Afterwards the destination addresses are external, and the association is set up with the help of the INIT collision. The NAT functions create their entries according to their internal peer's point of view. Therefore, NAT function A's Internal-VTag and Internal-Port are NAT function B's Remote-VTag and Remote-Port, respectively. The naming (internal/remote) of the verification tag in the packet flow is done from the sending host's point of view.



NAT Binding Tables

NAT A	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
-------	-------------	-------------	-------------	-------------	-------------

NAT B	Int v-tag	Int port	Rem v-tag	Rem port	Int Addr
-------	--------------	-------------	--------------	-------------	-------------

```

INIT[Initiate-Tag = 1234]
10.0.0.1:1 --> 203.0.113.1:2
    Rem-VTag = 0

```

NAT function A creates entry:

NAT A	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	0	2	10.0.0.1

```

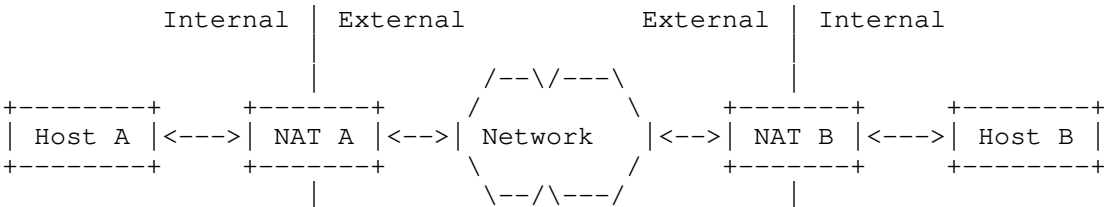
INIT[Initiate-Tag = 1234]
192.0.2.1:1 -----> 203.0.113.1:2
    Rem-VTag = 0

```

NAT function B processes the packet containing the INIT chunk, but cannot find an entry. The SCTP packet is silently discarded and leaves the NAT binding table of NAT function B unchanged.

NAT B	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
-------	-------------	-------------	-------------	-------------	-------------

Now Host B sends a packet containing an INIT chunk, which is processed by NAT function B. Its parameters are used to create an entry.

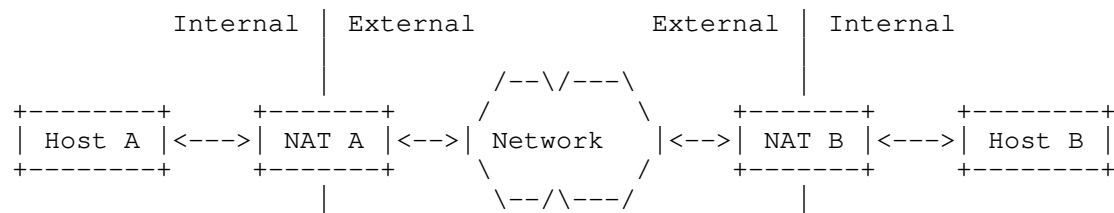


```
INIT[Initiate-Tag = 5678]
192.0.2.1:1 <-- 10.1.0.1:2
Rem-VTag = 0
```

NAT B	Int	Int	Rem	Rem	Int
	VTag	Port	VTag	Port	Addr
	5678	2	0	1	10.1.0.1

```
INIT[Initiate-Tag = 5678]
192.0.2.1:1 <----- 203.0.113.1:2
Rem-VTag = 0
```

NAT function A processes the packet containing the INIT chunk. As the outgoing packet containing an INIT chunk of Host A has already created an entry, the entry is found and updated:

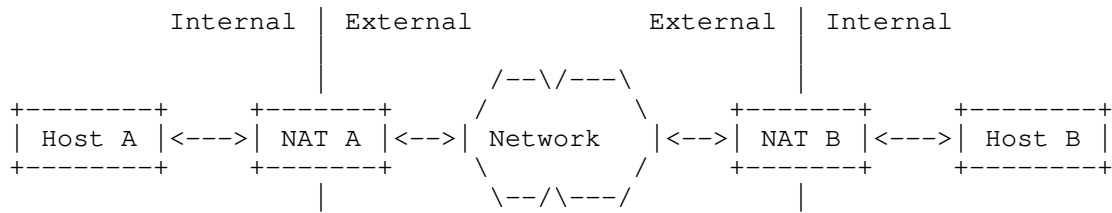


VTag != Int-VTag, but Rem-VTag == 0, find entry.

NAT A	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	5678	2	10.0.0.1

```
INIT[Initiate-tag = 5678]
10.0.0.1:1 <-- 203.0.113.1:2
    Rem-VTag = 0
```

Host A sends a packet containing an INIT ACK chunk, which can pass through NAT function B:



```

INIT ACK[Initiate-Tag = 1234]
10.0.0.1:1 --> 203.0.113.1:2
    Rem-VTag = 5678

```

```

        INIT ACK[Initiate-Tag = 1234]
192.0.2.1:1 -----> 203.0.113.1:2
        Rem-VTag = 5678

```

NAT function B updates entry:

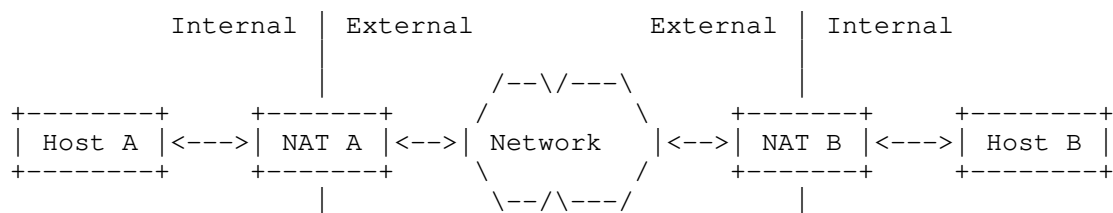
NAT B	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	-----	-----	-----	-----	-----
	5678	2	1234	1	10.1.0.1

```

INIT ACK[Initiate-Tag = 1234]
192.0.2.1:1 --> 10.1.0.1:2
    Rem-VTag = 5678

```

The lookup for COOKIE ECHO and COOKIE ACK is successful.



COOKIE ECHO
 192.0.2.1:1 <-- 10.1.0.1:2
 Rem-VTag = 1234

COOKIE ECHO
 192.0.2.1:1 <----- 203.0.113.1:2
 Rem-VTag = 1234

COOKIE ECHO
 10.0.0.1:1 <-- 203.0.113.1:2
 Rem-VTag = 1234

COOKIE ACK
 10.0.0.1:1 --> 203.0.113.1:2
 Rem-VTag = 5678

COOKIE ACK
 192.0.2.1:1 -----> 203.0.113.1:2
 Rem-VTag = 5678

COOKIE ACK
 192.0.2.1:1 --> 10.1.0.1:2
 Rem-VTag = 5678

9. Socket API Considerations

This section describes how the socket API defined in [RFC6458] is extended to provide a way for the application to control NAT friendliness.

Please note that this section is informational only.

A socket API implementation based on [RFC6458] is extended by supporting one new read/write socket option.

9.1. Get or Set the NAT Friendliness (SCTP_NAT_FRIENDLY)

This socket option uses the option_level IPPROTO_SCTP and the option_name SCTP_NAT_FRIENDLY. It can be used to enable/disable the NAT friendliness for future associations and retrieve the value for future and specific ones.

```
struct sctp_assoc_value {  
    sctp_assoc_t assoc_id;  
    uint32_t assoc_value;  
};
```

assoc_id

This parameter is ignored for one-to-one style sockets. For one-to-many style sockets the application can fill in an association identifier or SCTP_FUTURE_ASSOC for this query. It is an error to use SCTP_{CURRENT|ALL}_ASSOC in assoc_id.

assoc_value

A non-zero value indicates a NAT-friendly mode.

10. IANA Considerations

[NOTE to RFC-Editor: "RFCXXXX" is to be replaced by the RFC number you assign this document.]

[NOTE to RFC-Editor: The requested values for the chunk type and the chunk parameter types are tentative and to be confirmed by IANA.]

This document (RFCXXXX) is the reference for all registrations described in this section. The requested changes are described below.

10.1. New Chunk Flags for Two Existing Chunk Types

As defined in [RFC6096] two chunk flags have to be assigned by IANA for the ERROR chunk. The requested value for the T bit is 0x01 and for the M bit is 0x02.

This requires an update of the "ERROR Chunk Flags" registry for SCTP:

ERROR Chunk Flags

Chunk Flag Value	Chunk Flag Name	Reference
0x01	T bit	[RFCXXXX]
0x02	M bit	[RFCXXXX]
0x04	Unassigned	
0x08	Unassigned	
0x10	Unassigned	
0x20	Unassigned	
0x40	Unassigned	
0x80	Unassigned	

Table 2

As defined in [RFC6096] one chunk flag has to be assigned by IANA for the ABORT chunk. The requested value of the M bit is 0x02.

This requires an update of the "ABORT Chunk Flags" registry for SCTP:

ABORT Chunk Flags

Chunk Flag Value	Chunk Flag Name	Reference
0x01	T bit	[RFC4960]
0x02	M bit	[RFCXXXX]
0x04	Unassigned	
0x08	Unassigned	
0x10	Unassigned	
0x20	Unassigned	
0x40	Unassigned	
0x80	Unassigned	

Table 3

10.2. Three New Error Causes

Three error causes have to be assigned by IANA. It is requested to use the values given below.

This requires three additional lines in the "Error Cause Codes" registry for SCTP:

Error Cause Codes

Value	Cause Code	Reference
176	VTag and Port Number Collision	[RFCXXXX]
177	Missing State	[RFCXXXX]
178	Port Number Collision	[RFCXXXX]

Table 4

10.3. Two New Chunk Parameter Types

Two chunk parameter types have to be assigned by IANA. IANA is requested to assign these values from the pool of parameters with the upper two bits set to '11' and to use the values given below.

This requires two additional lines in the "Chunk Parameter Types" registry for SCTP:

Chunk Parameter Types

ID Value	Chunk Parameter Type	Reference
49159	Disable Restart (0xC007)	[RFCXXXX]
49160	VTags (0xC008)	[RFCXXXX]

Table 5

10.4. One New URI

An URI in the "ns" subregistry within the "IETF XML" registry has to be assigned by IANA ([RFC3688]):

URI: urn:ietf:params:xml:ns:yang:ietf-nat-sctp
 Registrant Contact: The IESG.
 XML: N/A; the requested URI is an XML namespace.

10.5. One New YANG Module

An YANG module in the "YANG Module Names" subregistry within the "YANG Parameters" registry has to be assigned by IANA ([RFC6020]):

Name: ietf-nat-sctp
 Namespace: urn:ietf:params:xml:ns:yang:ietf-nat-sctp
 Maintained by IANA: N
 Prefix: nat-sctp
 Reference: RFCXXXX

11. Security Considerations

State maintenance within a NAT function is always a subject of possible Denial Of Service attacks. This document recommends that at a minimum a NAT function runs a timer on any SCTP state so that old association state can be cleaned up.

Generic issues related to address sharing are discussed in [RFC6269] and apply to SCTP as well.

For SCTP endpoints not disabling the restart procedure, this document does not add any additional security considerations to the ones given in [RFC4960], [RFC4895], and [RFC5061].

SCTP endpoints disabling the restart procedure, need to monitor the status of all associations to mitigate resource exhaustion attacks by establishing a lot of associations sharing the same IP addresses and port numbers.

In any case, SCTP is protected by the verification tags and the usage of [RFC4895] against off-path attackers.

For IP-level fragmentation and reassembly related issues see [RFC4963].

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

All data nodes defined in the YANG module that can be created, modified, and deleted (i.e., config true, which is the default) are considered sensitive. Write operations (e.g., edit-config) applied to these data nodes without proper protection can negatively affect network operations. An attacker who is able to access the SCTP NAT function can undertake various attacks, such as:

- * Setting a low timeout for SCTP mapping entries to cause failures to deliver incoming SCTP packets.
- * Instantiating mapping entries to cause NAT collision.

12. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC4895] Tuexen, M., Stewart, R., Lei, P., and E. Rescorla, "Authenticated Chunks for the Stream Control Transmission Protocol (SCTP)", RFC 4895, DOI 10.17487/RFC4895, August 2007, <<https://www.rfc-editor.org/info/rfc4895>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.
- [RFC5061] Stewart, R., Xie, Q., Tuexen, M., Maruyama, S., and M. Kozuka, "Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration", RFC 5061, DOI 10.17487/RFC5061, September 2007, <<https://www.rfc-editor.org/info/rfc5061>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6096] Tuexen, M. and R. Stewart, "Stream Control Transmission Protocol (SCTP) Chunk Flags Registration", RFC 6096, DOI 10.17487/RFC6096, January 2011, <<https://www.rfc-editor.org/info/rfc6096>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8512] Boucadair, M., Ed., Sivakumar, S., Jacquenet, C., Vinapamula, S., and Q. Wu, "A YANG Module for Network Address Translation (NAT) and Network Prefix Translation (NPT)", RFC 8512, DOI 10.17487/RFC8512, January 2019, <<https://www.rfc-editor.org/info/rfc8512>>.

13. Informative References

- [DOI_10.1145_1496091.1496095]
Hayes, D., But, J., and G. Armitage, "Issues with network address translation for SCTP", ACM SIGCOMM Computer Communication Review Vol. 39, pp. 23-33, DOI 10.1145/1496091.1496095, December 2008, <<https://doi.org/10.1145/1496091.1496095>>.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC3022] Srisuresh, P. and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)", RFC 3022, DOI 10.17487/RFC3022, January 2001, <<https://www.rfc-editor.org/info/rfc3022>>.
- [RFC4787] Audet, F., Ed. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, DOI 10.17487/RFC4787, January 2007, <<https://www.rfc-editor.org/info/rfc4787>>.
- [RFC4963] Heffner, J., Mathis, M., and B. Chandler, "IPv4 Reassembly Errors at High Data Rates", RFC 4963, DOI 10.17487/RFC4963, July 2007, <<https://www.rfc-editor.org/info/rfc4963>>.
- [RFC5382] Guha, S., Ed., Biswas, K., Ford, B., Sivakumar, S., and P. Srisuresh, "NAT Behavioral Requirements for TCP", BCP 142, RFC 5382, DOI 10.17487/RFC5382, October 2008, <<https://www.rfc-editor.org/info/rfc5382>>.
- [RFC5508] Srisuresh, P., Ford, B., Sivakumar, S., and S. Guha, "NAT Behavioral Requirements for ICMP", BCP 148, RFC 5508, DOI 10.17487/RFC5508, April 2009, <<https://www.rfc-editor.org/info/rfc5508>>.
- [RFC6056] Larsen, M. and F. Gont, "Recommendations for Transport-Protocol Port Randomization", BCP 156, RFC 6056, DOI 10.17487/RFC6056, January 2011, <<https://www.rfc-editor.org/info/rfc6056>>.
- [RFC6146] Bagnulo, M., Matthews, P., and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", RFC 6146, DOI 10.17487/RFC6146, April 2011, <<https://www.rfc-editor.org/info/rfc6146>>.

- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC6269] Ford, M., Ed., Boucadair, M., Durand, A., Levis, P., and P. Roberts, "Issues with IP Address Sharing", RFC 6269, DOI 10.17487/RFC6269, June 2011, <<https://www.rfc-editor.org/info/rfc6269>>.
- [RFC6333] Durand, A., Droms, R., Woodyatt, J., and Y. Lee, "Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion", RFC 6333, DOI 10.17487/RFC6333, August 2011, <<https://www.rfc-editor.org/info/rfc6333>>.
- [RFC6458] Stewart, R., Tuexen, M., Poon, K., Lei, P., and V. Yasevich, "Sockets API Extensions for the Stream Control Transmission Protocol (SCTP)", RFC 6458, DOI 10.17487/RFC6458, December 2011, <<https://www.rfc-editor.org/info/rfc6458>>.
- [RFC6890] Cotton, M., Vegoda, L., Bonica, R., Ed., and B. Haberman, "Special-Purpose IP Address Registries", BCP 153, RFC 6890, DOI 10.17487/RFC6890, April 2013, <<https://www.rfc-editor.org/info/rfc6890>>.
- [RFC6951] Tuexen, M. and R. Stewart, "UDP Encapsulation of Stream Control Transmission Protocol (SCTP) Packets for End-Host to End-Host Communication", RFC 6951, DOI 10.17487/RFC6951, May 2013, <<https://www.rfc-editor.org/info/rfc6951>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC7857] Penno, R., Perreault, S., Boucadair, M., Ed., Sivakumar, S., and K. Naito, "Updates to Network Address Translation (NAT) Behavioral Requirements", BCP 127, RFC 7857, DOI 10.17487/RFC7857, April 2016, <<https://www.rfc-editor.org/info/rfc7857>>.

- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [RFC8900] Bonica, R., Baker, F., Huston, G., Hinden, R., Troan, O., and F. Gont, "IP Fragmentation Considered Fragile", BCP 230, RFC 8900, DOI 10.17487/RFC8900, September 2020, <<https://www.rfc-editor.org/info/rfc8900>>.

Acknowledgments

The authors wish to thank Mohamed Boucadair, Gorrry Fairhurst, Bryan Ford, David Hayes, Alfred Hines, Karen E. E. Nielsen, Henning Peters, Maksim Proshin, Timo Völker, Dan Wing, and Qiaobing Xie for their invaluable comments.

In addition, the authors wish to thank David Hayes, Jason But, and Grenville Armitage, the authors of [DOI_10.1145_1496091.1496095], for their suggestions.

The authors also wish to thank Mohamed Boucadair for contributing the text related to the YANG module.

Authors' Addresses

Randall R. Stewart
Netflix, Inc.
Chapin, SC 29036
United States of America

Email: randall@lakerest.net

Michael Tüxen
Münster University of Applied Sciences
Stegerwaldstrasse 39
48565 Steinfurt
Germany

Email: tuexen@fh-muenster.de

Irene Rüngeler
Münster University of Applied Sciences
Stegerwaldstrasse 39
48565 Steinfurt
Germany

Email: i.ruengeler@fh-muenster.de

Network Working Group
Internet-Draft

Obsoletes: 4460, 4960, 6096, 7053, 8540 (if
approved)

Intended status: Standards Track

Expires: 9 August 2022

R. R. Stewart
Netflix, Inc.

M. Tüxen
Münster Univ. of Appl. Sciences

K. E. E. Nielsen

Kamstrup A/S
5 February 2022

Stream Control Transmission Protocol
draft-ietf-tsvwg-rfc4960-bis-19

Abstract

This document obsoletes RFC 4960, if approved. It describes the Stream Control Transmission Protocol (SCTP) and incorporates the specification of the chunk flags registry from RFC 6096 and the specification of the I bit of DATA chunks from RFC 7053. Therefore, RFC 6096 and RFC 7053 are also obsoleted by this document, if approved. In addition to that, the Errata documents RFC 4460 and RFC 8540 are also obsoleted by this document, if approved.

SCTP was originally designed to transport Public Switched Telephone Network (PSTN) signaling messages over IP networks. It is also suited to be used for other applications, for example WebRTC.

SCTP is a reliable transport protocol operating on top of a connectionless packet network such as IP. It offers the following services to its users:

- * acknowledged error-free non-duplicated transfer of user data,
- * data fragmentation to conform to discovered path maximum transmission unit (PMTU) size,
- * sequenced delivery of user messages within multiple streams, with an option for order-of-arrival delivery of individual user messages,
- * optional bundling of multiple user messages into a single SCTP packet, and
- * network-level fault tolerance through supporting of multi-homing at either or both ends of an association.

The design of SCTP includes appropriate congestion avoidance behavior and resistance to flooding and masquerade attacks.

Status of This Memo

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

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

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

This Internet-Draft will expire on 9 August 2022.

Copyright Notice

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

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

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. Conventions	6
2. Introduction	6

2.1.	Motivation	7
2.2.	Architectural View of SCTP	7
2.3.	Key Terms	8
2.4.	Abbreviations	12
2.5.	Functional View of SCTP	13
2.5.1.	Association Startup and Takedown	13
2.5.2.	Sequenced Delivery within Streams	14
2.5.3.	User Data Fragmentation	15
2.5.4.	Acknowledgement and Congestion Avoidance	15
2.5.5.	Chunk Bundling	15
2.5.6.	Packet Validation	16
2.5.7.	Path Management	16
2.6.	Serial Number Arithmetic	17
2.7.	Changes from RFC 4960	17
3.	SCTP Packet Format	18
3.1.	SCTP Common Header Field Descriptions	19
3.2.	Chunk Field Descriptions	20
3.2.1.	Optional/Variable-Length Parameter Format	23
3.2.2.	Reporting of Unrecognized Parameters	25
3.3.	SCTP Chunk Definitions	25
3.3.1.	Payload Data (DATA) (0)	25
3.3.2.	Initiation (INIT) (1)	28
3.3.2.1.	Optional or Variable-Length Parameters in INIT chunks	32
3.3.3.	Initiation Acknowledgement (INIT ACK) (2)	35
3.3.3.1.	Optional or Variable-Length Parameters in INIT ACK chunks	39
3.3.4.	Selective Acknowledgement (SACK) (3)	40
3.3.5.	Heartbeat Request (HEARTBEAT) (4)	43
3.3.6.	Heartbeat Acknowledgement (HEARTBEAT ACK) (5)	44
3.3.7.	Abort Association (ABORT) (6)	45
3.3.8.	Shutdown Association (SHUTDOWN) (7)	46
3.3.9.	Shutdown Acknowledgement (SHUTDOWN ACK) (8)	47
3.3.10.	Operation Error (ERROR) (9)	47
3.3.10.1.	Invalid Stream Identifier (1)	49
3.3.10.2.	Missing Mandatory Parameter (2)	50
3.3.10.3.	Stale Cookie Error (3)	50
3.3.10.4.	Out of Resource (4)	51
3.3.10.5.	Unresolvable Address (5)	51
3.3.10.6.	Unrecognized Chunk Type (6)	52
3.3.10.7.	Invalid Mandatory Parameter (7)	52
3.3.10.8.	Unrecognized Parameters (8)	52
3.3.10.9.	No User Data (9)	53
3.3.10.10.	Cookie Received While Shutting Down (10)	53
3.3.10.11.	Restart of an Association with New Addresses (11)	54
3.3.10.12.	User-Initiated Abort (12)	54
3.3.10.13.	Protocol Violation (13)	54

3.3.11. Cookie Echo (COOKIE ECHO) (10)	55
3.3.12. Cookie Acknowledgement (COOKIE ACK) (11)	56
3.3.13. Shutdown Complete (SHUTDOWN COMPLETE) (14)	56
4. SCTP Association State Diagram	57
5. Association Initialization	60
5.1. Normal Establishment of an Association	60
5.1.1. Handle Stream Parameters	62
5.1.2. Handle Address Parameters	63
5.1.3. Generating State Cookie	64
5.1.4. State Cookie Processing	65
5.1.5. State Cookie Authentication	65
5.1.6. An Example of Normal Association Establishment	66
5.2. Handle Duplicate or Unexpected INIT, INIT ACK, COOKIE ECHO, and COOKIE ACK Chunks	68
5.2.1. INIT Chunk Received in COOKIE-WAIT or COOKIE-ECHOED State (Item B)	68
5.2.2. Unexpected INIT Chunk in States Other than CLOSED, COOKIE-ECHOED, COOKIE-WAIT, and SHUTDOWN-ACK-SENT	69
5.2.3. Unexpected INIT ACK Chunk	70
5.2.4. Handle a COOKIE ECHO Chunk when a TCB Exists	70
5.2.4.1. An Example of a Association Restart	73
5.2.5. Handle Duplicate COOKIE ACK Chunk	74
5.2.6. Handle Stale Cookie Error	74
5.3. Other Initialization Issues	74
5.3.1. Selection of Tag Value	75
5.4. Path Verification	75
6. User Data Transfer	76
6.1. Transmission of DATA Chunks	78
6.2. Acknowledgement on Reception of DATA Chunks	81
6.2.1. Processing a Received SACK Chunk	84
6.3. Management of Retransmission Timer	86
6.3.1. RTO Calculation	86
6.3.2. Retransmission Timer Rules	88
6.3.3. Handle T3-rtx Expiration	89
6.4. Multi-Homed SCTP Endpoints	90
6.4.1. Failover from an Inactive Destination Address	91
6.5. Stream Identifier and Stream Sequence Number	92
6.6. Ordered and Unordered Delivery	92
6.7. Report Gaps in Received DATA TSNS	93
6.8. CRC32c Checksum Calculation	94
6.9. Fragmentation and Reassembly	95
6.10. Bundling	96
7. Congestion Control	97
7.1. SCTP Differences from TCP Congestion Control	98
7.2. SCTP Slow-Start and Congestion Avoidance	99
7.2.1. Slow-Start	100
7.2.2. Congestion Avoidance	101
7.2.3. Congestion Control	102

7.2.4.	Fast Retransmit on Gap Reports	102
7.2.5.	Reinitialization	104
7.2.5.1.	Change of Differentiated Services Code Points . .	104
7.2.5.2.	Change of Routes	104
7.3.	PMTU Discovery	104
8.	Fault Management	105
8.1.	Endpoint Failure Detection	105
8.2.	Path Failure Detection	105
8.3.	Path Heartbeat	106
8.4.	Handle "Out of the Blue" Packets	109
8.5.	Verification Tag	110
8.5.1.	Exceptions in Verification Tag Rules	110
9.	Termination of Association	111
9.1.	Abort of an Association	112
9.2.	Shutdown of an Association	112
10.	ICMP Handling	115
11.	Interface with Upper Layer	116
11.1.	ULP-to-SCTP	117
11.1.1.	Initialize	117
11.1.2.	Associate	118
11.1.3.	Shutdown	119
11.1.4.	Abort	119
11.1.5.	Send	119
11.1.6.	Set Primary	121
11.1.7.	Receive	121
11.1.8.	Status	122
11.1.9.	Change Heartbeat	123
11.1.10.	Request Heartbeat	124
11.1.11.	Get SRTT Report	124
11.1.12.	Set Failure Threshold	125
11.1.13.	Set Protocol Parameters	125
11.1.14.	Receive Unsent Message	125
11.1.15.	Receive Unacknowledged Message	126
11.1.16.	Destroy SCTP Instance	127
11.2.	SCTP-to-ULP	127
11.2.1.	DATA ARRIVE Notification	127
11.2.2.	SEND FAILURE Notification	128
11.2.3.	NETWORK STATUS CHANGE Notification	128
11.2.4.	COMMUNICATION UP Notification	128
11.2.5.	COMMUNICATION LOST Notification	129
11.2.6.	COMMUNICATION ERROR Notification	130
11.2.7.	RESTART Notification	130
11.2.8.	SHUTDOWN COMPLETE Notification	130
12.	Security Considerations	130
12.1.	Security Objectives	130
12.2.	SCTP Responses to Potential Threats	131
12.2.1.	Countering Insider Attacks	131
12.2.2.	Protecting against Data Corruption in the Network .	131

12.2.3.	Protecting Confidentiality	131
12.2.4.	Protecting against Blind Denial-of-Service Attacks	132
12.2.4.1.	Flooding	132
12.2.4.2.	Blind Masquerade	133
12.2.4.3.	Improper Monopolization of Services	134
12.3.	SCTP Interactions with Firewalls	134
12.4.	Protection of Non-SCTP-Capable Hosts	134
13.	Network Management Considerations	135
14.	Recommended Transmission Control Block (TCB) Parameters	135
14.1.	Parameters Necessary for the SCTP Instance	135
14.2.	Parameters Necessary per Association (i.e., the TCB)	136
14.3.	Per Transport Address Data	138
14.4.	General Parameters Needed	139
15.	IANA Considerations	139
15.1.	IETF-Defined Chunk Extension	143
15.2.	IETF Chunk Flags Registration	144
15.3.	IETF-Defined Chunk Parameter Extension	144
15.4.	IETF-Defined Additional Error Causes	144
15.5.	Payload Protocol Identifiers	145
15.6.	Port Numbers Registry	145
16.	Suggested SCTP Protocol Parameter Values	145
17.	Acknowledgements	146
18.	Normative References	147
19.	Informative References	149
Appendix A.	CRC32c Checksum Calculation	152
Authors' Addresses	159

1. Conventions

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

2. Introduction

This section explains the reasoning behind the development of the Stream Control Transmission Protocol (SCTP), the services it offers, and the basic concepts needed to understand the detailed description of the protocol.

This document obsoletes [RFC4960], if approved. In addition to that, it incorporates the specification of the chunk flags registry from [RFC6096] and the specification of the I bit of DATA chunks from [RFC7053]. Therefore, [RFC6096] and [RFC7053] are also obsoleted by this document, if approved.

2.1. Motivation

TCP [RFC0793] has performed immense service as the primary means of reliable data transfer in IP networks. However, an increasing number of recent applications have found TCP too limiting, and have incorporated their own reliable data transfer protocol on top of UDP [RFC0768]. The limitations that users have wished to bypass include the following:

- * TCP provides both reliable data transfer and strict order-of-transmission delivery of data. Some applications need reliable transfer without sequence maintenance, while others would be satisfied with partial ordering of the data. In both of these cases, the head-of-line blocking offered by TCP causes unnecessary delay.
- * The stream-oriented nature of TCP is often an inconvenience. Applications add their own record marking to delineate their messages, and make explicit use of the push facility to ensure that a complete message is transferred in a reasonable time.
- * The limited scope of TCP sockets complicates the task of providing highly-available data transfer capability using multi-homed hosts.
- * TCP is relatively vulnerable to denial-of-service attacks, such as SYN attacks.

Transport of PSTN signaling across the IP network is an application for which all of these limitations of TCP are relevant. While this application directly motivated the development of SCTP, other applications might find SCTP a good match to their requirements. One example of this is the use of datachannels in the WebRTC infrastructure.

2.2. Architectural View of SCTP

SCTP is viewed as a layer between the SCTP user application ("SCTP user" for short) and a connectionless packet network service such as IP. The remainder of this document assumes SCTP runs on top of IP. The basic service offered by SCTP is the reliable transfer of user messages between peer SCTP users. It performs this service within the context of an association between two SCTP endpoints. Section 11 of this document sketches the API that exists at the boundary between the SCTP and the SCTP upper layers.

SCTP is connection-oriented in nature, but the SCTP association is a broader concept than the TCP connection. SCTP provides the means for each SCTP endpoint (Section 2.3) to provide the other endpoint

(during association startup) with a list of transport addresses (i.e., multiple IP addresses in combination with an SCTP port) through which that endpoint can be reached and from which it will originate SCTP packets. The association spans transfers over all of the possible source/destination combinations that can be generated from each endpoint's lists.

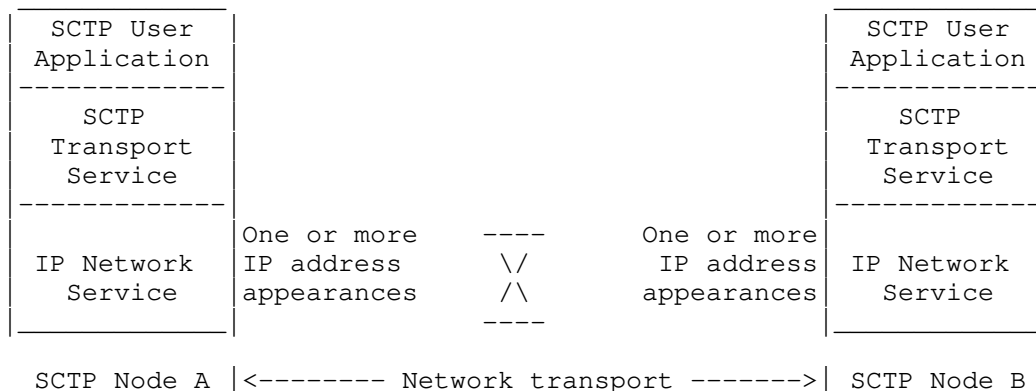


Figure 1: An SCTP Association

In addition to encapsulating SCTP packets in IPv4 or IPv6, it is also possible to encapsulate SCTP packets in UDP as specified in [RFC6951] or encapsulate them in DTLS as specified in [RFC8261].

2.3. Key Terms

Some of the language used to describe SCTP has been introduced in the previous sections. This section provides a consolidated list of the key terms and their definitions.

Active Destination Transport Address: A transport address on a peer endpoint that a transmitting endpoint considers available for receiving user messages.

Association Maximum DATA Chunk Size (AMDCS): The smallest Path Maximum DATA Chunk Size (PMDCS) of all destination addresses.

Bundling Of Chunks: An optional multiplexing operation, whereby more than one chunk can be carried in the same SCTP packet.

Bundling Of User Messages: An optional multiplexing operation, whereby more than one user message can be carried in the same SCTP packet. Each user message occupies its own DATA chunk.

Chunk: A unit of information within an SCTP packet, consisting of a chunk header and chunk-specific content.

Congestion Window (cwnd): An SCTP variable that limits outstanding data, in number of bytes, that a sender can send to a particular destination transport address before receiving an acknowledgement.

Control Chunk: A chunk not being used for transmitting user data, i.e. every chunk which is not a DATA chunk.

Cumulative TSN Ack Point: The Transmission Sequence Number (TSN) of the last DATA chunk acknowledged via the Cumulative TSN Ack field of a SACK chunk.

Flightsize: The number of bytes of outstanding data to a particular destination transport address at any given time.

Idle Destination Address: An address that has not had user messages sent to it within some length of time, normally the 'HB.interval' or greater.

Inactive Destination Transport Address: An address that is considered inactive due to errors and unavailable to transport user messages.

Message (or User Message): Data submitted to SCTP by the Upper Layer Protocol (ULP).

Network Byte Order: Most significant byte first, a.k.a., big endian.

Ordered Message: A user message that is delivered in order with respect to all previous user messages sent within the stream on which the message was sent.

Outstanding Data (or Data Outstanding or Data In Flight): The total size of the DATA chunks associated with outstanding TSNs. A retransmitted DATA chunk is counted once in outstanding data. A DATA chunk that is classified as lost but that has not yet been retransmitted is not in outstanding data.

Outstanding TSN (at an SCTP Endpoint): A TSN (and the associated DATA chunk) that has been sent by the endpoint but for which it has not yet received an acknowledgement.

Out Of The Blue (OOTB) Packet: A correctly formed packet, for which the receiver can not identify the association it belongs to. See Section 8.4.

Path: The route taken by the SCTP packets sent by one SCTP endpoint to a specific destination transport address of its peer SCTP endpoint. Sending to different destination transport addresses does not necessarily guarantee getting separate paths. Within this specification, a path is identified by the destination transport address, since the routing is assumed to be stable. This includes in particular the source address being selected when sending packets to the destination address.

Path Maximum DATA Chunk Size (PMDCS): The maximum size (including the DATA chunk header) of a DATA chunk which fits into an SCTP packet not exceeding the PMTU of a particular destination address.

Path Maximum Transmission Unit (PMTU): The maximum size (including the SCTP common header and all chunks including their paddings) of an SCTP packet which can be sent to a particular destination address without using IP level fragmentation.

Primary Path: The primary path is the destination and source address that will be put into a packet outbound to the peer endpoint by default. The definition includes the source address since an implementation MAY wish to specify both destination and source address to better control the return path taken by reply chunks and on which interface the packet is transmitted when the data sender is multi-homed.

Receiver Window (rwnd): An SCTP variable a data sender uses to store the most recently calculated receiver window of its peer, in number of bytes. This gives the sender an indication of the space available in the receiver's inbound buffer.

SCTP Association: A protocol relationship between SCTP endpoints, composed of the two SCTP endpoints and protocol state information including Verification Tags and the currently active set of Transmission Sequence Numbers (TSNs), etc. An association can be uniquely identified by the transport addresses used by the endpoints in the association. Two SCTP endpoints MUST NOT have more than one SCTP association between them at any given time.

SCTP Endpoint: The logical sender/receiver of SCTP packets. On a

multi-homed host, an SCTP endpoint is represented to its peers as a combination of a set of eligible destination transport addresses to which SCTP packets can be sent and a set of eligible source transport addresses from which SCTP packets can be received. All transport addresses used by an SCTP endpoint MUST use the same port number, but can use multiple IP addresses. A transport address used by an SCTP endpoint MUST NOT be used by another SCTP endpoint. In other words, a transport address is unique to an SCTP endpoint.

SCTP Packet (or Packet): The unit of data delivery across the interface between SCTP and the connectionless packet network (e.g., IP). An SCTP packet includes the common SCTP header, possible SCTP control chunks, and user data encapsulated within SCTP DATA chunks.

SCTP User Application (or SCTP User): The logical higher-layer application entity which uses the services of SCTP, also called the Upper-Layer Protocol (ULP).

Slow-Start Threshold (ssthresh): An SCTP variable. This is the threshold that the endpoint will use to determine whether to perform slow start or congestion avoidance on a particular destination transport address. Ssthresh is in number of bytes.

State Cookie: A container of all information needed to establish an association.

Stream: A unidirectional logical channel established from one to another associated SCTP endpoint, within which all user messages are delivered in sequence except for those submitted to the unordered delivery service.

Note: The relationship between stream numbers in opposite directions is strictly a matter of how the applications use them. It is the responsibility of the SCTP user to create and manage these correlations if they are so desired.

Stream Sequence Number: A 16-bit sequence number used internally by SCTP to ensure sequenced delivery of the user messages within a given stream. One Stream Sequence Number is attached to each ordered user message.

Tie-Tags: Two 32-bit random numbers that together make a 64-bit nonce. These tags are used within a State Cookie and TCB so that a newly restarting association can be linked to the original association within the endpoint that did not restart and yet not reveal the true Verification Tags of an existing association.

Transmission Control Block (TCB): An internal data structure created by an SCTP endpoint for each of its existing SCTP associations to other SCTP endpoints. TCB contains all the status and operational information for the endpoint to maintain and manage the corresponding association.

Transmission Sequence Number (TSN): A 32-bit sequence number used internally by SCTP. One TSN is attached to each chunk containing user data to permit the receiving SCTP endpoint to acknowledge its receipt and detect duplicate deliveries.

Transport Address: A transport address is traditionally defined by a network-layer address, a transport-layer protocol, and a transport-layer port number. In the case of SCTP running over IP, a transport address is defined by the combination of an IP address and an SCTP port number (where SCTP is the transport protocol).

Unordered Message: Unordered messages are "unordered" with respect to any other message; this includes both other unordered messages as well as other ordered messages. An unordered message might be delivered prior to or later than ordered messages sent on the same stream.

User Message: The unit of data delivery across the interface between SCTP and its user.

Verification Tag: A 32-bit unsigned integer that is randomly generated. The Verification Tag provides a key that allows a receiver to verify that the SCTP packet belongs to the current association and is not an old or stale packet from a previous association.

2.4. Abbreviations

MAC Message Authentication Code [RFC2104]
RTO Retransmission Timeout
RTT Round-Trip Time
RTTVAR Round-Trip Time Variation
SCTP Stream Control Transmission Protocol
SRTT Smoothed RTT
TCB Transmission Control Block
TLV Type-Length-Value coding format
TSN Transmission Sequence Number
ULP Upper-Layer Protocol

2.5. Functional View of SCTP

The SCTP transport service can be decomposed into a number of functions. These are depicted in Figure 2 and explained in the remainder of this section.

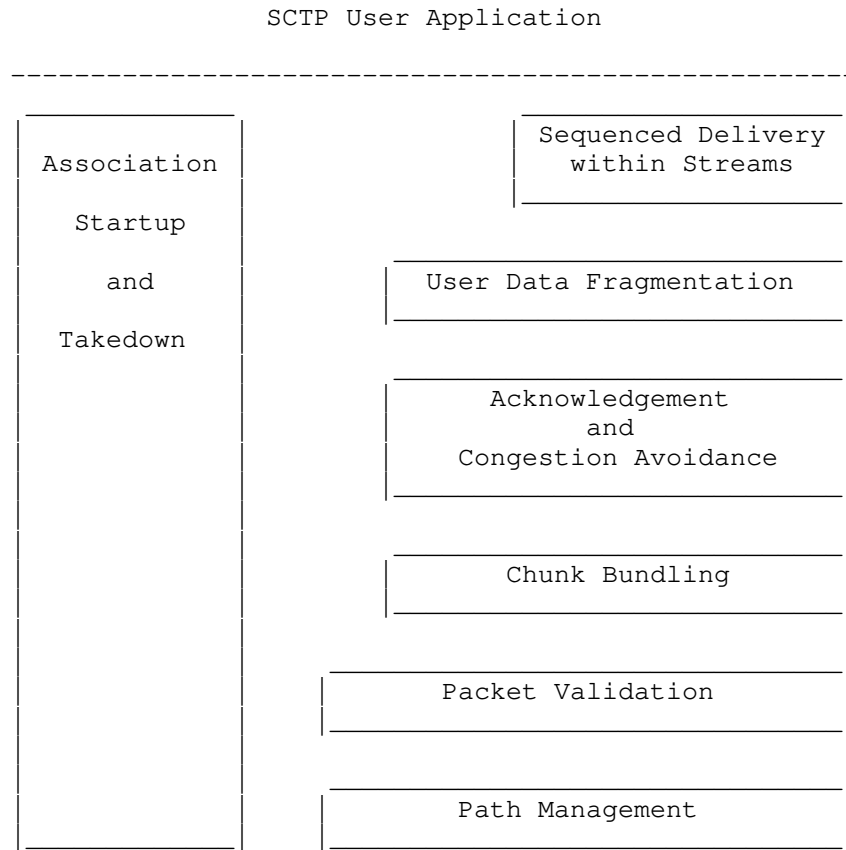


Figure 2: Functional View of the SCTP Transport Service

2.5.1. Association Startup and Takedown

An association is initiated by a request from the SCTP user (see the description of the ASSOCIATE (or SEND) primitive in Section 11).

A cookie mechanism, similar to one described by Karn and Simpson in [RFC2522], is employed during the initialization to provide protection against synchronization attacks. The cookie mechanism uses a four-way handshake, the last two legs of which are allowed to carry user data for fast setup. The startup sequence is described in Section 5 of this document.

SCTP provides for graceful close (i.e., shutdown) of an active association on request from the SCTP user. See the description of the SHUTDOWN primitive in Section 11. SCTP also allows ungraceful close (i.e., abort), either on request from the user (ABORT primitive) or as a result of an error condition detected within the SCTP layer. Section 9 describes both the graceful and the ungraceful close procedures.

SCTP does not support a half-open state (like TCP) wherein one side continues sending data while the other end is closed. When either endpoint performs a shutdown, the association on each peer will stop accepting new data from its user and only deliver data in queue at the time of the graceful close (see Section 9).

2.5.2. Sequenced Delivery within Streams

The term "stream" is used in SCTP to refer to a sequence of user messages that are to be delivered to the upper-layer protocol in order with respect to other messages within the same stream. This is in contrast to its usage in TCP, where it refers to a sequence of bytes (in this document, a byte is assumed to be 8 bits).

The SCTP user can specify at association startup time the number of streams to be supported by the association. This number is negotiated with the remote end (see Section 5.1.1). User messages are associated with stream numbers (SEND, RECEIVE primitives, Section 11). Internally, SCTP assigns a Stream Sequence Number to each message passed to it by the SCTP user. On the receiving side, SCTP ensures that messages are delivered to the SCTP user in sequence within a given stream. However, while one stream might be blocked waiting for the next in-sequence user message, delivery from other streams might proceed.

SCTP provides a mechanism for bypassing the sequenced delivery service. User messages sent using this mechanism are delivered to the SCTP user as soon as they are received.

2.5.3. User Data Fragmentation

When needed, SCTP fragments user messages to ensure that the size of the SCTP packet passed to the lower layer does not exceed the PMTU. Once a user message has been fragmented, this fragmentation cannot be changed anymore. On receipt, fragments are reassembled into complete messages before being passed to the SCTP user.

2.5.4. Acknowledgement and Congestion Avoidance

SCTP assigns a Transmission Sequence Number (TSN) to each user data fragment or unfragmented message. The TSN is independent of any Stream Sequence Number assigned at the stream level. The receiving end acknowledges all TSNs received, even if there are gaps in the sequence. If a user data fragment or unfragmented message needs to be retransmitted, the TSN assigned to it is used. In this way, reliable delivery is kept functionally separate from sequenced stream delivery.

The acknowledgement and congestion avoidance function is responsible for packet retransmission when timely acknowledgement has not been received. Packet retransmission is conditioned by congestion avoidance procedures similar to those used for TCP. See Section 6 and Section 7 for a detailed description of the protocol procedures associated with this function.

2.5.5. Chunk Bundling

As described in Section 3, the SCTP packet as delivered to the lower layer consists of a common header followed by one or more chunks. Each chunk contains either user data or SCTP control information. An SCTP implementation supporting bundling on the sender side might delay the sending of user messages to allow the corresponding DATA chunks to be bundled.

The SCTP user has the option to request that an SCTP implementation does not delay the sending of a user message just for this purpose. However, even if the SCTP user has chosen this option, the SCTP implementation might delay the sending due to other reasons, for example due to congestion control or flow control, and might also bundle multiple DATA chunks, if possible.

2.5.6. Packet Validation

A mandatory Verification Tag field and a 32-bit checksum field (see Appendix A for a description of the CRC32c checksum) are included in the SCTP common header. The Verification Tag value is chosen by each end of the association during association startup. Packets received without the expected Verification Tag value are discarded, as a protection against blind masquerade attacks and against stale SCTP packets from a previous association. The CRC32c checksum is set by the sender of each SCTP packet to provide additional protection against data corruption in the network. The receiver of an SCTP packet with an invalid CRC32c checksum silently discards the packet.

2.5.7. Path Management

The sending SCTP user is able to manipulate the set of transport addresses used as destinations for SCTP packets through the primitives described in Section 11. The SCTP path management function monitors reachability through heartbeats when other packet traffic is inadequate to provide this information and advises the SCTP user when reachability of any transport address of the peer endpoint changes. The path management function chooses the destination transport address for each outgoing SCTP packet based on the SCTP user's instructions and the currently perceived reachability status of the eligible destination set. The path management function is also responsible for reporting the eligible set of local transport addresses to the peer endpoint during association startup, and for reporting the transport addresses returned from the peer endpoint to the SCTP user.

At association startup, a primary path is defined for each SCTP endpoint, and is used for normal sending of SCTP packets.

On the receiving end, the path management is responsible for verifying the existence of a valid SCTP association to which the inbound SCTP packet belongs before passing it for further processing.

Note: Path Management and Packet Validation are done at the same time, so although described separately above, in reality they cannot be performed as separate items.

2.6. Serial Number Arithmetic

It is essential to remember that the actual Transmission Sequence Number space is finite, though very large. This space ranges from 0 to $2^{32} - 1$. Since the space is finite, all arithmetic dealing with Transmission Sequence Numbers MUST be performed modulo 2^{32} . This unsigned arithmetic preserves the relationship of sequence numbers as they cycle from $2^{32} - 1$ to 0 again. There are some subtleties to computer modulo arithmetic, so great care has to be taken in programming the comparison of such values. When referring to TSNs, the symbol " \leq " means "less than or equal" (modulo 2^{32}).

Comparisons and arithmetic on TSNs in this document SHOULD use Serial Number Arithmetic as defined in [RFC1982] where SERIAL_BITS = 32.

An endpoint SHOULD NOT transmit a DATA chunk with a TSN that is more than $2^{31} - 1$ above the beginning TSN of its current send window. Doing so will cause problems in comparing TSNs.

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

Any arithmetic done on Stream Sequence Numbers SHOULD use Serial Number Arithmetic as defined in [RFC1982] where SERIAL_BITS = 16. All other arithmetic and comparisons in this document use normal arithmetic.

2.7. Changes from RFC 4960

SCTP was originally defined in [RFC4960], which this document obsoletes, if approved. Readers interested in the details of the various changes that this document incorporates are asked to consult [RFC8540].

In addition to these and further editorial changes, the following changes have been incorporated in this document:

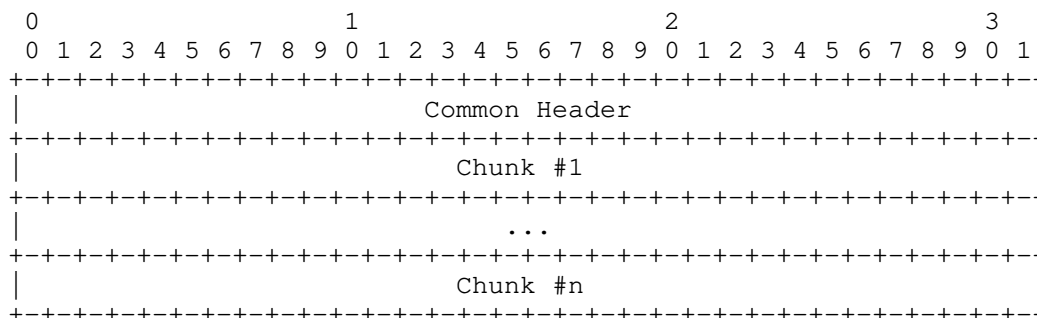
- * Update references.
- * Improve the language related to requirements levels.
- * Allow the ASSOCIATE primitive to take multiple remote addresses; also refer to the Socket API specification.
- * Refer to the PLPMTUD specification for path MTU discovery.

- * Move the description of ICMP handling from an Appendix to the main text.
- * Remove the Appendix describing ECN handling from the document.
- * Describe the packet size handling more precisely by introducing PMTU, PMDCS and AMDCS.
- * Add the definition of control chunk.
- * Improve the description of the handling of INIT and INIT ACK chunks with invalid mandatory parameters.
- * Allow using $L > 1$ for Appropriate Byte Counting (ABC) during slow start.
- * Explicitly describe the reinitialization of the congestion controller on route changes.
- * Improve the terminology to make clear that this specification does not describe a full mesh architecture.
- * Improve the description of sequence number generation (Transmission Sequence Number and Stream Sequence Number).
- * Improve the description of reneging.
- * Don't require the change of the cumulative TSN ACK anymore for increasing the congestion window. This improves the consistency with the handling in congestion avoidance.
- * Improve the description of the State Cookie.
- * Fix the API for retrieving messages in case of association failures.

3. SCTP Packet Format

An SCTP packet is composed of a common header and chunks. A chunk contains either control information or user data.

The SCTP packet format is shown below:

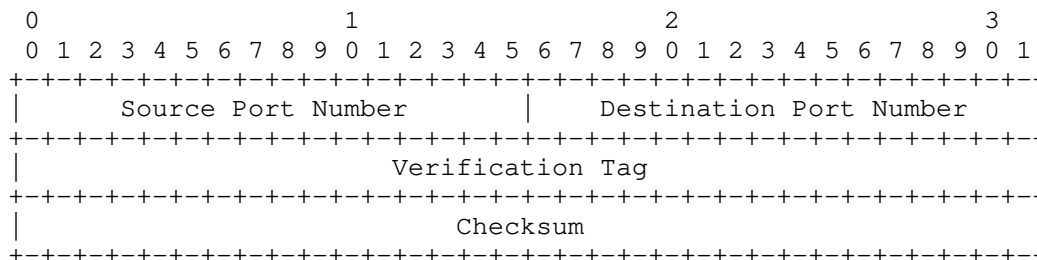


INIT, INIT ACK and SHUTDOWN COMPLETE chunks MUST NOT be bundled with any other chunk into an SCTP packet. All other chunks MAY be bundled to form an SCTP packet that does not exceed the PMTU. See Section 6.10 for more details on chunk bundling.

If a user data message does not fit into one SCTP packet it can be fragmented into multiple chunks using the procedure defined in Section 6.9.

All integer fields in an SCTP packet MUST be transmitted in network byte order, unless otherwise stated.

3.1. SCTP Common Header Field Descriptions



Source Port Number: 16 bits (unsigned integer)

This is the SCTP sender's port number. It can be used by the receiver in combination with the source IP address, the SCTP destination port, and possibly the destination IP address to identify the association to which this packet belongs. The source port number 0 MUST NOT be used.

Destination Port Number: 16 bits (unsigned integer)

This is the SCTP port number to which this packet is destined. The receiving host will use this port number to de-multiplex the SCTP packet to the correct receiving endpoint/application. The destination port number 0 MUST NOT be used.

Verification Tag: 32 bits (unsigned integer)

The receiver of an SCTP packet uses the Verification Tag to validate the sender of this packet. On transmit, the value of the Verification Tag MUST be set to the value of the Initiate Tag received from the peer endpoint during the association initialization, with the following exceptions:

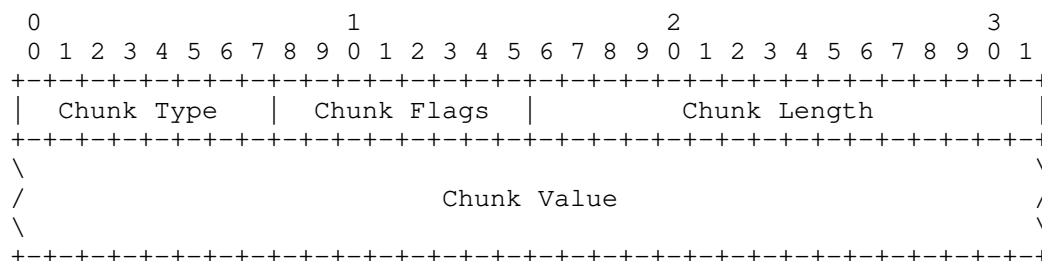
- * A packet containing an INIT chunk MUST have a zero Verification Tag.
- * A packet containing a SHUTDOWN COMPLETE chunk with the T bit set MUST have the Verification Tag copied from the packet with the SHUTDOWN ACK chunk.
- * A packet containing an ABORT chunk MAY have the verification tag copied from the packet that caused the ABORT chunk to be sent. For details see Section 8.4 and Section 8.5.

Checksum: 32 bits (unsigned integer)

This field contains the checksum of the SCTP packet. Its calculation is discussed in Section 6.8. SCTP uses the CRC32c algorithm as described in Appendix A for calculating the checksum.

3.2. Chunk Field Descriptions

The figure below illustrates the field format for the chunks to be transmitted in the SCTP packet. Each chunk is formatted with a Chunk Type field, a chunk-specific Flag field, a Chunk Length field, and a Value field.



Chunk Type: 8 bits (unsigned integer)

This field identifies the type of information contained in the Chunk Value field. It takes a value from 0 to 254. The value of 255 is reserved for future use as an extension field.

The values of Chunk Types are defined as follows:

ID Value	Chunk Type
0	Payload Data (DATA)
1	Initiation (INIT)
2	Initiation Acknowledgement (INIT ACK)
3	Selective Acknowledgement (SACK)
4	Heartbeat Request (HEARTBEAT)
5	Heartbeat Acknowledgement (HEARTBEAT ACK)
6	Abort (ABORT)
7	Shutdown (SHUTDOWN)
8	Shutdown Acknowledgement (SHUTDOWN ACK)
9	Operation Error (ERROR)
10	State Cookie (COOKIE ECHO)
11	Cookie Acknowledgement (COOKIE ACK)
12	Reserved for Explicit Congestion Notification Echo (ECNE)
13	Reserved for Congestion Window Reduced (CWR)
14	Shutdown Complete (SHUTDOWN COMPLETE)
15 to 62	available
63	reserved for IETF-defined Chunk Extensions
64 to 126	available
127	reserved for IETF-defined Chunk Extensions
128 to 190	available

191	reserved for IETF-defined Chunk Extensions
192 to 254	available
255	reserved for IETF-defined Chunk Extensions

Table 1: Chunk Types

Note: The ECNE and CWR chunk types are reserved for future use of Explicit Congestion Notification (ECN).

Chunk Types are encoded such that the highest-order 2 bits specify the action that is taken if the processing endpoint does not recognize the Chunk Type.

00	Stop processing this SCTP packet; discard the unrecognized chunk and all further chunks.
01	Stop processing this SCTP packet, discard the unrecognized chunk and all further chunks, and report the unrecognized chunk in an ERROR chunk using the 'Unrecognized Chunk Type' error cause.
10	Skip this chunk and continue processing.
11	Skip this chunk and continue processing, but report it in an ERROR chunk using the 'Unrecognized Chunk Type' error cause.

Table 2: Processing of Unknown Chunks

Chunk Flags: 8 bits

The usage of these bits depends on the Chunk type as given by the Chunk Type field. Unless otherwise specified, they are set to 0 on transmit and are ignored on receipt.

Chunk Length: 16 bits (unsigned integer)

This value represents the size of the chunk in bytes, including the Chunk Type, Chunk Flags, Chunk Length, and Chunk Value fields. Therefore, if the Chunk Value field is zero-length, the Length field will be set to 4. The Chunk Length field does not count any chunk padding. However, it does include any padding of variable-length parameters other than the last parameter in the chunk.

Note: A robust implementation is expected to accept the chunk whether or not the final padding has been included in the Chunk Length.

Chunk Value: variable length

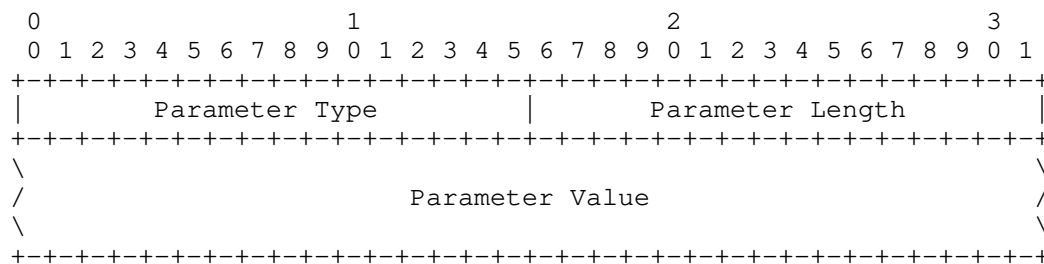
The Chunk Value field contains the actual information to be transferred in the chunk. The usage and format of this field is dependent on the Chunk Type.

The total length of a chunk (including Type, Length, and Value fields) MUST be a multiple of 4 bytes. If the length of the chunk is not a multiple of 4 bytes, the sender MUST pad the chunk with all zero bytes, and this padding is not included in the Chunk Length field. The sender MUST NOT pad with more than 3 bytes. The receiver MUST ignore the padding bytes.

SCTP-defined chunks are described in detail in Section 3.3. The guidelines for IETF-defined chunk extensions can be found in Section 15.1 of this document.

3.2.1. Optional/Variable-Length Parameter Format

Chunk values of SCTP control chunks consist of a chunk-type-specific header of required fields, followed by zero or more parameters. The optional and variable-length parameters contained in a chunk are defined in a Type-Length-Value format as shown below.



Parameter Type: 16 bits (unsigned integer)

The Type field is a 16-bit identifier of the type of parameter. It takes a value of 0 to 65534.

The value of 65535 is reserved for IETF-defined extensions. Values other than those defined in specific SCTP chunk descriptions are reserved for use by IETF.

Parameter Length: 16 bits (unsigned integer)

The Parameter Length field contains the size of the parameter in bytes, including the Parameter Type, Parameter Length, and Parameter Value fields. Thus, a parameter with a zero-length Parameter Value field would have a Parameter Length field of 4. The Parameter Length does not include any padding bytes.

Parameter Value: variable length

The Parameter Value field contains the actual information to be transferred in the parameter.

The total length of a parameter (including Parameter Type, Parameter Length, and Parameter Value fields) MUST be a multiple of 4 bytes. If the length of the parameter is not a multiple of 4 bytes, the sender pads the parameter at the end (i.e., after the Parameter Value field) with all zero bytes. The length of the padding is not included in the Parameter Length field. A sender MUST NOT pad with more than 3 bytes. The receiver MUST ignore the padding bytes.

The Parameter Types are encoded such that the highest-order 2 bits specify the action that is taken if the processing endpoint does not recognize the Parameter Type.

00	Stop processing this parameter; do not process any further parameters within this chunk.
01	Stop processing this parameter, do not process any further parameters within this chunk, and report the unrecognized parameter as described in Section 3.2.2.
10	Skip this parameter and continue processing.
11	Skip this parameter and continue processing but report the unrecognized parameter as described in Section 3.2.2.

Table 3: Processing of Unknown Parameters

Please note that, when an INIT or INIT ACK chunk is received, in all four cases, an INIT ACK or COOKIE ECHO chunk is sent in response, respectively. In the 00 or 01 case, the processing of the parameters after the unknown parameter is canceled, but no processing already done is rolled back.

The actual SCTP parameters are defined in the specific SCTP chunk sections. The rules for IETF-defined parameter extensions are defined in Section 15.3. Parameter types MUST be unique across all chunks. For example, the parameter type '5' is used to represent an IPv4 address (see Section 3.3.2.1). The value '5' then is reserved across all chunks to represent an IPv4 address and MUST NOT be reused with a different meaning in any other chunk.

3.2.2. Reporting of Unrecognized Parameters

If the receiver of an INIT chunk detects unrecognized parameters and has to report them according to Section 3.2.1, it MUST put the "Unrecognized Parameter" parameter(s) in the INIT ACK chunk sent in response to the INIT chunk. Note that if the receiver of the INIT chunk is not going to establish an association (e.g., due to lack of resources), an "Unrecognized Parameter" error cause would not be included with any ABORT chunk being sent to the sender of the INIT chunk.

If the receiver of any other chunk (e.g., INIT ACK) detects unrecognized parameters and has to report them according to Section 3.2.1, it SHOULD bundle the ERROR chunk containing the "Unrecognized Parameters" error cause with the chunk sent in response (e.g., COOKIE ECHO). If the receiver of an INIT ACK chunk cannot bundle the COOKIE ECHO chunk with the ERROR chunk, the ERROR chunk MAY be sent separately but not before the COOKIE ACK chunk has been received.

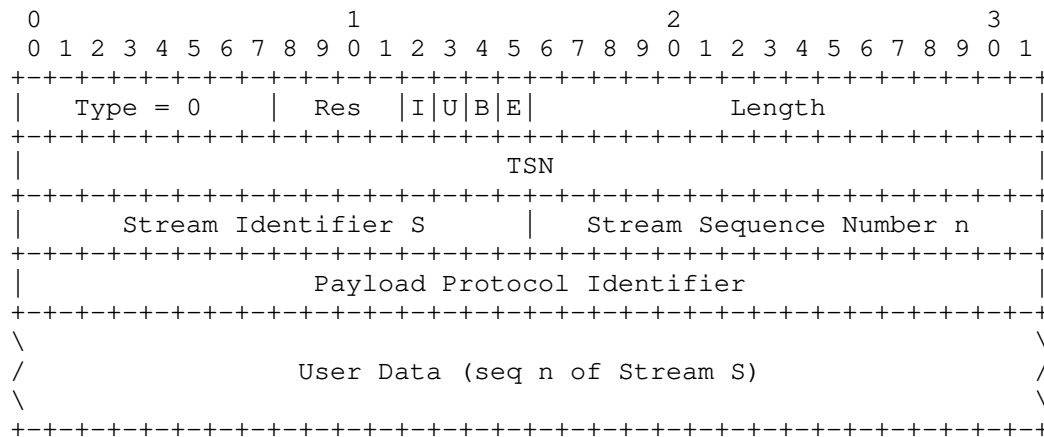
Any time a COOKIE ECHO chunk is sent in a packet, it MUST be the first chunk.

3.3. SCTP Chunk Definitions

This section defines the format of the different SCTP chunk types.

3.3.1. Payload Data (DATA) (0)

The following format MUST be used for the DATA chunk:



Res: 4 bits

All set to 0 on transmit and ignored on receipt.

I bit: 1 bit

The (I)mmEDIATE bit MAY be set by the sender whenever the sender of a DATA chunk can benefit from the corresponding SACK chunk being sent back without delay. See Section 4 of [RFC7053] for a discussion of the benefits.

U bit: 1 bit

The (U)NORDERED bit, if set to 1, indicates that this is an unordered DATA chunk, and there is no Stream Sequence Number assigned to this DATA chunk. Therefore, the receiver MUST ignore the Stream Sequence Number field.

After reassembly (if necessary), unordered DATA chunks MUST be dispatched to the upper layer by the receiver without any attempt to reorder.

If an unordered user message is fragmented, each fragment of the message MUST have its U bit set to 1.

B bit: 1 bit

The (B)EGINNING fragment bit, if set, indicates the first fragment of a user message.

E bit: 1 bit

The (E)NDING fragment bit, if set, indicates the last fragment of a user message.

Length: 16 bits (unsigned integer)

This field indicates the length of the DATA chunk in bytes from the beginning of the type field to the end of the User Data field excluding any padding. A DATA chunk with one byte of user data will have Length set to 17 (indicating 17 bytes).

A DATA chunk with a User Data field of length L will have the Length field set to $(16 + L)$ (indicating $16 + L$ bytes) where L MUST be greater than 0.

TSN: 32 bits (unsigned integer)

This value represents the TSN for this DATA chunk. The valid range of TSN is from 0 to 4294967295 ($2^{32} - 1$). TSN wraps back to 0 after reaching 4294967295.

Stream Identifier S: 16 bits (unsigned integer)

Identifies the stream to which the following user data belongs.

Stream Sequence Number n: 16 bits (unsigned integer)

This value represents the Stream Sequence Number of the following user data within the stream S. Valid range is 0 to 65535.

When a user message is fragmented by SCTP for transport, the same Stream Sequence Number MUST be carried in each of the fragments of the message.

Payload Protocol Identifier: 32 bits (unsigned integer)

This value represents an application (or upper layer) specified protocol identifier. This value is passed to SCTP by its upper layer and sent to its peer. This identifier is not used by SCTP but can be used by certain network entities, as well as by the peer application, to identify the type of information being carried in this DATA chunk. This field MUST be sent even in fragmented DATA chunks (to make sure it is available for agents in the middle of the network). Note that this field is not touched by an SCTP implementation; The upper layer is responsible for the host to network byte order conversion of this field.

The value 0 indicates that no application identifier is specified by the upper layer for this payload data.

User Data: variable length

This is the payload user data. The implementation MUST pad the end of the data to a 4-byte boundary with all-zero bytes. Any padding MUST NOT be included in the Length field. A sender MUST never add more than 3 bytes of padding.

An unfragmented user message **MUST** have both the B and E bits set to 1. Setting both B and E bits to 0 indicates a middle fragment of a multi-fragment user message, as summarized in the following table:

B	E	Description
1	0	First piece of a fragmented user message
0	0	Middle piece of a fragmented user message
0	1	Last piece of a fragmented user message
1	1	Unfragmented message

Table 4: Fragment Description Flags

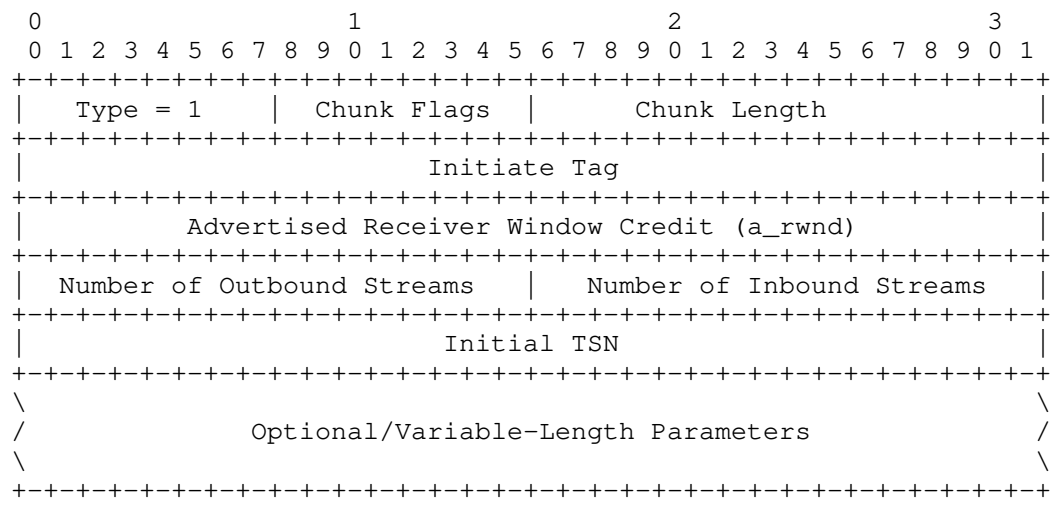
When a user message is fragmented into multiple chunks, the TSNs are used by the receiver to reassemble the message. This means that the TSNs for each fragment of a fragmented user message **MUST** be strictly sequential.

The TSNs of DATA chunks sent **SHOULD** be strictly sequential.

Note: The extension described in [RFC8260] can be used to mitigate the head of line blocking when transferring large user messages.

3.3.2. Initiation (INIT) (1)

This chunk is used to initiate an SCTP association between two endpoints. The format of the INIT chunk is shown below:



The following parameters are specified for the INIT chunk. Unless otherwise noted, each parameter MUST only be included once in the INIT chunk.

Fixed Length Parameter	Status
Initiate Tag	Mandatory
Advertised Receiver Window Credit	Mandatory
Number of Outbound Streams	Mandatory
Number of Inbound Streams	Mandatory
Initial TSN	Mandatory

Table 5: Fixed Length Parameters of INIT Chunks

Variable Length Parameter	Status	Type Value
IPv4 Address (Note 1)	Optional	5
IPv6 Address (Note 1)	Optional	6
Cookie Preservative	Optional	9
Reserved for ECN Capable (Note 2)	Optional	32768 (0x8000)
Host Name Address (Note 3)	Deprecated	11
Supported Address Types (Note 4)	Optional	12

Table 6: Variable Length Parameters of INIT Chunks

Note 1: The INIT chunks can contain multiple addresses that can be IPv4 and/or IPv6 in any combination.

Note 2: The ECN Capable field is reserved for future use of Explicit Congestion Notification.

Note 3: An INIT chunk MUST NOT contain the Host Name Address parameter. The receiver of an INIT chunk containing a Host Name Address parameter MUST send an ABORT chunk and MAY include an "Unresolvable Address" error cause.

Note 4: This parameter, when present, specifies all the address types the sending endpoint can support. The absence of this parameter indicates that the sending endpoint can support any address type.

If an INIT chunk is received with all mandatory parameters that are specified for the INIT chunk, then the receiver SHOULD process the INIT chunk and send back an INIT ACK. The receiver of the INIT chunk MAY bundle an ERROR chunk with the COOKIE ACK chunk later. However, restrictive implementations MAY send back an ABORT chunk in response to the INIT chunk.

The Chunk Flags field in INIT chunks is reserved, and all bits in it SHOULD be set to 0 by the sender and ignored by the receiver.

Initiate Tag: 32 bits (unsigned integer)

The receiver of the INIT chunk (the responding end) records the value of the Initiate Tag parameter. This value MUST be placed into the Verification Tag field of every SCTP packet that the receiver of the INIT chunk transmits within this association.

The Initiate Tag is allowed to have any value except 0. See Section 5.3.1 for more on the selection of the tag value.

If the value of the Initiate Tag in a received INIT chunk is found to be 0, the receiver MUST silently discard the packet.

Advertised Receiver Window Credit (a_rwnd): 32 bits (unsigned integer)

This value represents the dedicated buffer space, in number of bytes, the sender of the INIT chunk has reserved in association with this window.

The Advertised Receiver Window Credit MUST NOT be smaller than 1500.

A receiver of an INIT chunk with the a_rwnd value set to a value smaller than 1500 MUST discard the packet, SHOULD send a packet in response containing an ABORT chunk and using the Initiate Tag as the Verification Tag, and MUST NOT change the state of any existing association.

During the life of the association, this buffer space SHOULD NOT be reduced (i.e., dedicated buffers ought not to be taken away from this association); however, an endpoint MAY change the value of a_rwnd it sends in SACK chunks.

Number of Outbound Streams (OS): 16 bits (unsigned integer)

Defines the number of outbound streams the sender of this INIT chunk wishes to create in this association. The value of 0 MUST NOT be used.

A receiver of an INIT chunk with the OS value set to 0 MUST discard the packet, SHOULD send a packet in response containing an ABORT chunk and using the Initiate Tag as the Verification Tag, and MUST NOT change the state of any existing association.

Number of Inbound Streams (MIS): 16 bits (unsigned integer)

Defines the maximum number of streams the sender of this INIT chunk allows the peer end to create in this association. The value 0 MUST NOT be used.

Note: There is no negotiation of the actual number of streams but instead the two endpoints will use the min(requested, offered). See Section 5.1.1 for details.

A receiver of an INIT chunk with the MIS value set to 0 MUST discard the packet, SHOULD send a packet in response containing an ABORT chunk and using the Initiate Tag as the Verification Tag, and MUST NOT change the state of any existing association.

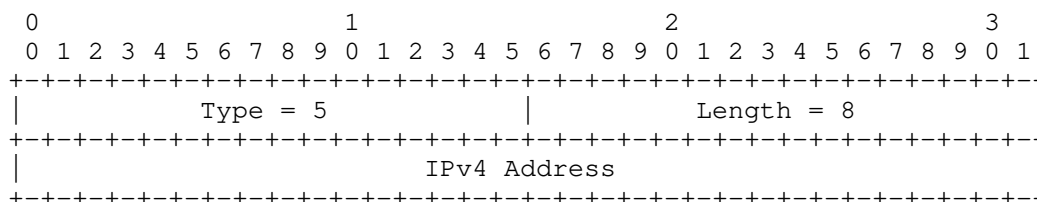
Initial TSN (I-TSN): 32 bits (unsigned integer)

Defines the initial TSN that the sender of the INIT chunk will use. The valid range is from 0 to 4294967295 and the Initial TSN SHOULD be set to a random value in that range. The methods described in [RFC4086] can be used for the Initial TSN randomization.

3.3.2.1. Optional or Variable-Length Parameters in INIT chunks

The following parameters follow the Type-Length-Value format as defined in Section 3.2.1. Any Type-Length-Value fields MUST be placed after the fixed-length fields. (The fixed-length fields are defined in the previous section.)

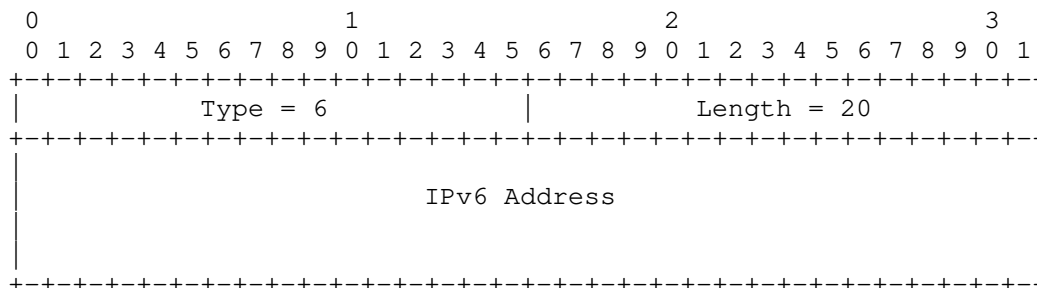
3.3.2.1.1. IPv4 Address (5)



IPv4 Address: 32 bits (unsigned integer)

Contains an IPv4 address of the sending endpoint. It is binary encoded.

3.3.2.1.2. IPv6 Address (6)



IPv6 Address: 128 bits (unsigned integer)

Contains an IPv6 [RFC8200] address of the sending endpoint. It is binary encoded.

A sender **MUST NOT** use an IPv4-mapped IPv6 address [RFC4291], but **SHOULD** instead use an IPv4 Address parameter for an IPv4 address.

Combined with the Source Port Number in the SCTP common header, the value passed in an IPv4 or IPv6 Address parameter indicates a transport address the sender of the INIT chunk will support for the association being initiated. That is, during the life time of this association, this IP address can appear in the source address field of an IP datagram sent from the sender of the INIT chunk, and can be used as a destination address of an IP datagram sent from the receiver of the INIT chunk.

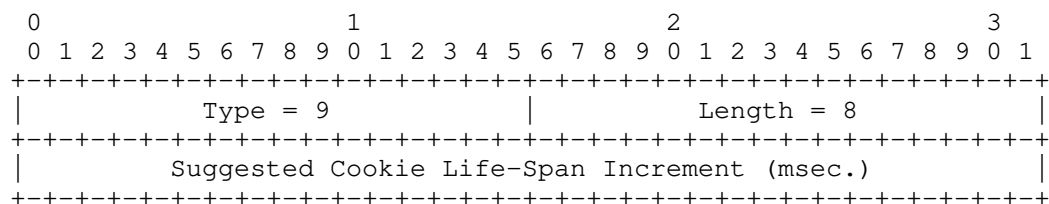
More than one IP Address parameter can be included in an INIT chunk when the sender of the INIT chunk is multi-homed. Moreover, a multi-homed endpoint might have access to different types of network; thus, more than one address type can be present in one INIT chunk, i.e., IPv4 and IPv6 addresses are allowed in the same INIT chunk.

If the INIT chunk contains at least one IP Address parameter, then the source address of the IP datagram containing the INIT chunk and any additional address(es) provided within the INIT can be used as destinations by the endpoint receiving the INIT chunk. If the INIT chunk does not contain any IP Address parameters, the endpoint receiving the INIT chunk **MUST** use the source address associated with the received IP datagram as its sole destination address for the association.

Note that not using any IP Address parameters in the INIT and INIT ACK chunk is a way to make an association more likely to work in combination with Network Address Translation (NAT).

3.3.2.1.3. Cookie Preservative (9)

The sender of the INIT chunk uses this parameter to suggest to the receiver of the INIT chunk a longer life-span for the State Cookie.



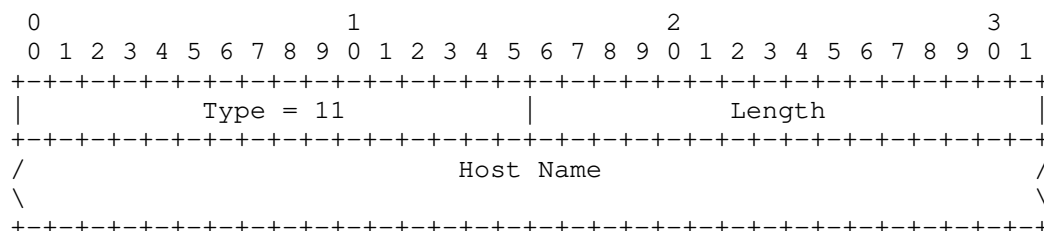
Suggested Cookie Life-Span Increment: 32 bits (unsigned integer)

This parameter indicates to the receiver how much increment in milliseconds the sender wishes the receiver to add to its default cookie life-span.

This optional parameter MAY be added to the INIT chunk by the sender when it reattempts establishing an association with a peer to which its previous attempt of establishing the association failed due to a stale cookie operation error. The receiver MAY choose to ignore the suggested cookie life-span increase for its own security reasons.

3.3.2.1.4. Host Name Address (11)

The sender of an INIT chunk or INIT ACK chunk MUST NOT include this parameter. The usage of the Host Name Address parameter is deprecated. The receiver of an INIT chunk or an INIT ACK containing a Host Name Address parameter MUST send an ABORT chunk and MAY include an "Unresolvable Address" error cause.



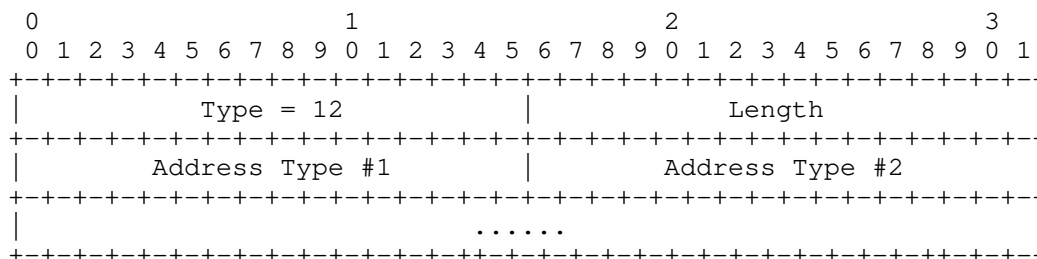
Host Name: variable length

This field contains a host name in "host name syntax" per Section 2.1 of [RFC1123]. The method for resolving the host name is out of scope of SCTP.

At least one null terminator is included in the Host Name string and MUST be included in the length.

3.3.2.1.5. Supported Address Types (12)

The sender of INIT chunk uses this parameter to list all the address types it can support.

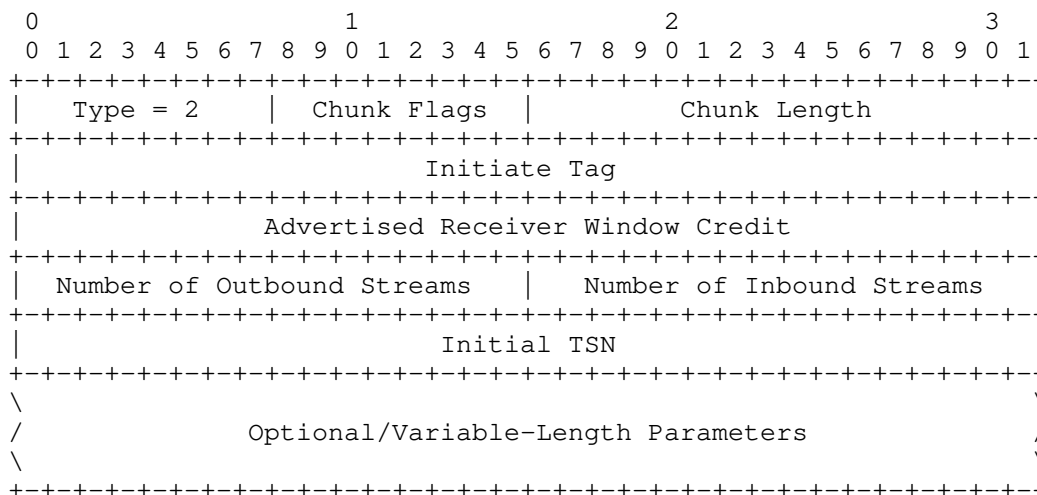


Address Type: 16 bits (unsigned integer)

This is filled with the type value of the corresponding address TLV (e.g., 5 for indicating IPv4, 6 for indicating IPv6). The value indicating the Host Name Address parameter MUST NOT be used when sending this parameter and MUST be ignored when receiving this parameter.

3.3.3. Initiation Acknowledgement (INIT ACK) (2)

The INIT ACK chunk is used to acknowledge the initiation of an SCTP association. The format of the INIT ACK chunk is shown below:



The parameter part of INIT ACK is formatted similarly to the INIT chunk. The following parameters are specified for the INIT ACK chunk:

Fixed Length Parameter	Status
Initiate Tag	Mandatory
Advertised Receiver Window Credit	Mandatory
Number of Outbound Streams	Mandatory
Number of Inbound Streams	Mandatory
Initial TSN	Mandatory

Table 7: Fixed Length Parameters of INIT ACK Chunks

It uses two extra variable parameters: The State Cookie and the Unrecognized Parameter:

Variable Length Parameter	Status	Type Value
State Cookie	Mandatory	7
IPv4 Address (Note 1)	Optional	5
IPv6 Address (Note 1)	Optional	6
Unrecognized Parameter	Optional	8
Reserved for ECN Capable (Note 2)	Optional	32768 (0x8000)
Host Name Address (Note 3)	Deprecated	11

Table 8: Variable Length Parameters of INIT ACK Chunks

Note 1: The INIT ACK chunks can contain any number of IP address parameters that can be IPv4 and/or IPv6 in any combination.

Note 2: The ECN Capable field is reserved for future use of Explicit Congestion Notification.

Note 3: An INIT ACK chunk MUST NOT contain the Host Name Address parameter. The receiver of INIT ACK chunks containing a Host Name Address parameter MUST send an ABORT chunk and MAY include an "Unresolvable Address" error cause.

The Chunk Flags field in INIT ACK chunks is reserved, and all bits in it SHOULD be set to 0 by the sender and ignored by the receiver.

Initiate Tag: 32 bits (unsigned integer)

The receiver of the INIT ACK chunk records the value of the Initiate Tag parameter. This value MUST be placed into the Verification Tag field of every SCTP packet that the receiver of the INIT ACK chunk transmits within this association.

The Initiate Tag MUST NOT take the value 0. See Section 5.3.1 for more on the selection of the Initiate Tag value.

If an endpoint in the COOKIE-WAIT state receives an INIT ACK chunk with the Initiate Tag set to 0, it MUST destroy the TCB and SHOULD send an ABORT chunk with the T bit set. If such an INIT-ACK chunk is received in any state other than CLOSED or COOKIE-WAIT, it SHOULD be discarded silently (see Section 5.2.3).

Advertised Receiver Window Credit (a_rwnd): 32 bits (unsigned integer)

This value represents the dedicated buffer space, in number of bytes, the sender of the INIT ACK chunk has reserved in association with this window.

The Advertised Receiver Window Credit MUST NOT be smaller than 1500.

A receiver of an INIT ACK chunk with the a_rwnd value set to a value smaller than 1500 MUST discard the packet, SHOULD send a packet in response containing an ABORT chunk and using the Initiate Tag as the Verification Tag, and MUST NOT change the state of any existing association.

During the life of the association, this buffer space SHOULD NOT be reduced (i.e., dedicated buffers ought not to be taken away from this association); however, an endpoint MAY change the value of a_rwnd it sends in SACK chunks.

Number of Outbound Streams (OS): 16 bits (unsigned integer)

Defines the number of outbound streams the sender of this INIT ACK chunk wishes to create in this association. The value of 0 MUST NOT be used, and the value MUST NOT be greater than the MIS value sent in the INIT chunk.

If an endpoint in the COOKIE-WAIT state receives an INIT ACK chunk with the OS value set to 0, it MUST destroy the TCB and SHOULD send an ABORT chunk. If such an INIT-ACK chunk is received in any state other than CLOSED or COOKIE-WAIT, it SHOULD be discarded silently (see Section 5.2.3).

Number of Inbound Streams (MIS): 16 bits (unsigned integer)

Defines the maximum number of streams the sender of this INIT ACK chunk allows the peer end to create in this association. The value 0 MUST NOT be used.

Note: There is no negotiation of the actual number of streams but instead the two endpoints will use the min(requested, offered). See Section 5.1.1 for details.

If an endpoint in the COOKIE-WAIT state receives an INIT ACK chunk with the MIS value set to 0, it MUST destroy the TCB and SHOULD send an ABORT chunk. If such an INIT-ACK chunk is received in any state other than CLOSED or COOKIE-WAIT, it SHOULD be discarded silently (see Section 5.2.3).

Initial TSN (I-TSN): 32 bits (unsigned integer)

Defines the initial TSN that the sender of the INIT ACK chunk will use. The valid range is from 0 to 4294967295 and the Initial TSN SHOULD be set to a random value in that range. The methods described in [RFC4086] can be used for the Initial TSN randomization.

Implementation Note: An implementation MUST be prepared to receive an INIT ACK chunk that is quite large (more than 1500 bytes) due to the variable size of the State Cookie and the variable address list. For example if a responder to the INIT chunk has 1000 IPv4 addresses it wishes to send, it would need at least 8,000 bytes to encode this in the INIT ACK chunk.

If an INIT ACK chunk is received with all mandatory parameters that are specified for the INIT ACK chunk, then the receiver SHOULD process the INIT ACK chunk and send back a COOKIE ECHO chunk. The receiver of the INIT ACK chunk MAY bundle an ERROR chunk with the COOKIE ECHO chunk. However, restrictive implementations MAY send back an ABORT chunk in response to the INIT ACK chunk.

In combination with the Source Port carried in the SCTP common header, each IP Address parameter in the INIT ACK chunk indicates to the receiver of the INIT ACK chunk a valid transport address supported by the sender of the INIT ACK chunk for the life time of the association being initiated.

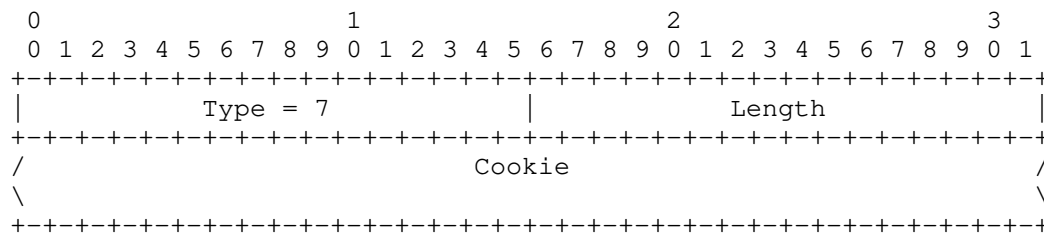
If the INIT ACK chunk contains at least one IP Address parameter, then the source address of the IP datagram containing the INIT ACK chunk and any additional address(es) provided within the INIT ACK chunk MAY be used as destinations by the receiver of the INIT ACK chunk. If the INIT ACK chunk does not contain any IP Address parameters, the receiver of the INIT ACK chunk MUST use the source address associated with the received IP datagram as its sole destination address for the association.

The State Cookie and Unrecognized Parameters use the Type-Length-Value format as defined in Section 3.2.1 and are described below. The other fields are defined the same as their counterparts in the INIT chunk.

3.3.3.1. Optional or Variable-Length Parameters in INIT ACK chunks

The State Cookie and Unrecognized Parameters use the Type-Length-Value format as defined in Section 3.2.1 and are described below. The IPv4 Address Parameter is described in Section 3.3.2.1.1, and the IPv6 Address Parameter is described in Section 3.3.2.1.2. The Host Name Address Parameter is described in Section 3.3.2.1.4 and MUST NOT be included in an INIT ACK chunk. Any Type-Length-Value fields MUST be placed after the fixed-length fields. (The fixed-length fields are defined in the previous section.)

3.3.3.1.1. State Cookie (7)

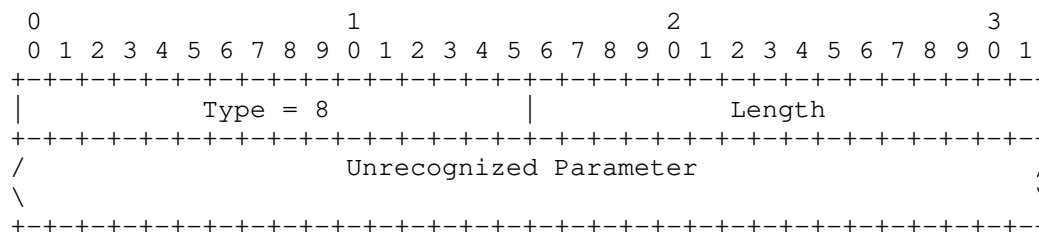


Cookie: variable length

This parameter value MUST contain all the necessary state and parameter information required for the sender of this INIT ACK chunk to create the association, along with a Message Authentication Code (MAC). See Section 5.1.3 for details on State Cookie definition.

3.3.3.1.2. Unrecognized Parameter (8)

This parameter is returned to the originator of the INIT chunk when the INIT chunk contains an unrecognized parameter that has a type that indicates it SHOULD be reported to the sender.



Unrecognized Parameter: variable length

The parameter value field will contain an unrecognized parameter copied from the INIT chunk complete with Parameter Type, Length, and Value fields.

3.3.4. Selective Acknowledgement (SACK) (3)

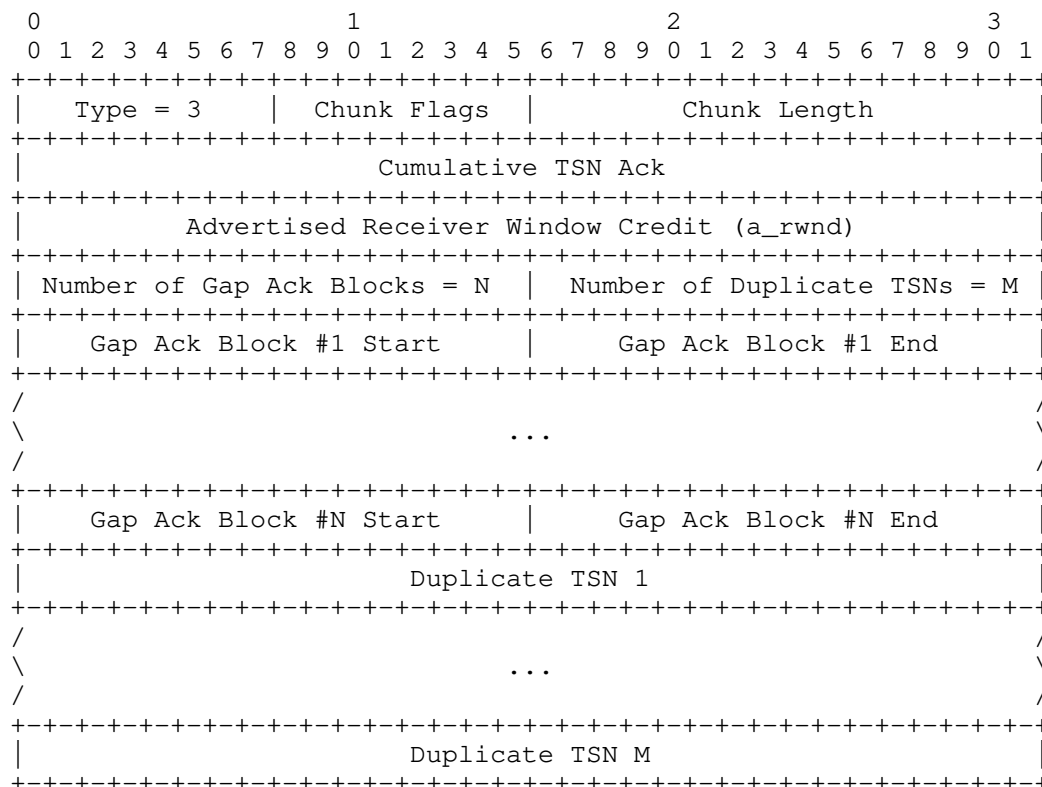
This chunk is sent to the peer endpoint to acknowledge received DATA chunks and to inform the peer endpoint of gaps in the received subsequences of DATA chunks as represented by their TSNs.

The SACK chunk MUST contain the Cumulative TSN Ack, Advertised Receiver Window Credit (`a_rwnd`), Number of Gap Ack Blocks, and Number of Duplicate TSNs fields.

By definition, the value of the Cumulative TSN Ack parameter is the last TSN received before a break in the sequence of received TSNs occurs; the next TSN value following this one has not yet been received at the endpoint sending the SACK chunk. This parameter therefore acknowledges receipt of all TSNs less than or equal to its value.

The handling of `a_rwnd` by the receiver of the SACK chunk is discussed in detail in Section 6.2.1.

The SACK chunk also contains zero or more Gap Ack Blocks. Each Gap Ack Block acknowledges a subsequence of TSNs received following a break in the sequence of received TSNs. The Gap Ack Blocks SHOULD be isolated. This means that the TSN just before each Gap Ack Block and the TSN just after each Gap Ack Block have not been received. By definition, all TSNs acknowledged by Gap Ack Blocks are greater than the value of the Cumulative TSN Ack.



Chunk Flags: 8 bits

All set to 0 on transmit and ignored on receipt.

Cumulative TSN Ack: 32 bits (unsigned integer)

The largest TSN, such that all TSNs smaller than or equal to it have been received and the next one has not been received. In the case where no DATA chunk has been received, this value is set to the peer's Initial TSN minus one.

Advertised Receiver Window Credit (a_rwnd): 32 bits (unsigned integer)

This field indicates the updated receive buffer space in bytes of the sender of this SACK chunk; see Section 6.2.1 for details.

Number of Gap Ack Blocks: 16 bits (unsigned integer)

Indicates the number of Gap Ack Blocks included in this SACK chunk.

Number of Duplicate TSNs: 16 bit

This field contains the number of duplicate TSNs the endpoint has received. Each duplicate TSN is listed following the Gap Ack Block list.

Gap Ack Blocks:

These fields contain the Gap Ack Blocks. They are repeated for each Gap Ack Block up to the number of Gap Ack Blocks defined in the Number of Gap Ack Blocks field. All DATA chunks with TSNs greater than or equal to (Cumulative TSN Ack + Gap Ack Block Start) and less than or equal to (Cumulative TSN Ack + Gap Ack Block End) of each Gap Ack Block are assumed to have been received correctly.

Gap Ack Block Start: 16 bits (unsigned integer)

Indicates the Start offset TSN for this Gap Ack Block. To calculate the actual TSN number the Cumulative TSN Ack is added to this offset number. This calculated TSN identifies the lowest TSN in this Gap Ack Block that has been received.

Gap Ack Block End: 16 bits (unsigned integer)

Indicates the End offset TSN for this Gap Ack Block. To calculate the actual TSN number, the Cumulative TSN Ack is added to this offset number. This calculated TSN identifies the highest TSN in this Gap Ack Block that has been received.

For example, assume that the receiver has the following DATA chunks newly arrived at the time when it decides to send a Selective ACK,

```

-----
| TSN = 17 |
-----
|           | <- still missing
-----
| TSN = 15 |
-----
| TSN = 14 |
-----
|           | <- still missing
-----
| TSN = 12 |
-----
| TSN = 11 |
-----
| TSN = 10 |
-----

```

then the parameter part of the SACK chunk MUST be constructed as follows (assuming the new a_rwnd is set to 4660 by the sender):

Cumulative TSN Ack = 12															
a_rwnd = 4660															
num of block = 2								num of dup = 0							
block #1 start = 2								block #1 end = 3							
block #2 start = 5								block #2 end = 5							

Duplicate TSN: 32 bits (unsigned integer)

Indicates the number of times a TSN was received in duplicate since the last SACK chunk was sent. Every time a receiver gets a duplicate TSN (before sending the SACK chunk), it adds it to the list of duplicates. The duplicate count is reinitialized to zero after sending each SACK chunk.

For example, if a receiver were to get the TSN 19 three times it would list 19 twice in the outbound SACK chunk. After sending the SACK chunk, if it received yet one more TSN 19 it would list 19 as a duplicate once in the next outgoing SACK chunk.

3.3.5. Heartbeat Request (HEARTBEAT) (4)

An endpoint SHOULD send a HEARTBEAT (HB) chunk to its peer endpoint to probe the reachability of a particular destination transport address defined in the present association.

The parameter field contains the Heartbeat Information, which is a variable-length opaque data structure understood only by the sender.

0																1																2																3															
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1																																
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+																																																															
Type = 4								Chunk Flags								Heartbeat Length																																															
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+																																																															
\																																																															
/ Heartbeat Information TLV (Variable-Length) /																																																															
\																																																															
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+																																																															

Chunk Flags: 8 bits

Set to 0 on transmit and ignored on receipt.

Heartbeat Length: 16 bits (unsigned integer)
Set to the size of the chunk in bytes, including the chunk header and the Heartbeat Information field.

Heartbeat Information: variable length
Defined as a variable-length parameter using the format described in Section 3.2.1, i.e.:

Variable Parameters	Status	Type Value
Heartbeat Info	Mandatory	1

Table 9: Variable Length Parameters of HEARTBEAT Chunks

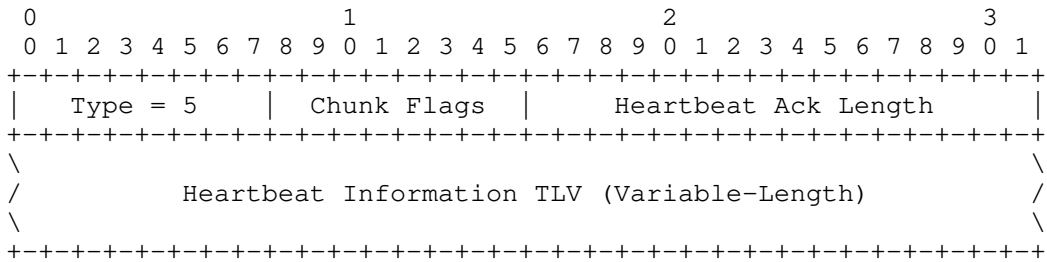
										1										2										3																													
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1																												
+										+										+										+										+																			
										Heartbeat Info Type = 1																				HB Info Length																													
+										+										+										+										+										+									
/										Sender-Specific Heartbeat Info										/																																							
\																																								\																			
+										+										+										+										+										+									

The Sender-Specific Heartbeat Info field SHOULD include information about the sender’s current time when this HEARTBEAT chunk is sent and the destination transport address to which this HEARTBEAT chunk is sent (see Section 8.3). This information is simply reflected back by the receiver in the HEARTBEAT ACK chunk (see Section 3.3.6). Note also that the HEARTBEAT chunk is both for reachability checking and for path verification (see Section 5.4). When a HEARTBEAT chunk is being used for path verification purposes, it MUST include a random nonce of length 64-bit or longer ([RFC4086] provides some information on randomness guidelines).

3.3.6. Heartbeat Acknowledgement (HEARTBEAT ACK) (5)

An endpoint MUST send this chunk to its peer endpoint as a response to a HEARTBEAT chunk (see Section 8.3). A packet containing the HEARTBEAT ACK chunk is always sent to the source IP address of the IP datagram containing the HEARTBEAT chunk to which this HEARTBEAT ACK chunk is responding.

The parameter field contains a variable-length opaque data structure.



Chunk Flags: 8 bits
Set to 0 on transmit and ignored on receipt.

Heartbeat Ack Length: 16 bits (unsigned integer)
Set to the size of the chunk in bytes, including the chunk header and the Heartbeat Information field.

Heartbeat Information: variable length
This field MUST contain the Heartbeat Info parameter (as defined in Section 3.3.5) of the Heartbeat Request to which this Heartbeat Acknowledgement is responding.

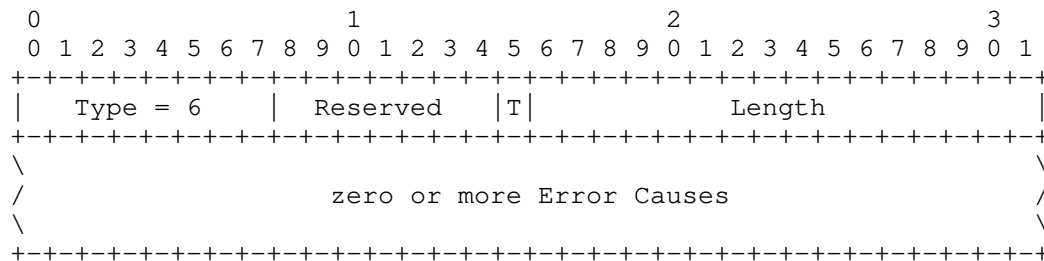
Variable Parameters	Status	Type Value
Heartbeat Info	Mandatory	1

Table 10: Variable Length Parameters of HEARTBEAT ACK Chunks

3.3.7. Abort Association (ABORT) (6)

The ABORT chunk is sent to the peer of an association to close the association. The ABORT chunk MAY contain Cause Parameters to inform the receiver about the reason of the abort. DATA chunks MUST NOT be bundled with ABORT chunks. Control chunks (except for INIT, INIT ACK, and SHUTDOWN COMPLETE) MAY be bundled with an ABORT chunk, but they MUST be placed before the ABORT chunk in the SCTP packet, otherwise they will be ignored by the receiver.

If an endpoint receives an ABORT chunk with a format error or no TCB is found, it MUST silently discard it. Moreover, under any circumstances, an endpoint that receives an ABORT chunk MUST NOT respond to that ABORT chunk by sending an ABORT chunk of its own.



Chunk Flags: 8 bits

Reserved: 7 bits

Set to 0 on transmit and ignored on receipt.

T bit: 1 bit

The T bit is set to 0 if the sender filled in the Verification Tag expected by the peer. If the Verification Tag is reflected, the T bit MUST be set to 1. Reflecting means that the sent Verification Tag is the same as the received one.

Length: 16 bits (unsigned integer)

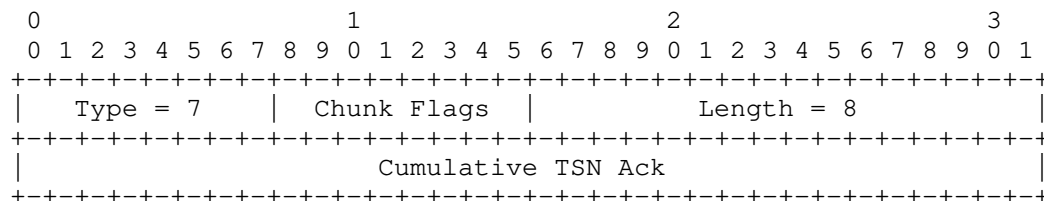
Set to the size of the chunk in bytes, including the chunk header and all the Error Cause fields present.

See Section 3.3.10 for Error Cause definitions.

Note: Special rules apply to this chunk for verification; please see Section 8.5.1 for details.

3.3.8. Shutdown Association (SHUTDOWN) (7)

An endpoint in an association MUST use this chunk to initiate a graceful close of the association with its peer. This chunk has the following format.

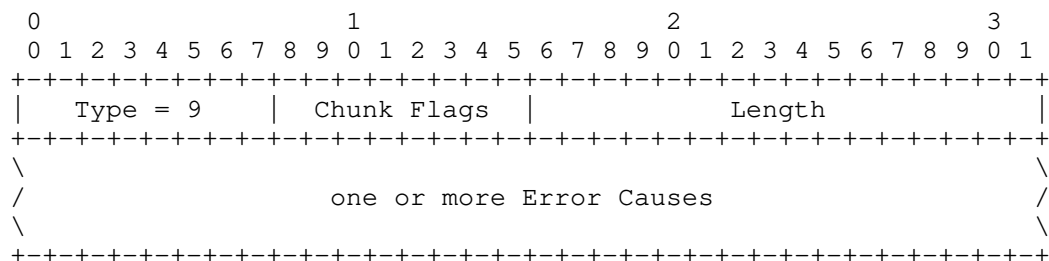


Chunk Flags: 8 bits

Set to 0 on transmit and ignored on receipt.

Length: 16 bits (unsigned integer)

Indicates the length of the parameter. Set to 8.



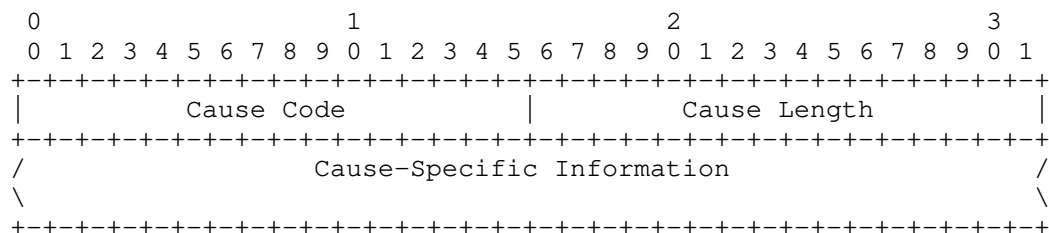
Chunk Flags: 8 bits

Set to 0 on transmit and ignored on receipt.

Length: 16 bits (unsigned integer)

Set to the size of the chunk in bytes, including the chunk header and all the Error Cause fields present.

Error causes are defined as variable-length parameters using the format described in Section 3.2.1, that is:



Cause Code: 16 bits (unsigned integer)

Defines the type of error conditions being reported.

Value	Cause Code
1	Invalid Stream Identifier
2	Missing Mandatory Parameter
3	Stale Cookie Error
4	Out of Resource
5	Unresolvable Address
6	Unrecognized Chunk Type
7	Invalid Mandatory Parameter
8	Unrecognized Parameters
9	No User Data
10	Cookie Received While Shutting Down
11	Restart of an Association with New Addresses
12	User Initiated Abort
13	Protocol Violation

Table 11: Cause Code

Cause Length: 16 bits (unsigned integer)

Set to the size of the parameter in bytes, including the Cause Code, Cause Length, and Cause-Specific Information fields.

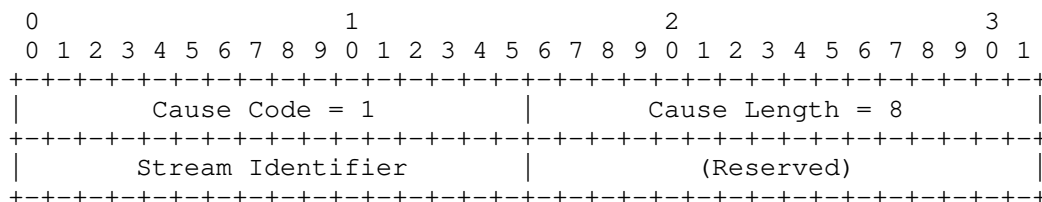
Cause-Specific Information: variable length

This field carries the details of the error condition.

Section 3.3.10.1 – Section 3.3.10.13 define error causes for SCTP. Guidelines for the IETF to define new error cause values are discussed in Section 15.4.

3.3.10.1. Invalid Stream Identifier (1)

Indicates that the endpoint received a DATA chunk sent using a nonexistent stream.



Stream Identifier: 16 bits (unsigned integer)

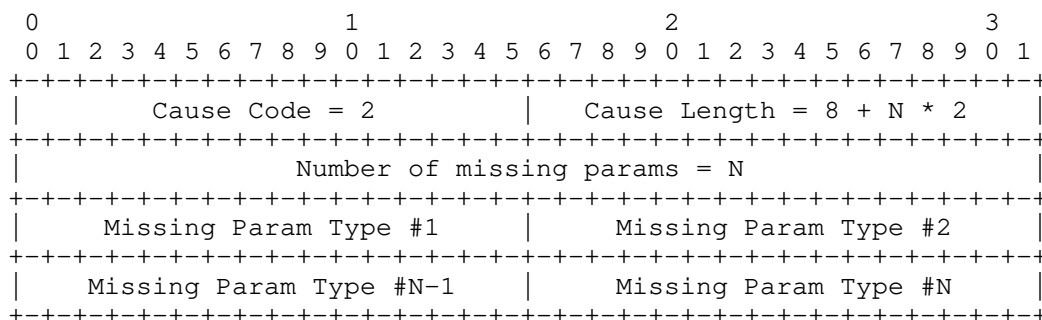
Contains the Stream Identifier of the DATA chunk received in error.

Reserved: 16 bits

This field is reserved. It is set to all 0's on transmit and ignored on receipt.

3.3.10.2. Missing Mandatory Parameter (2)

Indicates that one or more mandatory TLV parameters are missing in a received INIT or INIT ACK chunk.



Number of Missing params: 32 bits (unsigned integer)

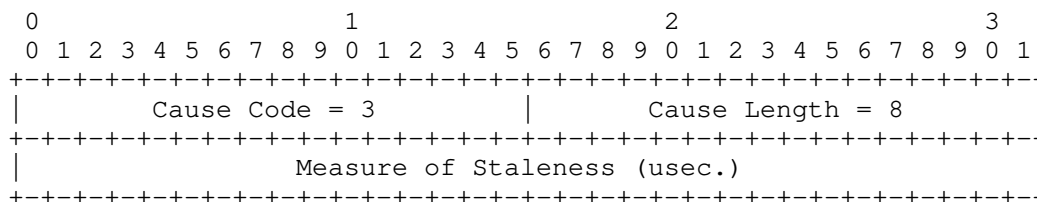
This field contains the number of parameters contained in the Cause-Specific Information field.

Missing Param Type: 16 bits (unsigned integer)

Each field will contain the missing mandatory parameter number.

3.3.10.3. Stale Cookie Error (3)

Indicates the receipt of a valid State Cookie that has expired.



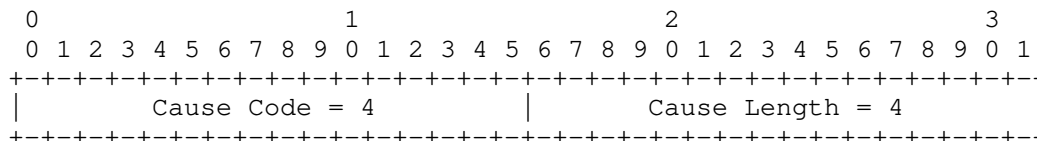
Measure of Staleness: 32 bits (unsigned integer)

This field contains the difference, rounded up in microseconds, between the current time and the time the State Cookie expired.

The sender of this error cause MAY choose to report how long past expiration the State Cookie is by including a non-zero value in the Measure of Staleness field. If the sender does not wish to provide the Measure of Staleness, it SHOULD set this field to the value of zero.

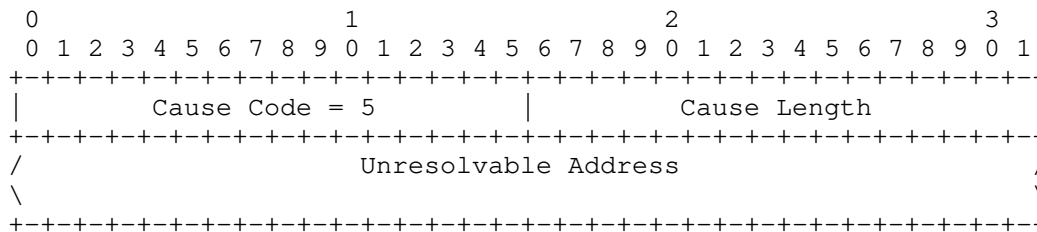
3.3.10.4. Out of Resource (4)

Indicates that the sender is out of resource. This is usually sent in combination with or within an ABORT chunk.



3.3.10.5. Unresolvable Address (5)

Indicates that the sender is not able to resolve the specified address parameter (e.g., type of address is not supported by the sender). This is usually sent in combination with or within an ABORT chunk.

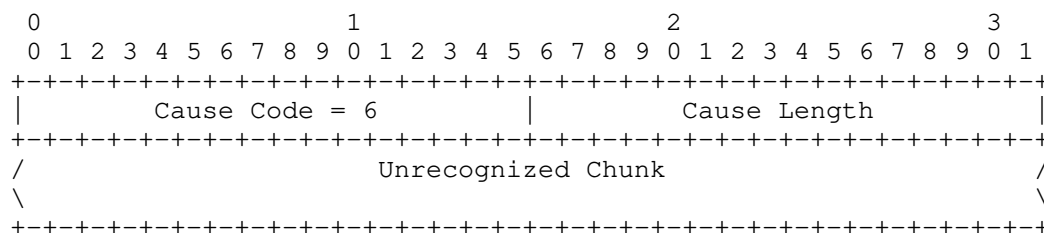


Unresolvable Address: variable length

The Unresolvable Address field contains the complete Type, Length, and Value of the address parameter (or Host Name parameter) that contains the unresolvable address or host name.

3.3.10.6. Unrecognized Chunk Type (6)

This error cause is returned to the originator of the chunk if the receiver does not understand the chunk and the upper bits of the 'Chunk Type' are set to 01 or 11.

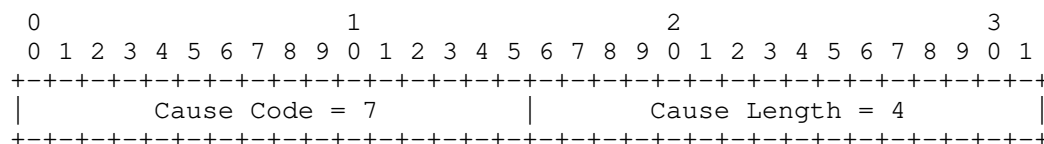


Unrecognized Chunk: variable length

The Unrecognized Chunk field contains the unrecognized chunk from the SCTP packet complete with Chunk Type, Chunk Flags, and Chunk Length.

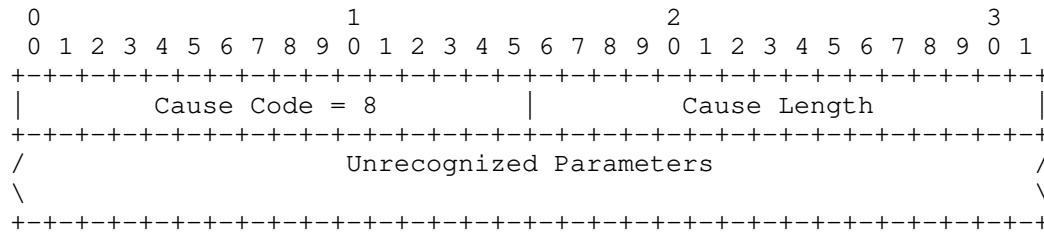
3.3.10.7. Invalid Mandatory Parameter (7)

This error cause is returned to the originator of an INIT or INIT ACK chunk when one of the mandatory parameters is set to an invalid value.



3.3.10.8. Unrecognized Parameters (8)

This error cause is returned to the originator of the INIT ACK chunk if the receiver does not recognize one or more Optional TLV parameters in the INIT ACK chunk.

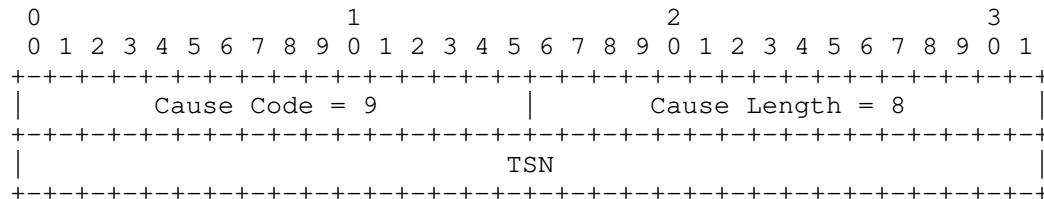


Unrecognized Parameters: variable length

The Unrecognized Parameters field contains the unrecognized parameters copied from the INIT ACK chunk complete with TLV. This error cause is normally contained in an ERROR chunk bundled with the COOKIE ECHO chunk when responding to the INIT ACK chunk, when the sender of the COOKIE ECHO chunk wishes to report unrecognized parameters.

3.3.10.9. No User Data (9)

This error cause is returned to the originator of a DATA chunk if a received DATA chunk has no user data.



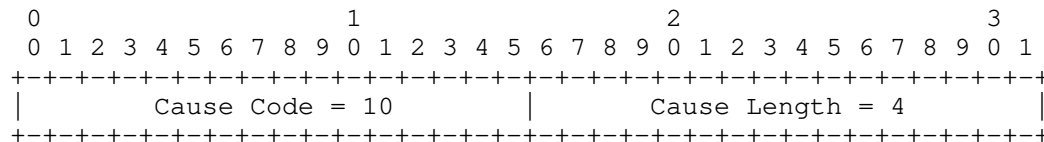
TSN: 32 bits (unsigned integer)

This parameter contains the TSN of the DATA chunk received with no user data field.

This cause code is normally returned in an ABORT chunk (see Section 6.2).

3.3.10.10. Cookie Received While Shutting Down (10)

A COOKIE ECHO chunk was received while the endpoint was in the SHUTDOWN-ACK-SENT state. This error is usually returned in an ERROR chunk bundled with the retransmitted SHUTDOWN ACK chunk.



3.3.10.11. Restart of an Association with New Addresses (11)

An INIT chunk was received on an existing association. But the INIT chunk added addresses to the association that were previously not part of the association. The new addresses are listed in the error cause. This error cause is normally sent as part of an ABORT chunk refusing the INIT chunk (see Section 5.2).

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|               Cause Code = 11               | Cause Length |
+-----+-----+-----+-----+-----+-----+-----+-----+
/                               New Address TLVs                               /
\                               /
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Note: Each New Address TLV is an exact copy of the TLV that was found in the INIT chunk that was new, including the Parameter Type and the Parameter Length.

3.3.10.12. User-Initiated Abort (12)

This error cause MAY be included in ABORT chunks that are sent because of an upper-layer request. The upper layer can specify an Upper Layer Abort Reason that is transported by SCTP transparently and MAY be delivered to the upper-layer protocol at the peer.

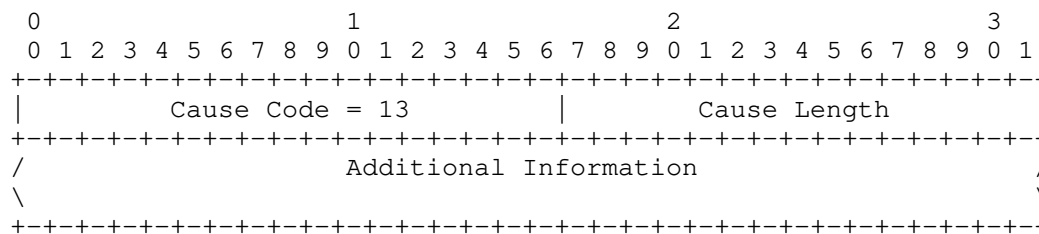
```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|               Cause Code = 12               | Cause Length |
+-----+-----+-----+-----+-----+-----+-----+-----+
/                               Upper Layer Abort Reason                               /
\                               /
+-----+-----+-----+-----+-----+-----+-----+-----+

```

3.3.10.13. Protocol Violation (13)

This error cause MAY be included in ABORT chunks that are sent because an SCTP endpoint detects a protocol violation of the peer that is not covered by the error causes described in Section 3.3.10.1 to Section 3.3.10.12. An implementation MAY provide additional information specifying what kind of protocol violation has been detected.



3.3.11. Cookie Echo (COOKIE ECHO) (10)

This chunk is used only during the initialization of an association. It is sent by the initiator of an association to its peer to complete the initialization process. This chunk **MUST** precede any DATA chunk sent within the association, but **MAY** be bundled with one or more DATA chunks in the same packet.



Chunk Flags: 8 bits

Set to 0 on transmit and ignored on receipt.

Length: 16 bits (unsigned integer)

Set to the size of the chunk in bytes, including the 4 bytes of the chunk header and the size of the cookie.

Cookie: variable size

This field **MUST** contain the exact cookie received in the State Cookie parameter from the previous INIT ACK chunk.

An implementation **SHOULD** make the cookie as small as possible to ensure interoperability.

Note: A Cookie Echo does not contain a State Cookie parameter; instead, the data within the State Cookie's Parameter Value becomes the data within the Cookie Echo's Chunk Value. This allows an implementation to change only the first 2 bytes of the State Cookie parameter to become a COOKIE ECHO chunk.

3.3.12. Cookie Acknowledgement (COOKIE ACK) (11)

This chunk is used only during the initialization of an association. It is used to acknowledge the receipt of a COOKIE ECHO chunk. This chunk MUST precede any DATA or SACK chunk sent within the association, but MAY be bundled with one or more DATA chunks or SACK chunk's in the same SCTP packet.

```

      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|  Type = 11    |  Chunk Flags  |                Length = 4          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Chunk Flags: 8 bits

Set to 0 on transmit and ignored on receipt.

3.3.13. Shutdown Complete (SHUTDOWN COMPLETE) (14)

This chunk MUST be used to acknowledge the receipt of the SHUTDOWN ACK chunk at the completion of the shutdown process; see Section 9.2 for details.

The SHUTDOWN COMPLETE chunk has no parameters.

```

      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|  Type = 14    |  Reserved    |T|                Length = 4          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Chunk Flags: 8 bits

Reserved: 7 bits

Set to 0 on transmit and ignored on receipt.

T bit: 1 bit

The T bit is set to 0 if the sender filled in the Verification Tag expected by the peer. If the Verification Tag is reflected, the T bit MUST be set to 1. Reflecting means that the sent Verification Tag is the same as the received one.

Note: Special rules apply to this chunk for verification, please see Section 8.5.1 for details.

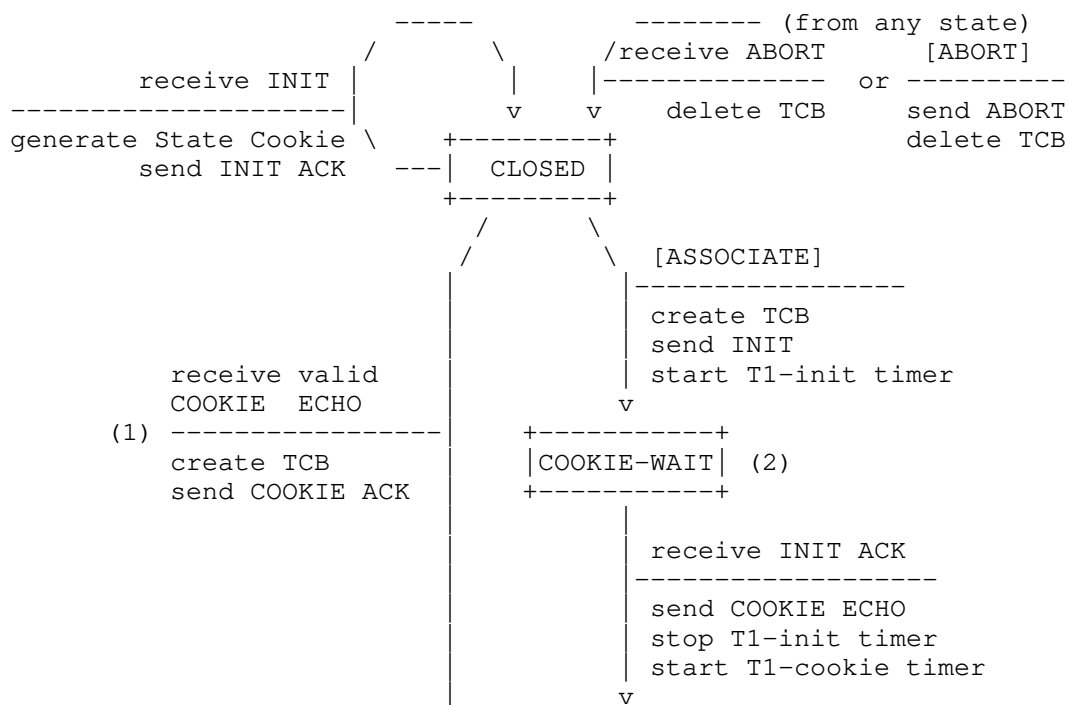
4. SCTP Association State Diagram

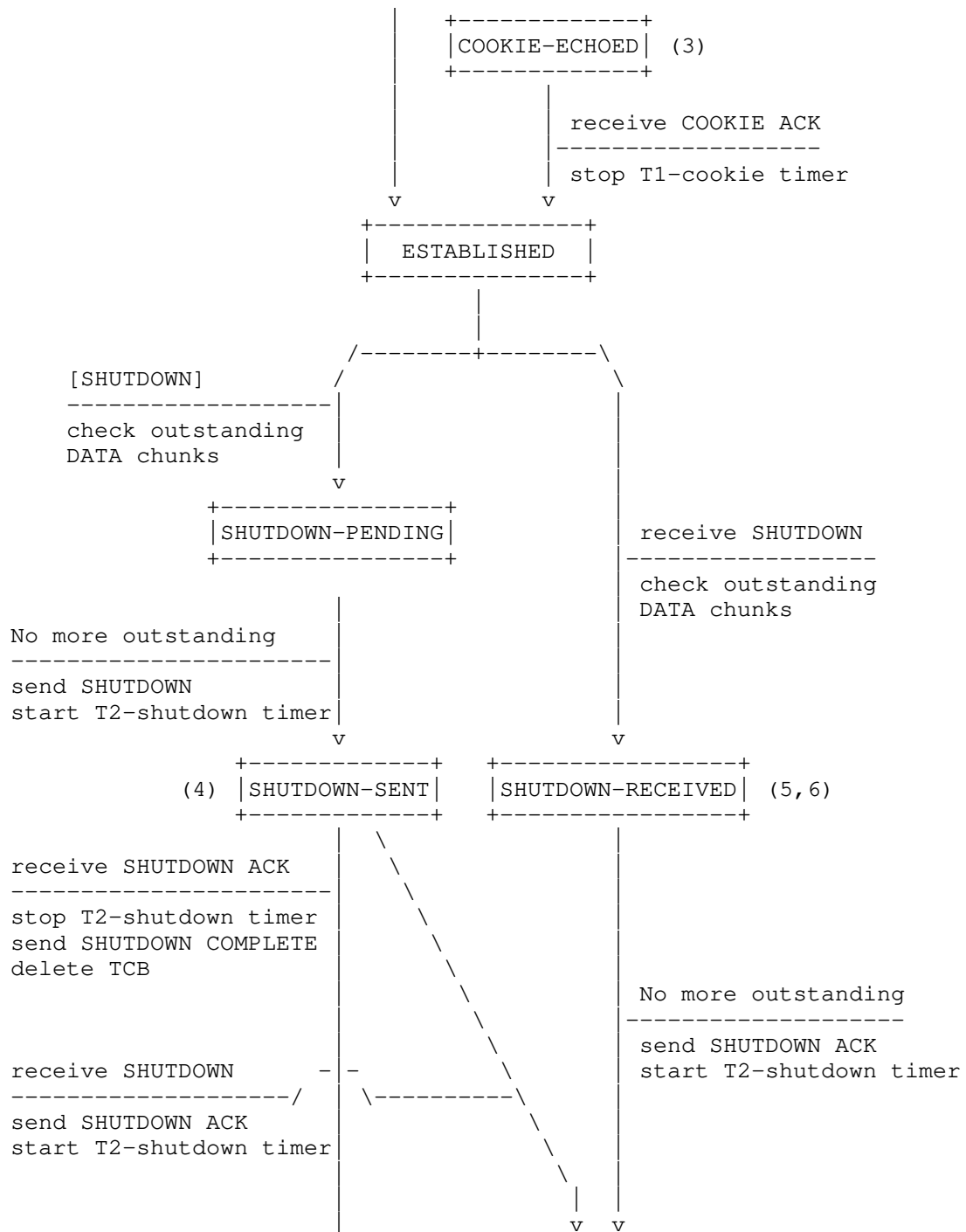
During the life time of an SCTP association, the SCTP endpoint's association progresses from one state to another in response to various events. The events that might potentially advance an association's state include:

- * SCTP user primitive calls, e.g., [ASSOCIATE], [SHUTDOWN], [ABORT],
- * Reception of INIT, COOKIE ECHO, ABORT, SHUTDOWN, etc., control chunks, or
- * Some timeout events.

The state diagram in the figures below illustrates state changes, together with the causing events and resulting actions. Note that some of the error conditions are not shown in the state diagram. Full descriptions of all special cases are found in the text.

Note: Chunk names are given in all capital letters, while parameter names have the first letter capitalized, e.g., COOKIE ECHO chunk type vs. State Cookie parameter. If more than one event/message can occur that causes a state transition, it is labeled (A), (B).





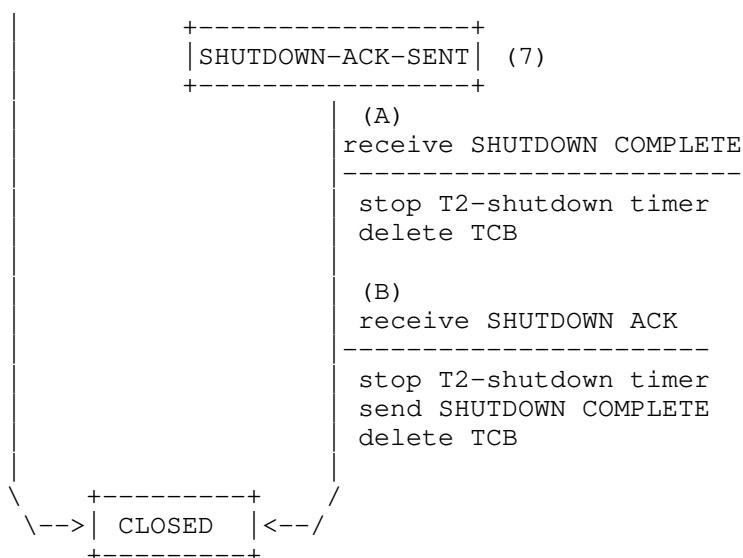


Figure 3: State Transition Diagram of SCTP

The following applies:

- 1) If the State Cookie in the received COOKIE ECHO chunk is invalid (i.e., failed to pass the integrity check), the receiver MUST silently discard the packet. Or, if the received State Cookie is expired (see Section 5.1.5), the receiver MUST send back an ERROR chunk. In either case, the receiver stays in the CLOSED state.
- 2) If the T1-init timer expires, the endpoint MUST retransmit the INIT chunk and restart the T1-init timer without changing state. This MUST be repeated up to 'Max.Init.Retransmits' times. After that, the endpoint MUST abort the initialization process and report the error to the SCTP user.
- 3) If the T1-cookie timer expires, the endpoint MUST retransmit COOKIE ECHO chunk and restart the T1-cookie timer without changing state. This MUST be repeated up to 'Max.Init.Retransmits' times. After that, the endpoint MUST abort the initialization process and report the error to the SCTP user.
- 4) In the SHUTDOWN-SENT state, the endpoint MUST acknowledge any received DATA chunks without delay.
- 5) In the SHUTDOWN-RECEIVED state, the endpoint MUST NOT accept any new send requests from its SCTP user.

- 6) In the SHUTDOWN-RECEIVED state, the endpoint MUST transmit or retransmit data and leave this state when all data in queue is transmitted.
- 7) In the SHUTDOWN-ACK-SENT state, the endpoint MUST NOT accept any new send requests from its SCTP user.

The CLOSED state is used to indicate that an association is not created (i.e., does not exist).

5. Association Initialization

Before the first data transmission can take place from one SCTP endpoint ("A") to another SCTP endpoint ("Z"), the two endpoints MUST complete an initialization process in order to set up an SCTP association between them.

The SCTP user at an endpoint can use the ASSOCIATE primitive to initialize an SCTP association to another SCTP endpoint.

Implementation Note: From an SCTP user's point of view, an association might be implicitly opened, without an ASSOCIATE primitive (see Section 11.1.2) being invoked, by the initiating endpoint's sending of the first user data to the destination endpoint. The initiating SCTP will assume default values for all mandatory and optional parameters for the INIT/INIT ACK chunk.

Once the association is established, unidirectional streams are open for data transfer on both ends (see Section 5.1.1).

5.1. Normal Establishment of an Association

The initialization process consists of the following steps (assuming that SCTP endpoint "A" tries to set up an association with SCTP endpoint "Z" and "Z" accepts the new association):

- A) "A" first builds a TCB and sends an INIT chunk to "Z". In the INIT chunk, "A" MUST provide its Verification Tag (Tag_A) in the Initiate Tag field. Tag_A SHOULD be a random number in the range of 1 to 4294967295 (see Section 5.3.1 for Tag value selection). After sending the INIT chunk, "A" starts the T1-init timer and enters the COOKIE-WAIT state.

- B) "Z" responds immediately with an INIT ACK chunk. The destination IP address of the INIT ACK chunk MUST be set to the source IP address of the INIT chunk to which this INIT ACK chunk is responding. In the response, besides filling in other parameters, "Z" MUST set the Verification Tag field to Tag_A, and also provide its own Verification Tag (Tag_Z) in the Initiate Tag field.

Moreover, "Z" MUST generate and send along with the INIT ACK chunk a State Cookie. See Section 5.1.3 for State Cookie generation.

After sending an INIT ACK chunk with the State Cookie parameter, "Z" MUST NOT allocate any resources or keep any states for the new association. Otherwise, "Z" will be vulnerable to resource attacks.

- C) Upon reception of the INIT ACK chunk from "Z", "A" stops the Tl-init timer and leaves the COOKIE-WAIT state. "A" then sends the State Cookie received in the INIT ACK chunk in a COOKIE ECHO chunk, starts the Tl-cookie timer, and enters the COOKIE-ECHOED state.

The COOKIE ECHO chunk MAY be bundled with any pending outbound DATA chunks, but it MUST be the first chunk in the packet and until the COOKIE ACK chunk is returned the sender MUST NOT send any other packets to the peer.

- D) Upon reception of the COOKIE ECHO chunk, endpoint "Z" replies with a COOKIE ACK chunk after building a TCB and moving to the ESTABLISHED state. A COOKIE ACK chunk MAY be bundled with any pending DATA chunks (and/or SACK chunks), but the COOKIE ACK chunk MUST be the first chunk in the packet.

Implementation Note: An implementation can choose to send the Communication Up notification to the SCTP user upon reception of a valid COOKIE ECHO chunk.

- E) Upon reception of the COOKIE ACK chunk, endpoint "A" moves from the COOKIE-ECHOED state to the ESTABLISHED state, stopping the Tl-cookie timer. It can also notify its ULP about the successful establishment of the association with a Communication Up notification (see Section 11).

An INIT or INIT ACK chunk MUST NOT be bundled with any other chunk. They MUST be the only chunks present in the SCTP packets that carry them.

An endpoint MUST send the INIT ACK chunk to the IP address from which it received the INIT chunk.

Tl-init timer and Tl-cookie timer SHOULD follow the same rules given in Section 6.3. If the application provided multiple IP addresses of the peer, there SHOULD be a Tl-init and Tl-cookie timer for each address of the peer. Retransmissions of INIT chunks and COOKIE ECHO chunks SHOULD use all addresses of the peer similar to retransmissions of DATA chunks.

If an endpoint receives an INIT, INIT ACK, or COOKIE ECHO chunk but decides not to establish the new association due to missing mandatory parameters in the received INIT or INIT ACK chunk, invalid parameter values, or lack of local resources, it SHOULD respond with an ABORT chunk. It SHOULD also specify the cause of abort, such as the type of the missing mandatory parameters, etc., by including the error cause parameters with the ABORT chunk. The Verification Tag field in the common header of the outbound SCTP packet containing the ABORT chunk MUST be set to the Initiate Tag value of the received INIT or INIT ACK chunk this ABORT chunk is responding to.

Note that a COOKIE ECHO chunk that does not pass the integrity check is not considered an 'invalid mandatory parameter' and requires special handling; see Section 5.1.5.

After the reception of the first DATA chunk in an association the endpoint MUST immediately respond with a SACK chunk to acknowledge the DATA chunk. Subsequent acknowledgements SHOULD be done as described in Section 6.2.

When the TCB is created, each endpoint MUST set its internal Cumulative TSN Ack Point to the value of its transmitted Initial TSN minus one.

Implementation Note: The IP addresses and SCTP port are generally used as the key to find the TCB within an SCTP instance.

5.1.1. Handle Stream Parameters

In the INIT and INIT ACK chunks, the sender of the chunk MUST indicate the number of outbound streams (OSs) it wishes to have in the association, as well as the maximum inbound streams (MISs) it will accept from the other endpoint.

After receiving the stream configuration information from the other side, each endpoint MUST perform the following check: If the peer's MIS is less than the endpoint's OS, meaning that the peer is incapable of supporting all the outbound streams the endpoint wants

to configure, the endpoint **MUST** use MIS outbound streams and **MAY** report any shortage to the upper layer. The upper layer can then choose to abort the association if the resource shortage is unacceptable.

After the association is initialized, the valid outbound stream identifier range for either endpoint **MUST** be 0 to min(local OS, remote MIS) - 1.

5.1.2. Handle Address Parameters

During the association initialization, an endpoint uses the following rules to discover and collect the destination transport address(es) of its peer.

- A) If there are no address parameters present in the received INIT or INIT ACK chunk, the endpoint **MUST** take the source IP address from which the chunk arrives and record it, in combination with the SCTP source port number, as the only destination transport address for this peer.
- B) If there is a Host Name Address parameter present in the received INIT or INIT ACK chunk, the endpoint **MUST** immediately send an ABORT chunk and **MAY** include an "Unresolvable Address" error cause to its peer. The ABORT chunk **SHOULD** be sent to the source IP address from which the last peer packet was received.
- C) If there are only IPv4/IPv6 addresses present in the received INIT or INIT ACK chunk, the receiver **MUST** derive and record all the transport addresses from the received chunk **AND** the source IP address that sent the INIT or INIT ACK chunk. The transport addresses are derived by the combination of SCTP source port (from the common header) and the IP Address parameter(s) carried in the INIT or INIT ACK chunk and the source IP address of the IP datagram. The receiver **SHOULD** use only these transport addresses as destination transport addresses when sending subsequent packets to its peer.
- D) An INIT or INIT ACK chunk **MUST** be treated as belonging to an already established association (or one in the process of being established) if the use of any of the valid address parameters contained within the chunk would identify an existing TCB.

Implementation Note: In some cases (e.g., when the implementation does not control the source IP address that is used for transmitting), an endpoint might need to include in its INIT or INIT ACK chunk all possible IP addresses from which packets to the peer could be transmitted.

After all transport addresses are derived from the INIT or INIT ACK chunk using the above rules, the endpoint selects one of the transport addresses as the initial primary path.

The packet containing the INIT ACK chunk MUST be sent to the source address of the packet containing the INIT chunk.

The sender of INIT chunks MAY include a 'Supported Address Types' parameter in the INIT chunk to indicate what types of addresses are acceptable.

Implementation Note: In the case that the receiver of an INIT ACK chunk fails to resolve the address parameter due to an unsupported type, it can abort the initiation process and then attempt a reinitiation by using a 'Supported Address Types' parameter in the new INIT chunk to indicate what types of address it prefers.

If an SCTP endpoint that only supports either IPv4 or IPv6 receives IPv4 and IPv6 addresses in an INIT or INIT ACK chunk from its peer, it MUST use all the addresses belonging to the supported address family. The other addresses MAY be ignored. The endpoint SHOULD NOT respond with any kind of error indication.

If an SCTP endpoint lists in the 'Supported Address Types' parameter either IPv4 or IPv6, but uses the other family for sending the packet containing the INIT chunk, or if it also lists addresses of the other family in the INIT chunk, then the address family that is not listed in the 'Supported Address Types' parameter SHOULD also be considered as supported by the receiver of the INIT chunk. The receiver of the INIT chunk SHOULD NOT respond with any kind of error indication.

5.1.3. Generating State Cookie

When sending an INIT ACK chunk as a response to an INIT chunk, the sender of INIT ACK chunk creates a State Cookie and sends it in the State Cookie parameter of the INIT ACK chunk. Inside this State Cookie, the sender MUST include a MAC (see [RFC2104] for an example) to provide integrity protection on the State Cookie. The State Cookie SHOULD also contain a timestamp on when the State Cookie is created, and the lifespan of the State Cookie, along with all the information necessary for it to establish the association including the port numbers and the verification tags.

The method used to generate the MAC is strictly a private matter for the receiver of the INIT chunk. The use of a MAC is mandatory to prevent denial-of-service attacks. MAC algorithms can have different performance depending on the platform. Choosing a high performance MAC algorithm increases the resistance against cookie flooding

attacks. A MAC with acceptable security properties SHOULD be used. The secret key SHOULD be random ([RFC4086] provides some information on randomness guidelines). The secret keys need to have an appropriate size. The secret key SHOULD be changed reasonably frequently (e.g., hourly), and the timestamp in the State Cookie MAY be used to determine which key is used to verify the MAC.

If the State Cookie is not encrypted, it MUST NOT contain information which is not being envisioned to be shared.

An implementation SHOULD make the cookie as small as possible to ensure interoperability.

5.1.4. State Cookie Processing

When an endpoint (in the COOKIE-WAIT state) receives an INIT ACK chunk with a State Cookie parameter, it MUST immediately send a COOKIE ECHO chunk to its peer with the received State Cookie. The sender MAY also add any pending DATA chunks to the packet after the COOKIE ECHO chunk.

The endpoint MUST also start the Tl-cookie timer after sending the COOKIE ECHO chunk. If the timer expires, the endpoint MUST retransmit the COOKIE ECHO chunk and restart the Tl-cookie timer. This is repeated until either a COOKIE ACK chunk is received or 'Max.Init.Retransmits' (see Section 16) is reached causing the peer endpoint to be marked unreachable (and thus the association enters the CLOSED state).

5.1.5. State Cookie Authentication

When an endpoint receives a COOKIE ECHO chunk from another endpoint with which it has no association, it takes the following actions:

- 1) Compute a MAC using the information carried in the State Cookie and the secret key. The timestamp in the State Cookie MAY be used to determine which secret key to use. If secrets are kept only for a limited amount of time and the secret key to use is not available anymore, the packet containing the COOKIE ECHO chunk MUST be silently discarded. [RFC2104] can be used as a guideline for generating the MAC,
- 2) Authenticate the State Cookie as one that it previously generated by comparing the computed MAC against the one carried in the State Cookie. If this comparison fails, the SCTP packet, including the COOKIE ECHO chunk and any DATA chunks, MUST be silently discarded,

- 3) Compare the port numbers and the Verification Tag contained within the COOKIE ECHO chunk to the actual port numbers and the Verification Tag within the SCTP common header of the received packet. If these values do not match, the packet MUST be silently discarded.
- 4) Compare the creation timestamp in the State Cookie to the current local time. If the elapsed time is longer than the lifespan carried in the State Cookie, then the packet, including the COOKIE ECHO chunk and any attached DATA chunks, SHOULD be discarded, and the endpoint MUST transmit an ERROR chunk with a "Stale Cookie" error cause to the peer endpoint.
- 5) If the State Cookie is valid, create an association to the sender of the COOKIE ECHO chunk with the information in the State Cookie carried in the COOKIE ECHO chunk and enter the ESTABLISHED state.
- 6) Send a COOKIE ACK chunk to the peer acknowledging receipt of the COOKIE ECHO chunk. The COOKIE ACK chunk MAY be bundled with an outbound DATA chunk or SACK chunk; however, the COOKIE ACK chunk MUST be the first chunk in the SCTP packet.
- 7) Immediately acknowledge any DATA chunk bundled with the COOKIE ECHO chunk with a SACK chunk (subsequent DATA chunk acknowledgement SHOULD follow the rules defined in Section 6.2). As mentioned in step 6, if the SACK chunk is bundled with the COOKIE ACK chunk, the COOKIE ACK chunk MUST appear first in the SCTP packet.

If a COOKIE ECHO chunk is received from an endpoint with which the receiver of the COOKIE ECHO chunk has an existing association, the procedures in Section 5.2 SHOULD be followed.

5.1.6. An Example of Normal Association Establishment

In the following example, "A" initiates the association and then sends a user message to "Z", then "Z" sends two user messages to "A" later (assuming no bundling or fragmentation occurs):

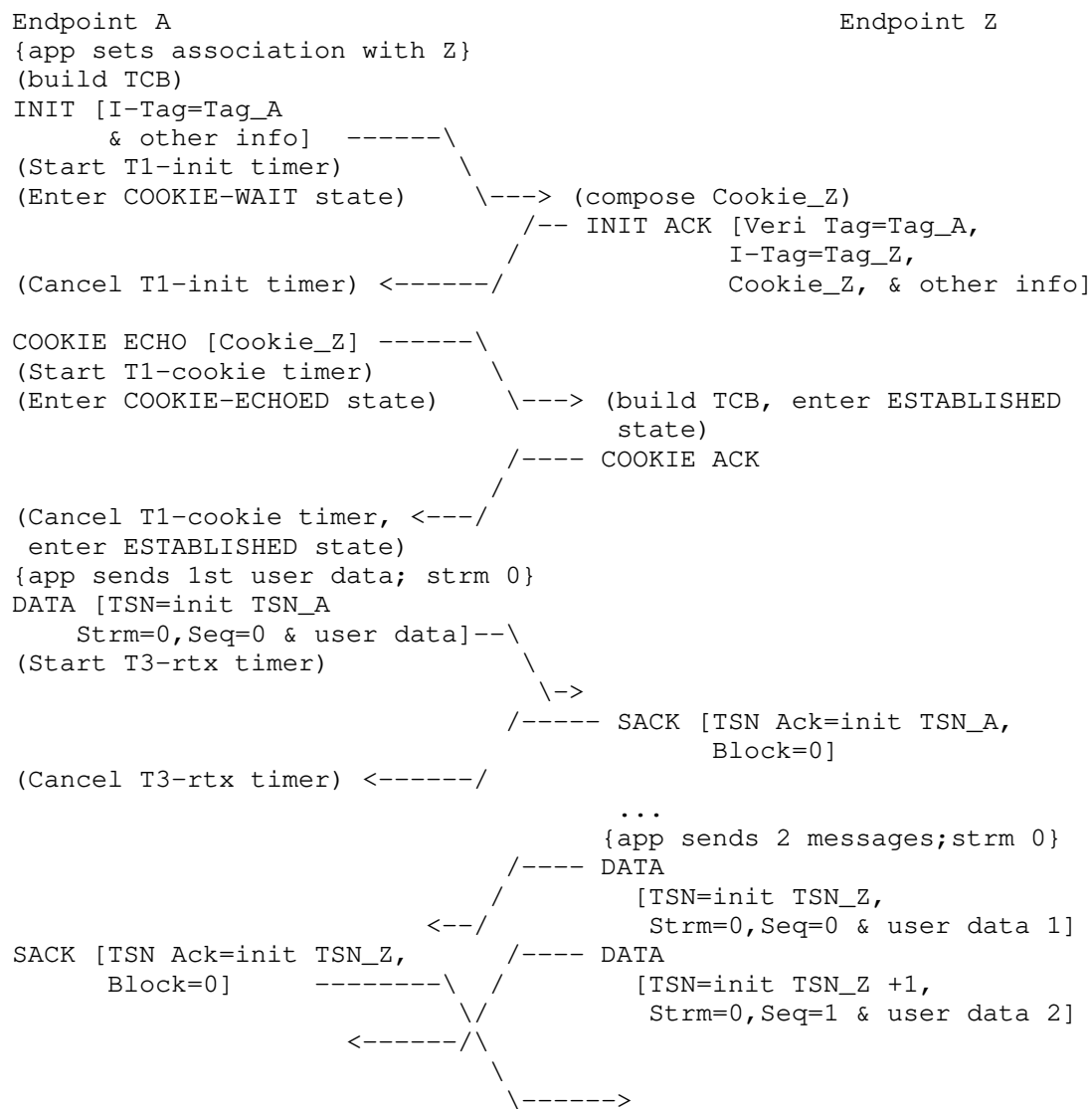


Figure 4: A Setup Example

If the Tl-init timer expires at "A" after the INIT or COOKIE ECHO chunks are sent, the same INIT or COOKIE ECHO chunk with the same Initiate Tag (i.e., Tag_A) or State Cookie is retransmitted and the timer restarted. This is repeated 'Max.Init.Retransmits' times before "A" considers "Z" unreachable and reports the failure to its upper layer (and thus the association enters the CLOSED state).

When retransmitting the INIT chunk, the endpoint MUST follow the rules defined in Section 6.3 to determine the proper timer value.

5.2. Handle Duplicate or Unexpected INIT, INIT ACK, COOKIE ECHO, and COOKIE ACK Chunks

During the life time of an association (in one of the possible states), an endpoint can receive from its peer endpoint one of the setup chunks (INIT, INIT ACK, COOKIE ECHO, and COOKIE ACK). The receiver treats such a setup chunk as a duplicate and process it as described in this section.

Note: An endpoint will not receive the chunk unless the chunk was sent to an SCTP transport address and is from an SCTP transport address associated with this endpoint. Therefore, the endpoint processes such a chunk as part of its current association.

The following scenarios can cause duplicated or unexpected chunks:

- A) The peer has crashed without being detected, restarted itself, and sent a new INIT chunk trying to restore the association,
- B) Both sides are trying to initialize the association at about the same time,
- C) The chunk is from a stale packet that was used to establish the present association or a past association that is no longer in existence,
- D) The chunk is a false packet generated by an attacker, or
- E) The peer never received the COOKIE ACK chunk and is retransmitting its COOKIE ECHO chunk.

The rules in the following sections are applied in order to identify and correctly handle these cases.

5.2.1. INIT Chunk Received in COOKIE-WAIT or COOKIE-ECHOED State (Item B)

This usually indicates an initialization collision, i.e., each endpoint is attempting, at about the same time, to establish an association with the other endpoint.

Upon receipt of an INIT chunk in the COOKIE-WAIT state, an endpoint MUST respond with an INIT ACK chunk 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 packet containing the INIT ACK chunk MUST only be sent to an address passed by the upper layer in the request to initialize the association.
- 2) The packet containing the INIT ACK chunk MUST only be sent to an address reported in the incoming INIT chunk.
- 3) The packet containing the INIT ACK chunk SHOULD be sent to the source address of the received packet containing the INIT chunk.

Upon receipt of an INIT chunk in the COOKIE-ECHOED state, an endpoint MUST respond with an INIT ACK chunk using the same parameters it sent in its original INIT chunk (including its Initiate Tag, unchanged), provided that no NEW address has been added to the forming association. If the INIT chunk indicates that a new address has been added to the association, then the entire INIT chunk MUST be discarded, and the state of the existing association SHOULD NOT be changed. An ABORT chunk SHOULD be sent in response that MAY include the error 'Restart of an association with new addresses'. The error SHOULD list the addresses that were added to the restarting association.

When responding in either state (COOKIE-WAIT or COOKIE-ECHOED) with an INIT ACK chunk, the original parameters are combined with those from the newly received INIT chunk. The endpoint MUST also generate a State Cookie with the INIT ACK chunk. The endpoint uses the parameters sent in its INIT chunk to calculate the State Cookie.

After that, the endpoint MUST NOT change its state, the T1-init timer MUST be left running, and the corresponding TCB MUST NOT be destroyed. The normal procedures for handling State Cookies when a TCB exists will resolve the duplicate INIT chunks to a single association.

For an endpoint that is in the COOKIE-ECHOED state, it MUST populate its Tie-Tags within both the association TCB and inside the State Cookie (see Section 5.2.2 for a description of the Tie-Tags).

5.2.2. Unexpected INIT Chunk in States Other than CLOSED, COOKIE-ECHOED, COOKIE-WAIT, and SHUTDOWN-ACK-SENT

Unless otherwise stated, upon receipt of an unexpected INIT chunk for this association, the endpoint MUST generate an INIT ACK chunk with a State Cookie. Before responding, the endpoint MUST check to see if the unexpected INIT chunk adds new addresses to the association. If new addresses are added to the association, the endpoint MUST respond with an ABORT chunk, copying the 'Initiate Tag' of the unexpected INIT chunk into the 'Verification Tag' of the outbound packet

carrying the ABORT chunk. In the ABORT chunk, the error cause MAY be set to 'restart of an association with new addresses'. The error SHOULD list the addresses that were added to the restarting association. If no new addresses are added, when responding to the INIT chunk in the outbound INIT ACK chunk, the endpoint MUST copy its current Tie-Tags to a reserved place within the State Cookie and the association's TCB. We refer to these locations inside the cookie as the Peer's-Tie-Tag and the Local-Tie-Tag. We will refer to the copy within an association's TCB as the Local Tag and Peer's Tag. The outbound SCTP packet containing this INIT ACK chunk MUST carry a Verification Tag value equal to the Initiate Tag found in the unexpected INIT chunk. And the INIT ACK chunk MUST contain a new Initiate Tag (randomly generated; see Section 5.3.1). Other parameters for the endpoint SHOULD be copied from the existing parameters of the association (e.g., number of outbound streams) into the INIT ACK chunk and cookie.

After sending the INIT ACK or ABORT chunk, the endpoint MUST take no further actions; i.e., the existing association, including its current state, and the corresponding TCB MUST NOT be changed.

Only when a TCB exists and the association is not in a COOKIE-WAIT or SHUTDOWN-ACK-SENT state are the Tie-Tags populated with a random value other than 0. For a normal association INIT chunk (i.e., the endpoint is in the CLOSED state), the Tie-Tags MUST be set to 0 (indicating that no previous TCB existed).

5.2.3. Unexpected INIT ACK Chunk

If an INIT ACK chunk is received by an endpoint in any state other than the COOKIE-WAIT or CLOSED state, the endpoint SHOULD discard the INIT ACK chunk. An unexpected INIT ACK chunk usually indicates the processing of an old or duplicated INIT chunk.

5.2.4. Handle a COOKIE ECHO Chunk when a TCB Exists

When a COOKIE ECHO chunk is received by an endpoint in any state for an existing association (i.e., not in the CLOSED state) the following rules are applied:

- 1) Compute a MAC as described in step 1 of Section 5.1.5,
- 2) Authenticate the State Cookie as described in step 2 of Section 5.1.5 (this is case C or D above).
- 3) Compare the timestamp in the State Cookie to the current time. If the State Cookie is older than the lifespan carried in the State Cookie and the Verification Tags contained in the State

Cookie do not match the current association's Verification Tags, the packet, including the COOKIE ECHO chunk and any DATA chunks, SHOULD be discarded. The endpoint also MUST transmit an ERROR chunk with a "Stale Cookie" error cause to the peer endpoint (this is case C or D in Section 5.2).

If both Verification Tags in the State Cookie match the Verification Tags of the current association, consider the State Cookie valid (this is case E in Section 5.2) even if the lifespan is exceeded.

- 4) If the State Cookie proves to be valid, unpack the TCB into a temporary TCB.
- 5) Refer to Table 12 to determine the correct action to be taken.

Local Tag	Peer's Tag	Local-Tie-Tag	Peer's-Tie-Tag	Action
X	X	M	M	(A)
M	X	A	A	(B)
M	0	A	A	(B)
X	M	0	0	(C)
M	M	A	A	(D)

Table 12: Handling of a COOKIE ECHO Chunk when a TCB Exists

Legend:

- X - Tag does not match the existing TCB.
- M - Tag matches the existing TCB.
- 0 - Tag unknown (Peer's Tag not known yet / No tie-tag in cookie).
- A - All cases, i.e., M, X, or 0.

For any case not shown in Table 12, the cookie SHOULD be silently discarded.

Action

- A) In this case, the peer might have restarted. When the endpoint recognizes this potential 'restart', the existing session is treated the same as if it received an ABORT chunk followed by a new COOKIE ECHO chunk with the following exceptions:

- * Any SCTP DATA chunks MAY be retained (this is an implementation-specific option).
- * A notification of RESTART SHOULD be sent to the ULP instead of a "COMMUNICATION LOST" notification.

All the congestion control parameters (e.g., cwnd, ssthresh) related to this peer MUST be reset to their initial values (see Section 6.2.1).

After this, the endpoint enters the ESTABLISHED state.

If the endpoint is in the SHUTDOWN-ACK-SENT state and recognizes that the peer has restarted (Action A), it MUST NOT set up a new association but instead resend the SHUTDOWN ACK chunk and send an ERROR chunk with a "Cookie Received While Shutting Down" error cause to its peer.

- B) In this case, both sides might be attempting to start an association at about the same time, but the peer endpoint sent its INIT chunk after responding to the local endpoint's INIT chunk. Thus, it might have picked a new Verification Tag, not being aware of the previous tag it had sent this endpoint. The endpoint SHOULD stay in or enter the ESTABLISHED state, but it MUST update its peer's Verification Tag from the State Cookie, stop any T1-init or T1-cookie timers that might be running, and send a COOKIE ACK chunk.
- C) In this case, the local endpoint's cookie has arrived late. Before it arrived, the local endpoint sent an INIT chunk and received an INIT ACK chunk and finally sent a COOKIE ECHO chunk with the peer's same tag but a new tag of its own. The cookie SHOULD be silently discarded. The endpoint SHOULD NOT change states and SHOULD leave any timers running.
- D) When both local and remote tags match, the endpoint SHOULD enter the ESTABLISHED state, if it is in the COOKIE-ECHOED state. It SHOULD stop any T1-cookie timer that is running and send a COOKIE ACK chunk.

Note: The "peer's Verification Tag" is the tag received in the Initiate Tag field of the INIT or INIT ACK chunk.

5.2.4.1. An Example of a Association Restart

In the following example, "A" initiates the association after a restart has occurred. Endpoint "Z" had no knowledge of the restart until the exchange (i.e., Heartbeats had not yet detected the failure of "A") (assuming no bundling or fragmentation occurs):

```

Endpoint A                                     Endpoint Z
<----- Association is established----->
Tag=Tag_A                                     Tag=Tag_Z
<----->
{A crashes and restarts}
{app sets up a association with Z}
{build TCB}
INIT [I-Tag=Tag_A'
      & other info] -----\
(Start T1-init timer)       \
(Enter COOKIE-WAIT state)   \----> (find an existing TCB,
                                   populate TieTags if needed,
                                   compose Cookie_Z with Tie-Tags
                                   and other info)
                                   /--- INIT ACK [Veri Tag=Tag_A',
                                   I-Tag=Tag_Z',
                                   Cookie_Z]
                                   (leave original TCB in place)
(Cancel T1-init timer) <-----/
COOKIE ECHO [Veri=Tag_Z',
             Cookie_Z]-----\
(Start T1-init timer)       \
(Enter COOKIE-ECHOED state)  \----> (Find existing association,
                                   Tie-Tags in Cookie_Z match
                                   Tie-Tags in TCB,
                                   Tags do not match, i.e.,
                                   case X X M M above,
                                   Announce Restart to ULP
                                   and reset association).
                                   /----- COOKIE ACK
(Cancel T1-init timer, <-----/
  Enter ESTABLISHED state)
{app sends 1st user data; strm 0}
DATA [TSN=initial TSN_A
      Strm=0,Seq=0 & user data]--\
(Start T3-rtx timer)           \
                                   \-->
                                   /--- SACK [TSN Ack=init TSN_A,Block=0]
(Cancel T3-rtx timer) <-----/

```

Figure 5: A Restart Example

5.2.5. Handle Duplicate COOKIE ACK Chunk

At any state other than COOKIE-ECHOED, an endpoint SHOULD silently discard a received COOKIE ACK chunk.

5.2.6. Handle Stale Cookie Error

Receipt of an ERROR chunk with a "Stale Cookie" error cause indicates one of a number of possible events:

- A) The association failed to completely setup before the State Cookie issued by the sender was processed.
- B) An old State Cookie was processed after setup completed.
- C) An old State Cookie is received from someone that the receiver is not interested in having an association with and the ABORT chunk was lost.

When processing an ERROR chunk with a "Stale Cookie" error cause an endpoint SHOULD first examine if an association is in the process of being set up, i.e., the association is in the COOKIE-ECHOED state. In all cases, if the association is not in the COOKIE-ECHOED state, the ERROR chunk SHOULD be silently discarded.

If the association is in the COOKIE-ECHOED state, the endpoint MAY elect one of the following three alternatives.

- 1) Send a new INIT chunk to the endpoint to generate a new State Cookie and reattempt the setup procedure.
- 2) Discard the TCB and report to the upper layer the inability to set up the association.
- 3) Send a new INIT chunk to the endpoint, adding a Cookie Preservative parameter requesting an extension to the life time of the State Cookie. When calculating the time extension, an implementation SHOULD use the RTT information measured based on the previous COOKIE ECHO / ERROR chunk exchange, and SHOULD add no more than 1 second beyond the measured RTT, due to long State Cookie life times making the endpoint more subject to a replay attack.

5.3. Other Initialization Issues

5.3.1. Selection of Tag Value

Initiate Tag values SHOULD be selected from the range of 1 to $2^{32} - 1$. It is very important that the Initiate Tag value be randomized to help protect against "man in the middle" and "sequence number" attacks. The methods described in [RFC4086] can be used for the Initiate Tag randomization. Careful selection of Initiate Tags is also necessary to prevent old duplicate packets from previous associations being mistakenly processed as belonging to the current association.

Moreover, the Verification Tag value used by either endpoint in a given association MUST NOT change during the life time of an association. A new Verification Tag value MUST be used each time the endpoint tears down and then reestablishes an association to the same peer.

5.4. Path Verification

During association establishment, the two peers exchange a list of addresses. In the predominant case, these lists accurately represent the addresses owned by each peer. However, a misbehaving peer might supply addresses that it does not own. To prevent this, the following rules are applied to all addresses of the new association:

- 1) Any addresses passed to the sender of the INIT chunk by its upper layer in the request to initialize an association are automatically considered to be CONFIRMED.
- 2) For the receiver of the COOKIE ECHO chunk, the only CONFIRMED address is the address to which the packet containing the INIT ACK chunk was sent.
- 3) All other addresses not covered by rules 1 and 2 are considered UNCONFIRMED and are subject to probing for verification.

To probe an address for verification, an endpoint will send HEARTBEAT chunks including a 64-bit random nonce and a path indicator (to identify the address that the HEARTBEAT chunk is sent to) within the Heartbeat Info parameter.

Upon receipt of the HEARTBEAT ACK chunk, a verification is made that the nonce included in the Heartbeat Info parameter is the one sent to the address indicated inside the Heartbeat Info parameter. When this match occurs, the address that the original HEARTBEAT was sent to is now considered CONFIRMED and available for normal data transfer.

These probing procedures are started when an association moves to the ESTABLISHED state and are ended when all paths are confirmed.

In each RTO, a probe MAY be sent on an active UNCONFIRMED path in an attempt to move it to the CONFIRMED state. If during this probing the path becomes inactive, this rate is lowered to the normal HEARTBEAT rate. At the expiration of the RTO timer, the error counter of any path that was probed but not CONFIRMED is incremented by one and subjected to path failure detection, as defined in Section 8.2. When probing UNCONFIRMED addresses, however, the association overall error count is not incremented.

The number of packets containing HEARTBEAT chunks sent at each RTO SHOULD be limited by the 'HB.Max.Burst' parameter. It is an implementation decision as to how to distribute packets containing HEARTBEAT chunks to the peer's addresses for path verification.

Whenever a path is confirmed, an indication MAY be given to the upper layer.

An endpoint MUST NOT send any chunks to an UNCONFIRMED address, with the following exceptions:

- * A HEARTBEAT chunk including a nonce MAY be sent to an UNCONFIRMED address.
- * A HEARTBEAT ACK chunk MAY be sent to an UNCONFIRMED address.
- * A COOKIE ACK chunk MAY be sent to an UNCONFIRMED address, but it MUST be bundled with a HEARTBEAT chunk including a nonce. An implementation that does not support bundling MUST NOT send a COOKIE ACK chunk to an UNCONFIRMED address.
- * A COOKIE ECHO chunk MAY be sent to an UNCONFIRMED address, but it MUST be bundled with a HEARTBEAT chunk including a nonce, and the size of the SCTP packet MUST NOT exceed the PMTU. If the implementation does not support bundling or if the bundled COOKIE ECHO chunk plus HEARTBEAT chunk (including nonce) would result in an SCTP packet larger than the PMTU, then the implementation MUST NOT send a COOKIE ECHO chunk to an UNCONFIRMED address.

6. User Data Transfer

Data transmission MUST only happen in the ESTABLISHED, SHUTDOWN-PENDING, and SHUTDOWN-RECEIVED states. The only exception to this is that DATA chunks are allowed to be bundled with an outbound COOKIE ECHO chunk when in the COOKIE-WAIT state.

DATA chunks MUST only be received according to the rules below in ESTABLISHED, SHUTDOWN-PENDING, and SHUTDOWN-SENT states. A DATA chunk received in CLOSED is out of the blue and SHOULD be handled per Section 8.4. A DATA chunk received in any other state SHOULD be discarded.

A SACK chunk MUST be processed in ESTABLISHED, SHUTDOWN-PENDING, and SHUTDOWN-RECEIVED states. An incoming SACK chunk MAY be processed in COOKIE-ECHOED. A SACK chunk in the CLOSED state is out of the blue and SHOULD be processed according to the rules in Section 8.4. A SACK chunk received in any other state SHOULD be discarded.

For transmission efficiency, SCTP defines mechanisms for bundling of small user messages and fragmentation of large user messages. The following diagram depicts the flow of user messages through SCTP.

In this section, the term "data sender" refers to the endpoint that transmits a DATA chunk and the term "data receiver" refers to the endpoint that receives a DATA chunk. A data receiver will transmit SACK chunks.

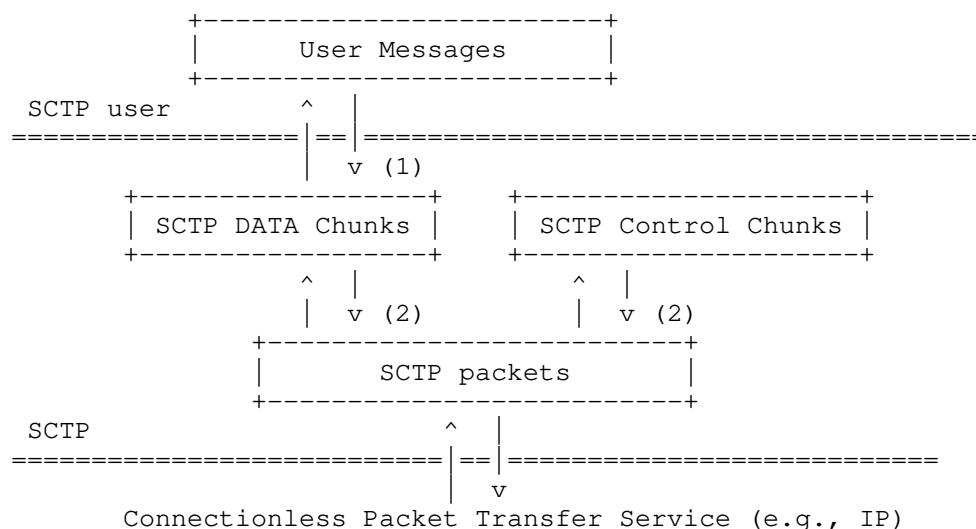


Figure 6: Illustration of User Data Transfer

The following applies:

- 1) When converting user messages into DATA chunks, an endpoint **MUST** fragment large user messages into multiple DATA chunks. The size of each DATA chunk **SHOULD** be smaller than or equal to the Association Maximum DATA Chunk Size (AMDCS). The data receiver will normally reassemble the fragmented message from DATA chunks before delivery to the user (see Section 6.9 for details).
- 2) Multiple DATA and control chunks **MAY** be bundled by the sender into a single SCTP packet for transmission, as long as the final size of the SCTP packet does not exceed the current PMTU. The receiver will unbundle the packet back into the original chunks. Control chunks **MUST** come before DATA chunks in the packet.

The fragmentation and bundling mechanisms, as detailed in Section 6.9 and Section 6.10, are **OPTIONAL** to implement by the data sender, but they **MUST** be implemented by the data receiver, i.e., an endpoint **MUST** properly receive and process bundled or fragmented data.

6.1. Transmission of DATA Chunks

This section specifies the rules for sending DATA chunks. In particular, it defines zero window probing, which is required to avoid the indefinite stalling of an association in case of a loss of packets containing SACK chunks performing window updates.

This document is specified as if there is a single retransmission timer per destination transport address, but implementations **MAY** have a retransmission timer for each DATA chunk.

The following general rules **MUST** be applied by the data sender for transmission and/or retransmission of outbound DATA chunks:

- A) At any given time, the data sender **MUST NOT** transmit new data to any destination transport address if its peer's `rwnd` indicates that the peer has no buffer space (i.e., `rwnd` is smaller than the size of the next DATA chunk; see Section 6.2.1), except for zero window probes.

A zero window probe is a DATA chunk sent when the receiver has no buffer space. This rule allows the sender to probe for a change in `rwnd` that the sender missed due to the SACK chunks having been lost in transit from the data receiver to the data sender. A zero window probe **MUST** only be sent when the `cwnd` allows (see Rule B below). A zero window probe **SHOULD** only be sent when all outstanding DATA chunks have been cumulatively acknowledged and no DATA chunks are in flight. Senders **MUST** support zero window probing.

If the sender continues to receive SACK chunks from the peer while doing zero window probing, the unacknowledged window probes SHOULD NOT increment the error counter for the association or any destination transport address. This is because the receiver could keep its window closed for an indefinite time. Section 6.2 describes the receiver behavior when it advertises a zero window. The sender SHOULD send the first zero window probe after 1 RTO when it detects that the receiver has closed its window and SHOULD increase the probe interval exponentially afterwards. Also note that the cwnd SHOULD be adjusted according to Section 7.2.1. Zero window probing does not affect the calculation of cwnd.

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

- B) At any given time, the sender MUST NOT transmit new data to a given transport address if it has $\text{cwnd} + (\text{PMDCS} - 1)$ or more bytes of data outstanding to that transport address. If data is available, the sender SHOULD exceed cwnd by up to $(\text{PMDCS} - 1)$ bytes on a new data transmission if the flightsize does not currently reach cwnd. The breach of cwnd MUST constitute one packet only.
- C) When the time comes for the sender to transmit, before sending new DATA chunks, the sender MUST first transmit any DATA chunks that are marked for retransmission (limited by the current cwnd).
- D) When the time comes for the sender to transmit new DATA chunks, the protocol parameter 'Max.Burst' SHOULD be used to limit the number of packets sent. The limit MAY be applied by adjusting cwnd temporarily, as follows:

```
if ((flightsize + Max.Burst * PMDCS) < cwnd)
    cwnd = flightsize + Max.Burst * PMDCS;
```

Or, it MAY be applied by strictly limiting the number of packets emitted by the output routine. When calculating the number of packets to transmit, and particularly when using the formula above, cwnd SHOULD NOT be changed permanently.

- E) Then, the sender can send as many new DATA chunks as rule A and rule B allow.

Multiple DATA chunks committed for transmission MAY be bundled in a single packet. Furthermore, DATA chunks being retransmitted MAY be bundled with new DATA chunks, as long as the resulting SCTP packet size does not exceed the PMTU. A ULP can request that no bundling is performed, but this only turns off any delays that an SCTP implementation might be using to increase bundling efficiency. It does not in itself stop all bundling from occurring (i.e., in case of congestion or retransmission).

Before an endpoint transmits a DATA chunk, if any received DATA chunks have not been acknowledged (e.g., due to delayed ack), the sender SHOULD create a SACK chunk and bundle it with the outbound DATA chunk, as long as the size of the final SCTP packet does not exceed the current PMTU. See Section 6.2.

When the window is full (i.e., transmission is disallowed by rule A and/or rule B), the sender MAY still accept send requests from its upper layer, but MUST transmit no more DATA chunks until some or all of the outstanding DATA chunks are acknowledged and transmission is allowed by rule A and rule B again.

Whenever a transmission or retransmission is made to any address, if the T3-rtx timer of that address is not currently running, the sender MUST start that timer. If the timer for that address is already running, the sender MUST restart the timer if the earliest (i.e., lowest TSN) outstanding DATA chunk sent to that address is being retransmitted. Otherwise, the data sender MUST NOT restart the timer.

When starting or restarting the T3-rtx timer, the timer value SHOULD be adjusted according to the timer rules defined in Section 6.3.2 and Section 6.3.3.

The data sender MUST NOT use a TSN that is more than $2^{31} - 1$ above the beginning TSN of the current send window.

For each stream, the data sender MUST NOT have more than $2^{16} - 1$ ordered user messages in the current send window.

Whenever the sender of a DATA chunk can benefit from the corresponding SACK chunk being sent back without delay, the sender MAY set the I bit in the DATA chunk header. Please note that why the sender has set the I bit is irrelevant to the receiver.

Reasons for setting the I bit include, but are not limited to, the following (see Section 4 of [RFC7053] for a discussion of the benefits):

- * The application requests that the I bit of the last DATA chunk of a user message be set when providing the user message to the SCTP implementation (see Section 11.1).
- * The sender is in the SHUTDOWN-PENDING state.
- * The sending of a DATA chunk fills the congestion or receiver window.

6.2. Acknowledgement on Reception of DATA Chunks

The SCTP endpoint MUST always acknowledge the reception of each valid DATA chunk when the DATA chunk received is inside its receive window.

When the receiver's advertised window is 0, the receiver MUST drop any new incoming DATA chunk with a TSN larger than the largest TSN received so far. Also, if the new incoming DATA chunk holds a TSN value less than the largest TSN received so far, then the receiver SHOULD drop the largest TSN held for reordering and accept the new incoming DATA chunk. In either case, if such a DATA chunk is dropped, the receiver MUST immediately send back a SACK chunk with the current receive window showing only DATA chunks received and accepted so far. The dropped DATA chunk(s) MUST NOT be included in the SACK chunk, as they were not accepted. The receiver MUST also have an algorithm for advertising its receive window to avoid receiver silly window syndrome (SWS), as described in [RFC1122]. The algorithm can be similar to the one described in Section 4.2.3.3 of [RFC1122].

The guidelines on delayed acknowledgement algorithm specified in Section 4.2 of [RFC5681] SHOULD be followed. Specifically, an acknowledgement SHOULD be generated for at least every second packet (not every second DATA chunk) received, and SHOULD be generated within 200 ms of the arrival of any unacknowledged DATA chunk. In some situations, it might be beneficial for an SCTP transmitter to be more conservative than the algorithms detailed in this document allow. However, an SCTP transmitter MUST NOT be more aggressive in sending SACK chunks than the following algorithms allow.

An SCTP receiver MUST NOT generate more than one SACK chunk for every incoming packet, other than to update the offered window as the receiving application consumes new data. When the window opens up, an SCTP receiver SHOULD send additional SACK chunks to update the window even if no new data is received. The receiver MUST avoid sending a large number of window updates -- in particular, large bursts of them. One way to achieve this is to send a window update only if the window can be increased by at least a quarter of the receive buffer size of the association.

Implementation Note: The maximum delay for generating an acknowledgement MAY be configured by the SCTP administrator, either statically or dynamically, in order to meet the specific timing requirement of the protocol being carried.

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

Acknowledgements MUST be sent in SACK chunks unless shutdown was requested by the ULP, in which case an endpoint MAY send an acknowledgement in the SHUTDOWN chunk. A SACK chunk can acknowledge the reception of multiple DATA chunks. See Section 3.3.4 for SACK chunk format. In particular, the SCTP endpoint MUST fill in the Cumulative TSN Ack field to indicate the latest sequential TSN (of a valid DATA chunk) it has received. Any received DATA chunks with TSN greater than the value in the Cumulative TSN Ack field are reported in the Gap Ack Block fields. The SCTP endpoint MUST report as many Gap Ack Blocks as can fit in a single SACK chunk such that the size of the SCTP packet does not exceed the current PMTU.

The SHUTDOWN chunk does not contain Gap Ack Block fields. Therefore, the endpoint SHOULD use a SACK chunk instead of the SHUTDOWN chunk to acknowledge DATA chunks received out of order.

Upon receipt of an SCTP packet containing a DATA chunk with the I bit set, the receiver SHOULD NOT delay the sending of the corresponding SACK chunk, i.e., the receiver SHOULD immediately respond with the corresponding SACK chunk.

When a packet arrives with duplicate DATA chunk(s) and with no new DATA chunk(s), the endpoint MUST immediately send a SACK chunk with no delay. If a packet arrives with duplicate DATA chunk(s) bundled with new DATA chunks, the endpoint MAY immediately send a SACK chunk. Normally, receipt of duplicate DATA chunks will occur when the original SACK chunk was lost and the peer's RTO has expired. The duplicate TSN number(s) SHOULD be reported in the SACK chunk as duplicate.

When an endpoint receives a SACK chunk, it MAY use the duplicate TSN information to determine if SACK chunk loss is occurring. Further use of this data is for future study.

The data receiver is responsible for maintaining its receive buffers. The data receiver SHOULD notify the data sender in a timely manner of changes in its ability to receive data. How an implementation manages its receive buffers is dependent on many factors (e.g.,

operating system, memory management system, amount of memory, etc.). However, the data sender strategy defined in Section 6.2.1 is based on the assumption of receiver operation similar to the following:

- A) At initialization of the association, the endpoint tells the peer how much receive buffer space it has allocated to the association in the INIT or INIT ACK chunk. The endpoint sets `a_rwnd` to this value.
- B) As DATA chunks are received and buffered, decrement `a_rwnd` by the number of bytes received and buffered. This is, in effect, closing `rwnd` at the data sender and restricting the amount of data it can transmit.
- C) As DATA chunks are delivered to the ULP and released from the receive buffers, increment `a_rwnd` by the number of bytes delivered to the upper layer. This is, in effect, opening up `rwnd` on the data sender and allowing it to send more data. The data receiver SHOULD NOT increment `a_rwnd` unless it has released bytes from its receive buffer. For example, if the receiver is holding fragmented DATA chunks in a reassembly queue, it SHOULD NOT increment `a_rwnd`.
- D) When sending a SACK chunk, the data receiver SHOULD place the current value of `a_rwnd` into the `a_rwnd` field. The data receiver SHOULD take into account that the data sender will not retransmit DATA chunks that are acked via the Cumulative TSN Ack (i.e., will drop from its retransmit queue).

Under certain circumstances, the data receiver MAY drop DATA chunks that it has received but has not released from its receive buffers (i.e., delivered to the ULP). These DATA chunks might have been acked in Gap Ack Blocks. For example, the data receiver might be holding data in its receive buffers while reassembling a fragmented user message from its peer when it runs out of receive buffer space. It MAY drop these DATA chunks even though it has acknowledged them in Gap Ack Blocks. If a data receiver drops DATA chunks, it MUST NOT include them in Gap Ack Blocks in subsequent SACK chunks until they are received again via retransmission. In addition, the endpoint SHOULD take into account the dropped data when calculating its `a_rwnd`.

An endpoint SHOULD NOT revoke a SACK chunk and discard data. Only in extreme circumstances might an endpoint use this procedure (such as out of buffer space). The data receiver SHOULD take into account that dropping data that has been acked in Gap Ack Blocks can result in suboptimal retransmission strategies in the data sender and thus in suboptimal performance.

The following example illustrates the use of delayed acknowledgements:

Endpoint A	Endpoint Z
{App sends 3 messages; strm 0}	
DATA [TSN=7,Strm=0,Seq=3] ----->	(ack delayed)
(Start T3-rtx timer)	
DATA [TSN=8,Strm=0,Seq=4] ----->	(send ack)
	/----- SACK [TSN Ack=8,block=0]
(cancel T3-rtx timer) <-----/	
DATA [TSN=9,Strm=0,Seq=5] ----->	(ack delayed)
(Start T3-rtx timer)	
	...
	{App sends 1 message; strm 1}
	(bundle SACK with DATA)
	/----- SACK [TSN Ack=9,block=0] \
	DATA [TSN=6,Strm=1,Seq=2]
(cancel T3-rtx timer) <-----/	(Start T3-rtx timer)
(ack delayed)	
(send ack)	
SACK [TSN Ack=6,block=0] ----->	(cancel T3-rtx timer)

Figure 7: Delayed Acknowledgement Example

If an endpoint receives a DATA chunk with no user data (i.e., the Length field is set to 16), it SHOULD send an ABORT chunk with a "No User Data" error cause.

An endpoint SHOULD NOT send a DATA chunk with no user data part. This avoids the need to be able to return a zero-length user message in the API, especially in the socket API as specified in [RFC6458] for details.

6.2.1. Processing a Received SACK Chunk

Each SACK chunk an endpoint receives contains an `a_rwnd` value. This value represents the amount of buffer space the data receiver, at the time of transmitting the SACK chunk, has left of its total receive buffer space (as specified in the INIT/INIT ACK chunk). Using `a_rwnd`, Cumulative TSN Ack, and Gap Ack Blocks, the data sender can develop a representation of the peer's receive buffer space.

One of the problems the data sender takes into account when processing a SACK chunk is that a SACK chunk can be received out of order. That is, a SACK chunk sent by the data receiver can pass an earlier SACK chunk and be received first by the data sender. If a SACK chunk is received out of order, the data sender can develop an incorrect view of the peer's receive buffer space.

Since there is no explicit identifier that can be used to detect out-of-order SACK chunks, the data sender uses heuristics to determine if a SACK chunk is new.

An endpoint SHOULD use the following rules to calculate the `rwnd`, using the `a_rwnd` value, the Cumulative TSN Ack, and Gap Ack Blocks in a received SACK chunk.

- A) At the establishment of the association, the endpoint initializes the `rwnd` to the Advertised Receiver Window Credit (`a_rwnd`) the peer specified in the INIT or INIT ACK chunk.
- B) Any time a DATA chunk is transmitted (or retransmitted) to a peer, the endpoint subtracts the data size of the chunk from the `rwnd` of that peer.
- C) Any time a DATA chunk is marked for retransmission, either via T3-rtx timer expiration (Section 6.3.3) or via Fast Retransmit (Section 7.2.4), add the data size of those chunks to the `rwnd`.
- D) Any time a SACK chunk arrives, the endpoint performs the following:
 - i) If Cumulative TSN Ack is less than the Cumulative TSN Ack Point, then drop the SACK chunk. Since Cumulative TSN Ack is monotonically increasing, a SACK chunk whose Cumulative TSN Ack is less than the Cumulative TSN Ack Point indicates an out-of-order SACK chunk.
 - ii) Set `rwnd` equal to the newly received `a_rwnd` minus the number of bytes still outstanding after processing the Cumulative TSN Ack and the Gap Ack Blocks.
 - iii) If the SACK chunk is missing a TSN that was previously acknowledged via a Gap Ack Block (e.g., the data receiver reneged on the data), then consider the corresponding DATA that might be possibly missing: Count one miss indication towards Fast Retransmit as described in Section 7.2.4, and if no retransmit timer is running for the destination address to which the DATA chunk was originally transmitted, then T3-rtx is started for that destination address.

- iv) If the Cumulative TSN Ack matches or exceeds the Fast Recovery exitpoint (Section 7.2.4), Fast Recovery is exited.

6.3. Management of Retransmission Timer

An SCTP endpoint uses a retransmission timer T3-rtx to ensure data delivery in the absence of any feedback from its peer. The duration of this timer is referred to as RTO (retransmission timeout).

When an endpoint's peer is multi-homed, the endpoint will calculate a separate RTO for each different destination transport address of its peer endpoint.

The computation and management of RTO in SCTP follow closely how TCP manages its retransmission timer. To compute the current RTO, an endpoint maintains two state variables per destination transport address: SRTT (smoothed round-trip time) and RTTVAR (round-trip time variation).

6.3.1. RTO Calculation

The rules governing the computation of SRTT, RTTVAR, and RTO are as follows:

- C1) Until an RTT measurement has been made for a packet sent to the given destination transport address, set RTO to the protocol parameter 'RTO.Initial'.

- C2) When the first RTT measurement R is made, perform

```
SRTT = R;  
RTTVAR = R/2;  
RTO = SRTT + 4 * RTTVAR;
```

- C3) When a new RTT measurement R' is made, perform:

```
RTTVAR = (1 - RTO.Beta) * RTTVAR + RTO.Beta * |SRTT - R'|;  
SRTT = (1 - RTO.Alpha) * SRTT + RTO.Alpha * R';
```

Note: The value of SRTT used in the update to RTTVAR is its value before updating SRTT itself using the second assignment.

After the computation, update

```
RTO = SRTT + 4 * RTTVAR;
```

C4) When data is in flight and when allowed by rule C5 below, a new RTT measurement MUST be made each round trip. Furthermore, new RTT measurements SHOULD be made no more than once per round trip for a given destination transport address. There are two reasons for this recommendation: First, it appears that measuring more frequently often does not in practice yield any significant benefit [ALLMAN99]; second, if measurements are made more often, then the values of 'RTO.Alpha' and 'RTO.Beta' in rule C3 above SHOULD be adjusted so that SRTT and RTTVAR still adjust to changes at roughly the same rate (in terms of how many round trips it takes them to reflect new values) as they would if making only one measurement per round-trip and using 'RTO.Alpha' and 'RTO.Beta' as given in rule C3. However, the exact nature of these adjustments remains a research issue.

C5) Karn's algorithm: RTT measurements MUST NOT be made using chunks that were retransmitted (and thus for which it is ambiguous whether the reply was for the first instance of the chunk or for a later instance).

RTT measurements SHOULD only be made using a DATA chunk with TSN r , if no DATA chunk with TSN less than or equal to r was retransmitted since the DATA chunk with TSN r was sent first.

C6) Whenever RTO is computed, if it is less than 'RTO.Min' seconds then it is rounded up to 'RTO.Min' seconds. The reason for this rule is that RTOs that do not have a high minimum value are susceptible to unnecessary timeouts [ALLMAN99].

C7) A maximum value MAY be placed on RTO provided it is at least 'RTO.max' seconds.

There is no requirement for the clock granularity G used for computing RTT measurements and the different state variables, other than:

G1) Whenever RTTVAR is computed, if $RTTVAR == 0$, then adjust $RTTVAR = G$.

Experience [ALLMAN99] has shown that finer clock granularities (less than 100 msec) perform somewhat better than more coarse granularities.

See Section 16 for suggested parameter values.

6.3.2. Retransmission Timer Rules

The rules for managing the retransmission timer are as follows:

- R1) Every time a DATA chunk is sent to any address (including a retransmission), if the T3-rtx timer of that address is not running, start it running so that it will expire after the RTO of that address. The RTO used here is that obtained after any doubling due to previous T3-rtx timer expirations on the corresponding destination address as discussed in rule E2 below.
- R2) Whenever all outstanding data sent to an address have been acknowledged, turn off the T3-rtx timer of that address.
- R3) Whenever a SACK chunk is received that acknowledges the DATA chunk with the earliest outstanding TSN for that address, restart the T3-rtx timer for that address with its current RTO (if there is still outstanding data on that address).
- R4) Whenever a SACK chunk is received missing a TSN that was previously acknowledged via a Gap Ack Block, start the T3-rtx for the destination address to which the DATA chunk was originally transmitted if it is not already running.

The following example shows the use of various timer rules (assuming that the receiver uses delayed acks).

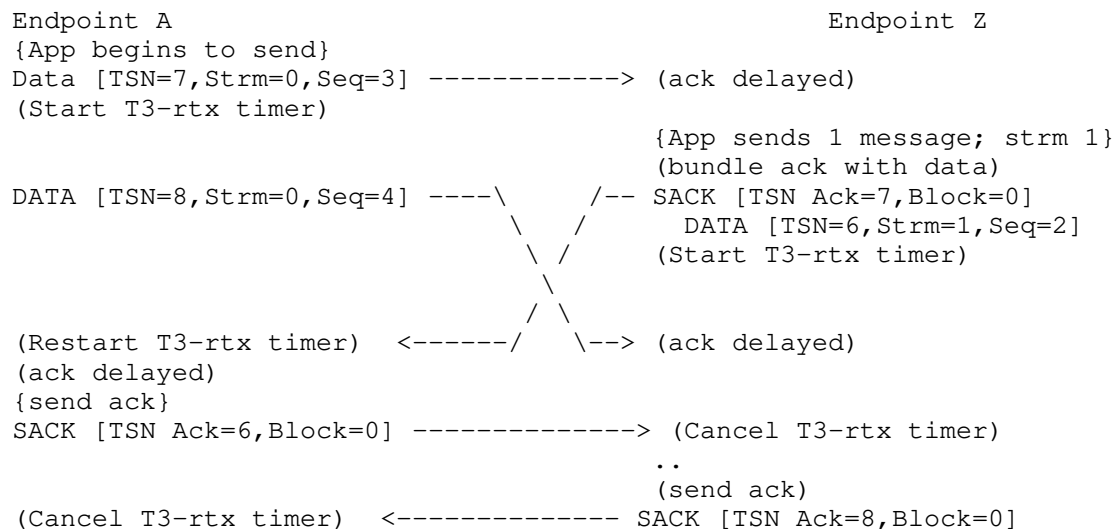


Figure 8: Timer Rule Examples

6.3.3. Handle T3-rtx Expiration

Whenever the retransmission timer T3-rtx expires for a destination address, do the following:

- E1) For the destination address for which the timer expires, adjust its ssthresh with rules defined in Section 7.2.3 and set the cwnd = PMDCS.
- E2) For the destination address for which the timer expires, set $RTO = RTO * 2$ ("back off the timer"). The maximum value discussed in rule C7 above ('RTO.max') MAY be used to provide an upper bound to this doubling operation.
- E3) Determine how many of the earliest (i.e., lowest TSN) outstanding DATA chunks for the address for which the T3-rtx has expired will fit into a single SCTP packet, subject to the PMTU corresponding to the destination transport address to which the retransmission is being sent (this might be different from the address for which the timer expires; see Section 6.4). Call this value K. Bundle and retransmit those K DATA chunks in a single packet to the destination endpoint.
- E4) Start the retransmission timer T3-rtx on the destination address to which the retransmission is sent, if rule R1 above indicates to do so. The RTO to be used for starting T3-rtx SHOULD be the one for the destination address to which the retransmission is sent, which, when the receiver is multi-homed, might be different from the destination address for which the timer expired (see Section 6.4 below).

After retransmitting, once a new RTT measurement is obtained (which can happen only when new data has been sent and acknowledged, per rule C5, or for a measurement made from a HEARTBEAT chunk; see Section 8.3), the computation in rule C3 is performed, including the computation of RTO, which might result in "collapsing" RTO back down after it has been subject to doubling (rule E2).

Any DATA chunks that were sent to the address for which the T3-rtx timer expired but did not fit in an SCTP packet of size smaller than or equal to the PMTU (rule E3 above) SHOULD be marked for retransmission and sent as soon as cwnd allows (normally, when a SACK chunk arrives).

The final rule for managing the retransmission timer concerns failover (see Section 6.4.1):

F1) Whenever an endpoint switches from the current destination transport address to a different one, the current retransmission timers are left running. As soon as the endpoint transmits a packet containing DATA chunk(s) to the new transport address, start the timer on that transport address, using the RTO value of the destination address to which the data is being sent, if rule R1 indicates to do so.

6.4. Multi-Homed SCTP Endpoints

An SCTP endpoint is considered multi-homed if there is more than one transport address that can be used as a destination address to reach that endpoint.

Moreover, the ULP of an endpoint selects one of the multiple destination addresses of a multi-homed peer endpoint as the primary path (see Section 5.1.2 and Section 11.1 for details).

By default, an endpoint SHOULD always transmit to the primary path, unless the SCTP user explicitly specifies the destination transport address (and possibly source transport address) to use.

An endpoint SHOULD transmit reply chunks (e.g., INIT ACK, COOKIE ACK, HEARTBEAT ACK) in response to control chunks to the same destination transport address from which it received the control chunk to which it is replying.

The selection of the destination transport address for packets containing SACK chunks is implementation dependent. However, an endpoint SHOULD NOT vary the destination transport address of a SACK chunk 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 chunk, the SACK chunk MAY be transmitted to one of the destination transport addresses from which the DATA or control chunks being acknowledged were received.

When a receiver of a duplicate DATA chunk sends a SACK chunk to a multi-homed endpoint, it MAY be beneficial to vary the destination address and not use the source address of the DATA chunk. The reason is that receiving a duplicate from a multi-homed endpoint might indicate that the return path (as specified in the source address of the DATA chunk) for the SACK chunk is broken.

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

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

Retransmissions do not affect the total outstanding data count. However, if the DATA chunk is retransmitted onto a different destination address, both the outstanding data counts on the new destination address and the old destination address to which the data chunk was last sent is adjusted accordingly.

6.4.1. Failover from an Inactive Destination Address

Some of the transport addresses of a multi-homed SCTP endpoint might become inactive due to either the occurrence of certain error conditions (see Section 8.2) or adjustments from the SCTP user.

When there is outbound data to send and the primary path becomes inactive (e.g., due to failures), or where the SCTP user explicitly requests to send data to an inactive destination transport address, before reporting an error to its ULP, the SCTP endpoint SHOULD try to send the data to an alternate active destination transport address if one exists.

When retransmitting data that timed out, if the endpoint is multi-homed, it needs to consider each source-destination address pair in its retransmission selection policy. When retransmitting timed-out data, the endpoint SHOULD attempt to pick the most divergent source-destination pair from the original source-destination pair to which the packet was transmitted.

Note: Rules for picking the most divergent source-destination pair are an implementation decision and are not specified within this document.

6.5. Stream Identifier and Stream Sequence Number

Every DATA chunk MUST carry a valid stream identifier. If an endpoint receives a DATA chunk with an invalid stream identifier, it SHOULD acknowledge the reception of the DATA chunk following the normal procedure, immediately send an ERROR chunk with cause set to "Invalid Stream Identifier" (see Section 3.3.10), and discard the DATA chunk. The endpoint MAY bundle the ERROR chunk and the SACK chunk in the same packet.

The Stream Sequence Number in all the outgoing streams MUST start from 0 when the association is established. The Stream Sequence Number of an outgoing stream MUST be incremented by 1 for each ordered user message sent on that outgoing stream. In particular, when the Stream Sequence Number reaches the value 65535 the next Stream Sequence Number MUST be set to 0. For unordered user messages the Stream Sequence Number MUST NOT be changed.

6.6. Ordered and Unordered Delivery

Within a stream, an endpoint MUST deliver DATA chunks received with the U flag set to 0 to the upper layer according to the order of their Stream Sequence Number. If DATA chunks arrive out of order of their Stream Sequence Number, the endpoint MUST hold the received DATA chunks from delivery to the ULP until they are reordered.

However, an SCTP endpoint can indicate that no ordered delivery is required for a particular DATA chunk transmitted within the stream by setting the U flag of the DATA chunk to 1.

When an endpoint receives a DATA chunk with the U flag set to 1, it bypasses the ordering mechanism and immediately deliver the data to the upper layer (after reassembly if the user data is fragmented by the data sender).

This provides an effective way of transmitting "out-of-band" data in a given stream. Also, a stream can be used as an "unordered" stream by simply setting the U flag to 1 in all DATA chunks sent through that stream.

Implementation Note: When sending an unordered DATA chunk, an implementation MAY choose to place the DATA chunk in an outbound packet that is at the head of the outbound transmission queue if possible.

The 'Stream Sequence Number' field in a DATA chunk with U flag set to 1 has no significance. The sender can fill the 'Stream Sequence Number' with arbitrary value, but the receiver MUST ignore the field.

Note: When transmitting ordered and unordered data, an endpoint does not increment its Stream Sequence Number when transmitting a DATA chunk with U flag set to 1.

6.7. Report Gaps in Received DATA TSNs

Upon the reception of a new DATA chunk, an endpoint examines the continuity of the TSNs received. If the endpoint detects a gap in the received DATA chunk sequence, it SHOULD send a SACK chunk with Gap Ack Blocks immediately. The data receiver continues sending a SACK chunk after receipt of each SCTP packet that does not fill the gap.

Based on the Gap Ack Block from the received SACK chunk, the endpoint can calculate the missing DATA chunks and make decisions on whether to retransmit them (see Section 6.2.1 for details).

Multiple gaps can be reported in one single SACK chunk (see Section 3.3.4).

When its peer is multi-homed, the SCTP endpoint SHOULD always try to send the SACK chunk to the same destination address from which the last DATA chunk was received.

Upon the reception of a SACK chunk, the endpoint MUST remove all DATA chunks that have been acknowledged by the SACK chunk's Cumulative TSN Ack from its transmit queue. All DATA chunks with TSNs not included in the Gap Ack Blocks that are smaller than the highest acknowledged TSN reported in the SACK chunk MUST be treated as "missing" by the sending endpoint. The number of "missing" reports for each outstanding DATA chunk MUST be recorded by the data sender to make retransmission decisions. See Section 7.2.4 for details.

The following example shows the use of SACK chunk to report a gap.

```

Endpoint A                                Endpoint Z
{App sends 3 messages; strm 0}
DATA [TSN=6,Strm=0,Seq=2] -----> (ack delayed)
(Start T3-rtx timer)

DATA [TSN=7,Strm=0,Seq=3] -----> X (lost)

DATA [TSN=8,Strm=0,Seq=4] -----> (gap detected,
                                   immediately send ack)
                                   /----- SACK [TSN Ack=6,Block=1,
                                   /          Start=2,End=2]
                                   <-----/
(remove 6 from out-queue,
 and mark 7 as "1" missing report)

```

Figure 9: Reporting a Gap using SACK Chunk

The maximum number of Gap Ack Blocks that can be reported within a single SACK chunk is limited by the current PMTU. When a single SACK chunk cannot cover all the Gap Ack Blocks needed to be reported due to the PMTU limitation, the endpoint **MUST** send only one SACK chunk. This single SACK chunk **MUST** report the Gap Ack Blocks from the lowest to highest TSNs, within the size limit set by the PMTU, and leave the remaining highest TSN numbers unacknowledged.

6.8. CRC32c Checksum Calculation

When sending an SCTP packet, the endpoint **MUST** strengthen the data integrity of the transmission by including the CRC32c checksum value calculated on the packet, as described below.

After the packet is constructed (containing the SCTP common header and one or more control or DATA chunks), the transmitter **MUST**

- 1) fill in the proper Verification Tag in the SCTP common header and initialize the checksum field to 0,
- 2) calculate the CRC32c checksum of the whole packet, including the SCTP common header and all the chunks (refer to Appendix A for details of the CRC32c algorithm); and
- 3) put the resultant value into the checksum field in the common header, and leave the rest of the bits unchanged.

When an SCTP packet is received, the receiver **MUST** first check the CRC32c checksum as follows:

- 1) Store the received CRC32c checksum value aside.

- 2) Replace the 32 bits of the checksum field in the received SCTP packet with 0 and calculate a CRC32c checksum value of the whole received packet.
- 3) Verify that the calculated CRC32c checksum is the same as the received CRC32c checksum. If it is not, the receiver MUST treat the packet as an invalid SCTP packet.

The default procedure for handling invalid SCTP packets is to silently discard them.

Any hardware implementation SHOULD permit alternative verification of the CRC in software.

6.9. Fragmentation and Reassembly

An endpoint MAY support fragmentation when sending DATA chunks, but it MUST support reassembly when receiving DATA chunks. If an endpoint supports fragmentation, it MUST fragment a user message if the size of the user message to be sent causes the outbound SCTP packet size to exceed the current PMTU. An endpoint that does not support fragmentation and is requested to send a user message such that the outbound SCTP packet size would exceed the current PMTU MUST return an error to its upper layer and MUST NOT attempt to send the user message.

An SCTP implementation MAY provide a mechanism to the upper layer that disables fragmentation when sending DATA chunks. When fragmentation of DATA chunks is disabled, the SCTP implementation MUST behave in the same way an implementation that does not support fragmentation, i.e., it rejects calls that would result in sending SCTP packets that exceed the current PMTU.

Implementation Note: In this error case, the SEND primitive discussed in Section 11.1 would need to return an error to the upper layer.

If its peer is multi-homed, the endpoint SHOULD choose a DATA chunk size smaller than or equal to the AMDCS.

Once a user message is fragmented, it cannot be re-fragmented. Instead, if the PMTU has been reduced, then IP fragmentation MUST be used. Therefore, an SCTP association can fail if IP fragmentation is not working on any path. Please see Section 7.3 for details of PMTU discovery.

When determining when to fragment, the SCTP implementation MUST take into account the SCTP packet header as well as the DATA chunk header(s). The implementation MUST also take into account the space required for a SACK chunk if bundling a SACK chunk with the DATA chunk.

Fragmentation takes the following steps:

- 1) The data sender MUST break the user message into a series of DATA chunks. The sender SHOULD choose a size of DATA chunks that is smaller than or equal to the AMDCS.
- 2) The transmitter MUST then assign, in sequence, a separate TSN to each of the DATA chunks in the series. The transmitter assigns the same Stream Sequence Number to each of the DATA chunks. If the user indicates that the user message is to be delivered using unordered delivery, then the U flag of each DATA chunk of the user message MUST be set to 1.
- 3) The transmitter MUST also set the B/E bits of the first DATA chunk in the series to '10', the B/E bits of the last DATA chunk in the series to '01', and the B/E bits of all other DATA chunks in the series to '00'.

An endpoint MUST recognize fragmented DATA chunks by examining the B/E bits in each of the received DATA chunks, and queue the fragmented DATA chunks for reassembly. Once the user message is reassembled, SCTP passes the reassembled user message to the specific stream for possible reordering and final dispatching.

If the data receiver runs out of buffer space while still waiting for more fragments to complete the reassembly of the message, it SHOULD dispatch part of its inbound message through a partial delivery API (see Section 11), freeing some of its receive buffer space so that the rest of the message can be received.

6.10. Bundling

An endpoint bundles chunks by simply including multiple chunks in one outbound SCTP packet. The total size of the resultant SCTP packet MUST be less than or equal to the current PMTU.

If its peer endpoint is multi-homed, the sending endpoint SHOULD choose a size no larger than the PMTU of the current primary path.

When bundling control chunks with DATA chunks, an endpoint **MUST** place control chunks first in the outbound SCTP packet. The transmitter **MUST** transmit DATA chunks within an SCTP packet in increasing order of TSN.

Note: Since control chunks are placed first in a packet and since DATA chunks are transmitted before SHUTDOWN or SHUTDOWN ACK chunks, DATA chunks cannot be bundled with SHUTDOWN or SHUTDOWN ACK chunks.

Partial chunks **MUST NOT** be placed in an SCTP packet. A partial chunk is a chunk that is not completely contained in the SCTP packet; i.e., the SCTP packet is too short to contain all the bytes of the chunk as indicated by the chunk length.

An endpoint **MUST** process received chunks in their order in the packet. The receiver uses the Chunk Length field to determine the end of a chunk and beginning of the next chunk taking account of the fact that all chunks end on a 4-byte boundary. If the receiver detects a partial chunk, it **MUST** drop the chunk.

An endpoint **MUST NOT** bundle INIT, INIT ACK, or SHUTDOWN COMPLETE chunks with any other chunks.

7. Congestion Control

Congestion control is one of the basic functions in SCTP. To manage congestion, the mechanisms and algorithms in this section are to be employed.

Implementation Note: As far as its specific performance requirements are met, an implementation is always allowed to adopt a more conservative congestion control algorithm than the one defined below.

The congestion control algorithms used by SCTP are based on [RFC5681]. This section describes how the algorithms defined in [RFC5681] are adapted for use in SCTP. We first list differences in protocol designs between TCP and SCTP, and then describe SCTP's congestion control scheme. The description will use the same terminology as in TCP congestion control whenever appropriate.

SCTP congestion control is always applied to the entire association, and not to individual streams.

7.1. SCTP Differences from TCP Congestion Control

Gap Ack Blocks in the SCTP SACK chunk carry the same semantic meaning as the TCP SACK. TCP considers the information carried in the SACK as advisory information only. SCTP considers the information carried in the Gap Ack Blocks in the SACK chunk as advisory. In SCTP, any DATA chunk that has been acknowledged by a SACK chunk, including DATA that arrived at the receiving end out of order, is not considered fully delivered until the Cumulative TSN Ack Point passes the TSN of the DATA chunk (i.e., the DATA chunk has been acknowledged by the Cumulative TSN Ack field in the SACK chunk). Consequently, the value of cwnd controls the amount of outstanding data, rather than (as in the case of non-SACK TCP) the upper bound between the highest acknowledged sequence number and the latest DATA chunk that can be sent within the congestion window. SCTP SACK leads to different implementations of Fast Retransmit and Fast Recovery than non-SACK TCP. As an example, see [FALL96].

The biggest difference between SCTP and TCP, however, is multi-homing. SCTP is designed to establish robust communication associations between two endpoints each of which might be reachable by more than one transport address. Potentially different addresses might lead to different data paths between the two endpoints; thus, ideally one needs a separate set of congestion control parameters for each of the paths. The treatment here of congestion control for multi-homed receivers is new with SCTP and might require refinement in the future. The current algorithms make the following assumptions:

- * The sender usually uses the same destination address until being instructed by the upper layer to do otherwise; however, SCTP MAY change to an alternate destination in the event an address is marked inactive (see Section 8.2). Also, SCTP MAY retransmit to a different transport address than the original transmission.
- * The sender keeps a separate congestion control parameter set for each of the destination addresses it can send to (not each source-destination pair but for each destination). The parameters SHOULD decay if the address is not used for a long enough time period. [RFC5681] specifies this period of time as a retransmission timeout.
- * For each of the destination addresses, an endpoint does slow start upon the first transmission to that address.

Note: TCP guarantees in-sequence delivery of data to its upper-layer protocol within a single TCP session. This means that when TCP notices a gap in the received sequence number, it waits until the gap

is filled before delivering the data that was received with sequence numbers higher than that of the missing data. On the other hand, SCTP can deliver data to its upper-layer protocol even if there is a gap in TSN if the Stream Sequence Numbers are in sequence for a particular stream (i.e., the missing DATA chunks are for a different stream) or if unordered delivery is indicated. Although this does not affect cwnd, it might affect rwnd calculation.

7.2. SCTP Slow-Start and Congestion Avoidance

The slow-start and congestion avoidance algorithms **MUST** be used by an endpoint to control the amount of data being injected into the network. The congestion control in SCTP is employed in regard to the association, not to an individual stream. In some situations, it might be beneficial for an SCTP sender to be more conservative than the algorithms allow; however, an SCTP sender **MUST NOT** be more aggressive than the following algorithms allow.

Like TCP, an SCTP endpoint uses the following three control variables to regulate its transmission rate.

- * Receiver advertised window size (rwnd, in bytes), which is set by the receiver based on its available buffer space for incoming packets.

Note: This variable is kept on the entire association.

- * Congestion control window (cwnd, in bytes), which is adjusted by the sender based on observed network conditions.

Note: This variable is maintained on a per-destination-address basis.

- * Slow-start threshold (ssthresh, in bytes), which is used by the sender to distinguish slow-start and congestion avoidance phases.

Note: This variable is maintained on a per-destination-address basis.

SCTP also requires one additional control variable, `partial_bytes_acked`, which is used during congestion avoidance phase to facilitate cwnd adjustment.

Unlike TCP, an SCTP sender **MUST** keep a set of these control variables cwnd, ssthresh, and `partial_bytes_acked` for **EACH** destination address of its peer (when its peer is multi-homed). When calculating one of these variables, the length of the DATA chunk including the padding **SHOULD** be used.

Only one `rwnd` is kept for the whole association (no matter if the peer is multi-homed or has a single address).

7.2.1. Slow-Start

Beginning data transmission into a network with unknown conditions or after a sufficiently long idle period requires SCTP to probe the network to determine the available capacity. The slow-start algorithm is used for this purpose at the beginning of a transfer, or after repairing loss detected by the retransmission timer.

- * The initial `cwnd` before data transmission MUST be set to $\min(4 * PMDCS, \max(2 * PMDCS, 4404))$ bytes if the peer address is an IPv4 address and to $\min(4 * PMDCS, \max(2 * PMDCS, 4344))$ bytes if the peer address is an IPv6 address.
- * The initial `cwnd` after a retransmission timeout MUST be no more than `PMDCS`, and only one packet is allowed to be in flight until successful acknowledgement.
- * The initial value of `ssthresh` SHOULD be arbitrarily high (e.g., the size of the largest possible advertised window).
- * Whenever `cwnd` is greater than zero, the endpoint is allowed to have `cwnd` bytes of data outstanding on that transport address. A limited overbooking as described in Section 6.1 B) SHOULD be supported.
- * When `cwnd` is less than or equal to `ssthresh`, an SCTP endpoint MUST use the slow-start algorithm to increase `cwnd` only if the current congestion window is being fully utilized, and the data sender is not in Fast Recovery. Only when these two conditions are met can the `cwnd` be increased; otherwise, the `cwnd` MUST NOT be increased. If these conditions are met, then `cwnd` MUST be increased by, at most, the lesser of
 1. the total size of the previously outstanding DATA chunk(s) acknowledged, and
 2. L times the destination's `PMDCS`.

The first upper bound protects against the ACK-Splitting attack outlined in [SAVAGE99]. The positive integer L SHOULD be 1, and MAY be larger than 1. See [RFC3465] for details of choosing L .

In instances where its peer endpoint is multi-homed, if an endpoint receives a SACK chunk that results in updating the cwnd, then it SHOULD update its cwnd (or cwnds) apportioned to the destination addresses to which it transmitted the acknowledged data.

- * While the endpoint does not transmit data on a given transport address, the cwnd of the transport address SHOULD be adjusted to $\max(\text{cwnd} / 2, 4 * \text{PMDCS})$ once per RTT. Before the first cwnd adjustment, the ssthresh of the transport address SHOULD be set to the cwnd.

7.2.2. Congestion Avoidance

When cwnd is greater than ssthresh, cwnd SHOULD be incremented by PMDCS per RTT if the sender has cwnd or more bytes of data outstanding for the corresponding transport address. The basic recommendations for incrementing cwnd during congestion avoidance are as follows:

- * SCTP MAY increment cwnd by PMDCS.
- * SCTP SHOULD increment cwnd by PMDCS 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 PMDCS per RTT.

In practice, an implementation can achieve this goal in the following way:

- * `partial_bytes_acked` is initialized to 0.
- * Whenever cwnd is greater than ssthresh, upon each SACK chunk arrival, increase `partial_bytes_acked` by the total number of bytes (including the chunk header and the padding) of all new DATA chunks acknowledged in that SACK chunk, 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 (1) `partial_bytes_acked` is greater than cwnd and (2) before the arrival of the SACK chunk the sender had less than cwnd bytes of data outstanding (i.e., before the arrival of the SACK chunk, `flightsize` was less than cwnd), reset `partial_bytes_acked` to cwnd.

- * When (1) `partial_bytes_acked` is equal to or greater than `cwnd` and (2) before the arrival of the SACK chunk the sender had `cwnd` or more bytes of data outstanding (i.e., before the arrival of the SACK chunk, `flightsize` was greater than or equal to `cwnd`), `partial_bytes_acked` is reset to `(partial_bytes_acked - cwnd)`. Next, `cwnd` is increased by `PMDCS`.
- * Same as in the slow start, when the sender does not transmit DATA chunks on a given transport address, the `cwnd` of the transport address SHOULD be adjusted to `max(cwnd / 2, 4 * PMDCS)` per RTO.
- * When all of the data transmitted by the sender has been acknowledged by the receiver, `partial_bytes_acked` is initialized to 0.

7.2.3. Congestion Control

Upon detection of packet losses from SACK chunks (see Section 7.2.4), an endpoint SHOULD do the following:

```
ssthresh = max(cwnd / 2, 4 * PMDCS)
cwnd = ssthresh
partial_bytes_acked = 0
```

Basically, a packet loss causes `cwnd` to be cut in half.

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

```
ssthresh = max(cwnd / 2, 4 * PMDCS)
cwnd = PMDCS
partial_bytes_acked = 0
```

and ensure that no more than one SCTP packet will be in flight for that address until the endpoint receives acknowledgement for successful delivery of data to that address.

7.2.4. Fast Retransmit on Gap Reports

In the absence of data loss, an endpoint performs delayed acknowledgement. However, whenever an endpoint notices a hole in the arriving TSN sequence, it SHOULD start sending a SACK chunk back every time a packet arrives carrying data until the hole is filled.

Whenever an endpoint receives a SACK chunk that indicates that some TSNs are missing, it SHOULD wait for two further miss indications (via subsequent SACK chunks for a total of three missing reports) on the same TSNs before taking action with regard to Fast Retransmit.

Miss indications SHOULD follow the HTNA (Highest TSN Newly Acknowledged) algorithm. For each incoming SACK chunk, miss indications are incremented only for missing TSNs prior to the highest TSN newly acknowledged in the SACK chunk. A newly acknowledged DATA chunk is one not previously acknowledged in a SACK chunk. If an endpoint is in Fast Recovery and a SACK chunk arrives that advances the Cumulative TSN Ack Point, the miss indications are incremented for all TSNs reported missing in the SACK chunk.

When the third consecutive miss indication is received for a TSN(s), the data sender does the following:

- 1) Mark the DATA chunk(s) with three miss indications for retransmission.
- 2) If not in Fast Recovery, adjust the ssthresh and cwnd of the destination address(es) to which the missing DATA chunks were last sent, according to the formula described in Section 7.2.3.
- 3) If not in Fast Recovery, determine how many of the earliest (i.e., lowest TSN) DATA chunks marked for retransmission will fit into a single packet, subject to constraint of the PMTU of the destination transport address to which the packet is being sent. Call this value K. Retransmit those K DATA chunks in a single packet. When a Fast Retransmit is being performed, the sender SHOULD ignore the value of cwnd and SHOULD NOT delay retransmission for this single packet.
- 4) Restart the T3-rtx timer only if the last SACK chunk acknowledged the lowest outstanding TSN number sent to that address, or the endpoint is retransmitting the first outstanding DATA chunk sent to that address.
- 5) Mark the DATA chunk(s) as being fast retransmitted and thus ineligible for a subsequent Fast Retransmit. Those TSNs marked for retransmission due to the Fast-Retransmit algorithm that did not fit in the sent datagram carrying K other TSNs are also marked as ineligible for a subsequent Fast Retransmit. However, as they are marked for retransmission they will be retransmitted later on as soon as cwnd allows.
- 6) If not in Fast Recovery, enter Fast Recovery and mark the highest outstanding TSN as the Fast Recovery exit point. When a SACK chunk acknowledges all TSNs up to and including this exit point, Fast Recovery is exited. While in Fast Recovery, the ssthresh and cwnd SHOULD NOT change for any destinations due to a subsequent Fast Recovery event (i.e., one SHOULD NOT reduce the cwnd further due to a subsequent Fast Retransmit).

Note: Before the above adjustments, if the received SACK chunk also acknowledges new DATA chunks and advances the Cumulative TSN Ack Point, the cwnd adjustment rules defined in Section 7.2.1 and Section 7.2.2 MUST be applied first.

7.2.5. Reinitialization

During the lifetime of an SCTP association events can happen, which result in using the network under unknown new conditions. When detected by an SCTP implementation, the congestion control MUST be reinitialized.

7.2.5.1. 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.

7.2.5.2. Change of Routes

SCTP implementations MAY be aware of routing changes affecting packets sent to a destination address. In particular, this includes the selection of a different source address used for sending packets to a destination address. If such a routing change happens, the congestion control parameters for the affected destination addresses MUST be reset to their initial values.

7.3. PMTU Discovery

[RFC8899], [RFC8201], and [RFC1191] specify "Packetization Layer Path MTU Discovery", whereby an endpoint maintains an estimate of PMTU along a given Internet path and refrains from sending packets along that path that exceed the PMTU, other than occasional attempts to probe for a change in the PMTU. [RFC8899] is thorough in its discussion of the PMTU discovery mechanism and strategies for determining the current end-to-end PMTU setting as well as detecting changes in this value.

An endpoint SHOULD apply these techniques, and SHOULD do so on a per-destination-address basis.

There are two important SCTP-specific points regarding PMTU discovery:

- 1) SCTP associations can span multiple addresses. An endpoint **MUST** maintain separate PMTU estimates for each destination address of its peer.
- 2) The sender **SHOULD** track an AMDCS that will be the smallest PMDCS discovered for all of the peer's destination addresses. When fragmenting messages into multiple parts this AMDCS **SHOULD** be used to calculate the size of each DATA chunk. This will allow retransmissions to be seamlessly sent to an alternate address without encountering IP fragmentation.

8. Fault Management

8.1. Endpoint Failure Detection

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

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

8.2. Path Failure Detection

When its peer endpoint is multi-homed, an endpoint **SHOULD** keep an error counter for each of the destination transport addresses of the peer endpoint.

Each time the T3-rtx timer expires on any address, or when a HEARTBEAT chunk sent to an idle address is not acknowledged within an RTO, the error counter of that destination address will be incremented. When the value in the error counter exceeds the protocol parameter 'Path.Max.Retrans' of that destination address, the endpoint SHOULD mark the destination transport address as inactive, and a notification SHOULD be sent to the upper layer.

When an outstanding TSN is acknowledged or a HEARTBEAT chunk sent to that address is acknowledged with a HEARTBEAT ACK chunk, the endpoint SHOULD clear the error counter of the destination transport address to which the DATA chunk was last sent (or HEARTBEAT chunk was sent) and SHOULD also report to the upper layer when an inactive destination address is marked as active. When the peer endpoint is multi-homed and the last chunk sent to it was a retransmission to an alternate address, there exists an ambiguity as to whether or not the acknowledgement could be credited to the address of the last chunk sent. However, this ambiguity does not seem to have significant consequences for SCTP behavior. If this ambiguity is undesirable, the transmitter MAY choose not to clear the error counter if the last chunk sent was a retransmission.

Note: When configuring the SCTP endpoint, the user ought to avoid having the value of 'Association.Max.Retrans' larger than the summation of the 'Path.Max.Retrans' of all the destination addresses for the remote endpoint. Otherwise, all the destination addresses might become inactive while the endpoint still considers the peer endpoint reachable. When this condition occurs, how SCTP chooses to function is implementation specific.

When the primary path is marked inactive (due to excessive retransmissions, for instance), the sender MAY automatically transmit new packets to an alternate destination address if one exists and is active. If more than one alternate address is active when the primary path is marked inactive, only ONE transport address SHOULD be chosen and used as the new destination transport address.

8.3. Path Heartbeat

By default, an SCTP endpoint SHOULD monitor the reachability of the idle destination transport address(es) of its peer by sending a HEARTBEAT chunk periodically to the destination transport address(es). The sending of HEARTBEAT chunks MAY begin upon reaching the ESTABLISHED state and is discontinued after sending either a SHUTDOWN chunk or SHUTDOWN ACK chunk. A receiver of a HEARTBEAT chunks MUST respond to a HEARTBEAT chunk with a HEARTBEAT ACK chunk after entering the COOKIE-ECHOED state (sender of the INIT chunk) or the ESTABLISHED state (receiver of the INIT chunk), up until reaching

the SHUTDOWN-SENT state (sender of the SHUTDOWN chunk) or the SHUTDOWN-ACK-SENT state (receiver of the SHUTDOWN chunk).

A destination transport address is considered "idle" if no new chunk that can be used for updating path RTT (usually including first transmission DATA, INIT, COOKIE ECHO, or HEARTBEAT chunks, etc.) and no HEARTBEAT chunk has been sent to it within the current heartbeat period of that address. This applies to both active and inactive destination addresses.

The upper layer can optionally initiate the following functions:

- A) Disable heartbeat on a specific destination transport address of a given association,
- B) Change the 'HB.interval',
- C) Re-enable heartbeat on a specific destination transport address of a given association, and
- D) Request the sending of an on-demand HEARTBEAT chunk on a specific destination transport address of a given association.

The endpoint SHOULD increment the respective error counter of the destination transport address each time a HEARTBEAT chunk is sent to that address and not acknowledged within one RTO.

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

The sender of the HEARTBEAT chunk SHOULD include in the Heartbeat Information field of the chunk the current time when the packet is sent and the destination address to which the packet is sent.

Implementation Note: An alternative implementation of the heartbeat mechanism that can be used is to increment the error counter variable every time a HEARTBEAT chunk is sent to a destination. Whenever a HEARTBEAT ACK chunk arrives, the sender SHOULD clear the error counter of the destination that the HEARTBEAT chunk was sent to. This in effect would clear the previously stroked error (and any other error counts as well).

The receiver of the HEARTBEAT chunk SHOULD immediately respond with a HEARTBEAT ACK chunk that contains the Heartbeat Information TLV, together with any other received TLVs, copied unchanged from the received HEARTBEAT chunk.

Upon the receipt of the HEARTBEAT ACK chunk, the sender of the HEARTBEAT chunk SHOULD clear the error counter of the destination transport address to which the HEARTBEAT chunk 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 chunk. The receiver of the HEARTBEAT ACK chunk SHOULD also clear the association overall error count (as defined in Section 8.1).

The receiver of the HEARTBEAT ACK chunk SHOULD also perform an RTT measurement for that destination transport address using the time value carried in the HEARTBEAT ACK chunk.

On an idle destination address that is allowed to heartbeat, it is RECOMMENDED that a HEARTBEAT chunk is sent once per RTO of that destination address plus the protocol parameter 'HB.interval', with jittering of +/- 50% of the RTO value, and exponential backoff of the RTO if the previous HEARTBEAT chunk is unanswered.

A primitive is provided for the SCTP user to change the 'HB.interval' and turn on or off the heartbeat on a given destination address. The 'HB.interval' set by the SCTP user is added to the RTO of that destination (including any exponential backoff). Only one heartbeat SHOULD be sent each time the heartbeat timer expires (if multiple destinations are idle). It is an implementation decision on how to choose which of the candidate idle destinations to heartbeat to (if more than one destination is idle).

When tuning the 'HB.interval', there is a side effect that SHOULD be taken into account. When this value is increased, i.e., the time between the sending of HEARTBEAT chunks is longer, the detection of lost ABORT chunks takes longer as well. If a peer endpoint sends an ABORT chunk for any reason and the ABORT chunk is lost, the local endpoint will only discover the lost ABORT chunk by sending a DATA chunk or HEARTBEAT chunk (thus causing the peer to send another ABORT chunk). This is to be considered when tuning the heartbeat timer. If the sending of HEARTBEAT chunks is disabled, only sending DATA chunks to the association will discover a lost ABORT chunk from the peer.

8.4. Handle "Out of the Blue" Packets

An SCTP packet is called an "out of the blue" (OOTB) packet if it is correctly formed (i.e., passed the receiver's CRC32c check; see Section 6.8), but the receiver is not able to identify the association to which this packet belongs.

The receiver of an OOTB packet does the following:

- 1) If the OOTB packet is to or from a non-unicast address, a receiver SHOULD silently discard the packet. Otherwise,
- 2) If the OOTB packet contains an ABORT chunk, the receiver MUST silently discard the OOTB packet and take no further action. Otherwise,
- 3) If the packet contains an INIT chunk with a Verification Tag set to 0, it SHOULD be processed as described in Section 5.1. If, for whatever reason, the INIT chunk cannot be processed normally and an ABORT chunk has to be sent in response, the Verification Tag of the packet containing the ABORT chunk MUST be the Initiate Tag of the received INIT chunk, and the T bit of the ABORT chunk has to be set to 0, indicating that the Verification Tag is not reflected. Otherwise,
- 4) If the packet contains a COOKIE ECHO chunk as the first chunk, it MUST be processed as described in Section 5.1. Otherwise,
- 5) If the packet contains a SHUTDOWN ACK chunk, the receiver SHOULD respond to the sender of the OOTB packet with a SHUTDOWN COMPLETE chunk. When sending the SHUTDOWN COMPLETE chunk, the receiver of the OOTB packet MUST fill in the Verification Tag field of the outbound packet with the Verification Tag received in the SHUTDOWN ACK chunk and set the T bit in the Chunk Flags to indicate that the Verification Tag is reflected. Otherwise,
- 6) If the packet contains a SHUTDOWN COMPLETE chunk, the receiver SHOULD silently discard the packet and take no further action. Otherwise,
- 7) If the packet contains a ERROR chunk with the "Stale Cookie" error cause or a COOKIE ACK chunk, the SCTP packet SHOULD be silently discarded. Otherwise,
- 8) The receiver SHOULD respond to the sender of the OOTB packet with an ABORT chunk. When sending the ABORT chunk, the receiver of the OOTB packet MUST fill in the Verification Tag field of the outbound packet with the value found in the Verification Tag

field of the OOTB packet and set the T bit in the Chunk Flags to indicate that the Verification Tag is reflected. After sending this ABORT chunk, the receiver of the OOTB packet MUST discard the OOTB packet and MUST NOT take any further action.

8.5. Verification Tag

The Verification Tag rules defined in this section apply when sending or receiving SCTP packets that do not contain an INIT, SHUTDOWN COMPLETE, COOKIE ECHO (see Section 5.1), ABORT, or SHUTDOWN ACK chunk. The rules for sending and receiving SCTP packets containing one of these chunk types are discussed separately in Section 8.5.1.

When sending an SCTP packet, the endpoint MUST fill in the Verification Tag field of the outbound packet with the tag value in the Initiate Tag parameter of the INIT or INIT ACK chunk received from its peer.

When receiving an SCTP packet, the endpoint MUST ensure that the value in the Verification Tag field of the received SCTP packet matches its own tag. If the received Verification Tag value does not match the receiver's own tag value, the receiver MUST silently discard the packet and MUST NOT process it any further except for those cases listed in Section 8.5.1 below.

8.5.1. Exceptions in Verification Tag Rules

A) Rules for packets carrying an INIT chunk:

- * The sender MUST set the Verification Tag of the packet to 0.
- * When an endpoint receives an SCTP packet with the Verification Tag set to 0, it SHOULD verify that the packet contains only an INIT chunk. Otherwise, the receiver MUST silently discard the packet.

B) Rules for packets carrying an ABORT chunk:

- * The endpoint MUST always fill in the Verification Tag field of the outbound packet with the destination endpoint's tag value, if it is known.
- * If the ABORT chunk is sent in response to an OOTB packet, the endpoint MUST follow the procedure described in Section 8.4.
- * The receiver of an ABORT chunk MUST accept the packet if the Verification Tag field of the packet matches its own tag and the T bit is not set OR if it is set to its peer's tag and the T bit is set in the Chunk Flags. Otherwise, the receiver MUST silently discard the packet and take no further action.

C) Rules for packets carrying a SHUTDOWN COMPLETE chunk:

- * When sending a SHUTDOWN COMPLETE chunk, if the receiver of the SHUTDOWN ACK chunk has a TCB, then the destination endpoint's tag MUST be used, and the T bit MUST NOT be set. Only where no TCB exists SHOULD the sender use the Verification Tag from the SHUTDOWN ACK chunk, and MUST set the T bit.
- * The receiver of a SHUTDOWN COMPLETE chunk accepts the packet if the Verification Tag field of the packet matches its own tag and the T bit is not set OR if it is set to its peer's tag and the T bit is set in the Chunk Flags. Otherwise, the receiver MUST silently discard the packet and take no further action. An endpoint MUST ignore the SHUTDOWN COMPLETE chunk if it is not in the SHUTDOWN-ACK-SENT state.

D) Rules for packets carrying a COOKIE ECHO chunk:

- * When sending a COOKIE ECHO chunk, the endpoint MUST use the value of the Initiate Tag received in the INIT ACK chunk.
- * The receiver of a COOKIE ECHO chunk follows the procedures in Section 5.

E) Rules for packets carrying a SHUTDOWN ACK chunk:

- * If the receiver is in COOKIE-ECHOED or COOKIE-WAIT state the procedures in Section 8.4 SHOULD be followed; in other words, it is treated as an OOTB packet.

9. Termination of Association

An endpoint SHOULD terminate its association when it exits from service. An association can be terminated by either abort or shutdown. An abort of an association is abortive by definition in that any data pending on either end of the association is discarded and not delivered to the peer. A shutdown of an association is considered a graceful close where all data in queue by either endpoint is delivered to the respective peers. However, in the case of a shutdown, SCTP does not support a half-open state (like TCP) wherein one side might continue sending data while the other end is closed. When either endpoint performs a shutdown, the association on each peer will stop accepting new data from its user and only deliver data in queue at the time of sending or receiving the SHUTDOWN chunk.

9.1. Abort of an Association

When an endpoint decides to abort an existing association, it MUST send an ABORT chunk to its peer endpoint. The sender MUST fill in the peer's Verification Tag in the outbound packet and MUST NOT bundle any DATA chunk with the ABORT chunk. If the association is aborted on request of the upper layer, a "User-Initiated Abort" error cause (see Section 3.3.10.12) SHOULD be present in the ABORT chunk.

An endpoint MUST NOT respond to any received packet that contains an ABORT chunk (also see Section 8.4).

An endpoint receiving an ABORT chunk MUST apply the special Verification Tag check rules described in Section 8.5.1.

After checking the Verification Tag, the receiving endpoint MUST remove the association from its record and SHOULD report the termination to its upper layer. If a "User-Initiated Abort" error cause is present in the ABORT chunk, the Upper Layer Abort Reason SHOULD be made available to the upper layer.

9.2. Shutdown of an Association

Using the SHUTDOWN primitive (see Section 11.1), the upper layer of an endpoint in an association can gracefully close the association. This will allow all outstanding DATA chunks from the peer of the shutdown initiator to be delivered before the association terminates.

Upon receipt of the SHUTDOWN primitive from its upper layer, the endpoint enters the SHUTDOWN-PENDING state and remains there until all outstanding data has been acknowledged by its peer. The endpoint accepts no new data from its upper layer, but retransmits data to the peer endpoint if necessary to fill gaps.

Once all its outstanding data has been acknowledged, the endpoint sends a SHUTDOWN chunk to its peer including in the Cumulative TSN Ack field the last sequential TSN it has received from the peer. It SHOULD then start the T2-shutdown timer and enter the SHUTDOWN-SENT state. If the timer expires, the endpoint MUST resend the SHUTDOWN chunk with the updated last sequential TSN received from its peer.

The rules in Section 6.3 MUST be followed to determine the proper timer value for T2-shutdown. To indicate any gaps in TSN, the endpoint MAY also bundle a SACK chunk with the SHUTDOWN chunk in the same SCTP packet.

An endpoint SHOULD limit the number of retransmissions of the SHUTDOWN chunk to the protocol parameter 'Association.Max.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). The reception of any packet from its peer (i.e., as the peer sends all of its queued DATA chunks) SHOULD clear the endpoint's retransmission count and restart the T2-shutdown timer, giving its peer ample opportunity to transmit all of its queued DATA chunks that have not yet been sent.

Upon reception of the SHUTDOWN chunk, the peer endpoint does the following:

- * enter the SHUTDOWN-RECEIVED state,
- * stop accepting new data from its SCTP user, and
- * verify, by checking the Cumulative TSN Ack field of the chunk, that all its outstanding DATA chunks have been received by the SHUTDOWN chunk sender.

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

If there are still outstanding DATA chunks left, the SHUTDOWN chunk receiver MUST continue to follow normal data transmission procedures defined in Section 6, until all outstanding DATA chunks are acknowledged; however, the SHUTDOWN chunk receiver MUST NOT accept new data from its SCTP user.

While in the SHUTDOWN-SENT state, the SHUTDOWN chunk sender MUST immediately respond to each received packet containing one or more DATA chunks with a SHUTDOWN chunk and restart the T2-shutdown timer. If a SHUTDOWN chunk by itself cannot acknowledge all of the received DATA chunks (i.e., there are TSNs that can be acknowledged that are larger than the cumulative TSN, and thus gaps exist in the TSN sequence), or if duplicate TSNs have been received, then a SACK chunk MUST also be sent.

The sender of the SHUTDOWN chunk MAY also start an overall guard timer T5-shutdown-guard to bound the overall time for the shutdown sequence. At the expiration of this timer, the sender SHOULD abort the association by sending an ABORT chunk. If the T5-shutdown-guard timer is used, it SHOULD be set to the RECOMMENDED value of 5 times 'RTO.Max'.

If the receiver of the SHUTDOWN chunk has no more outstanding DATA chunks, the SHUTDOWN chunk receiver MUST send a SHUTDOWN ACK chunk and start a T2-shutdown timer of its own, entering the SHUTDOWN-ACK-SENT state. If the timer expires, the endpoint MUST resend the SHUTDOWN ACK chunk.

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

Upon the receipt of the SHUTDOWN ACK chunk, the sender of the SHUTDOWN chunk MUST stop the T2-shutdown timer, send a SHUTDOWN COMPLETE chunk to its peer, and remove all record of the association.

Upon reception of the SHUTDOWN COMPLETE chunk, the endpoint verifies that it is in the SHUTDOWN-ACK-SENT state; if it is not, the chunk SHOULD be discarded. If the endpoint is in the SHUTDOWN-ACK-SENT state, the endpoint SHOULD stop the T2-shutdown timer and remove all knowledge of the association (and thus the association enters the CLOSED state).

An endpoint SHOULD ensure that all its outstanding DATA chunks have been acknowledged before initiating the shutdown procedure.

An endpoint SHOULD reject any new data request from its upper layer if it is in the SHUTDOWN-PENDING, SHUTDOWN-SENT, SHUTDOWN-RECEIVED, or SHUTDOWN-ACK-SENT state.

If an endpoint is in the SHUTDOWN-ACK-SENT state and receives an INIT chunk (e.g., if the SHUTDOWN COMPLETE chunk was lost) with source and destination transport addresses (either in the IP addresses or in the INIT chunk) that belong to this association, it SHOULD discard the INIT chunk and retransmit the SHUTDOWN ACK chunk.

Note: Receipt of a packet containing an INIT chunk with the same source and destination IP addresses as used in transport addresses assigned to an endpoint but with a different port number indicates the initialization of a separate association.

The sender of the INIT or COOKIE ECHO chunk SHOULD respond to the receipt of a SHUTDOWN ACK chunk with a stand-alone SHUTDOWN COMPLETE chunk in an SCTP packet with the Verification Tag field of its common header set to the same tag that was received in the packet containing the SHUTDOWN ACK chunk. This is considered an OOTB packet as defined in Section 8.4. The sender of the INIT chunk lets T1-init continue

running and remains in the COOKIE-WAIT or COOKIE-ECHOED state. Normal T1-init timer expiration will cause the INIT or COOKIE chunk to be retransmitted and thus start a new association.

If a SHUTDOWN chunk is received in the COOKIE-WAIT or COOKIE ECHOED state, the SHUTDOWN chunk SHOULD be silently discarded.

If an endpoint is in the SHUTDOWN-SENT state and receives a SHUTDOWN chunk from its peer, the endpoint SHOULD respond immediately with a SHUTDOWN ACK chunk to its peer, and move into the SHUTDOWN-ACK-SENT state restarting its T2-shutdown timer.

If an endpoint is in the SHUTDOWN-ACK-SENT state and receives a SHUTDOWN ACK, it MUST stop the T2-shutdown timer, send a SHUTDOWN COMPLETE chunk to its peer, and remove all record of the association.

10. ICMP Handling

Whenever an ICMP message is received by an SCTP endpoint, the following procedures MUST be followed to ensure proper utilization of the information being provided by layer 3.

- ICMP1) An implementation MAY ignore all ICMPv4 messages where the type field is not set to "Destination Unreachable".
- ICMP2) An implementation MAY ignore all ICMPv6 messages where the type field is not "Destination Unreachable", "Parameter Problem", or "Packet Too Big".
- ICMP3) An implementation SHOULD ignore any ICMP messages where the code indicates "Port Unreachable".
- ICMP4) An implementation MAY ignore all ICMPv6 messages of type "Parameter Problem" if the code is not "Unrecognized Next Header Type Encountered".
- ICMP5) An implementation MUST use the payload of the ICMP message (v4 or v6) to locate the association that sent the message to which ICMP is responding. If the association cannot be found, an implementation SHOULD ignore the ICMP message.

- ICMP6) An implementation MUST validate that the Verification Tag contained in the ICMP message matches the Verification Tag of the peer. If the Verification Tag is not 0 and does not match, discard the ICMP message. If it is 0 and the ICMP message contains enough bytes to verify that the chunk type is an INIT chunk and that the Initiate Tag matches the tag of the peer, continue with ICMP7. If the ICMP message is too short or the chunk type or the Initiate Tag does not match, silently discard the packet.
- ICMP7) If the ICMP message is either an ICMPv6 message of type "Packet Too Big" or an ICMPv4 message of type "Destination Unreachable" and code "Fragmentation Needed", an implementation SHOULD process this information as defined for PMTU discovery.
- ICMP8) If the ICMP code is an "Unrecognized Next Header Type Encountered" or a "Protocol Unreachable", an implementation MUST treat this message as an abort with the T bit set if it does not contain an INIT chunk. If it does contain an INIT chunk and the association is in the COOKIE-WAIT state, handle the ICMP message like an ABORT chunk.
- ICMP9) If the ICMP type is "Destination Unreachable", the implementation MAY move the destination to the unreachable state or, alternatively, increment the path error counter. SCTP MAY provide information to the upper layer indicating the reception of ICMP messages when reporting a network status change.

These procedures differ from [RFC1122] and from its requirements for processing of port-unreachable messages and the requirements that an implementation MUST abort associations in response to a "protocol unreachable" message. Port-unreachable messages are not processed, since an implementation will send an ABORT chunk, not a port unreachable. The stricter handling of the "protocol unreachable" message is due to security concerns for hosts that do not support SCTP.

11. Interface with Upper Layer

The Upper Layer Protocols (ULPs) request services by passing primitives to SCTP and receive notifications from SCTP for various events.

The primitives and notifications described in this section can be used as a guideline for implementing SCTP. The following functional description of ULP interface primitives is shown for illustrative

purposes. Different SCTP implementations can have different ULP interfaces. However, all SCTP implementations are expected to provide a certain minimum set of services to guarantee that all SCTP implementations can support the same protocol hierarchy.

Please note that this section is informational only.

[RFC6458] and the Socket API Considerations section of [RFC7053] define an extension of the socket API for SCTP as described in this document.

11.1. ULP-to-SCTP

The following sections functionally characterize a ULP/SCTP interface. The notation used is similar to most procedure or function calls in high-level languages.

The ULP primitives described below specify the basic functions that SCTP performs to support inter-process communication. Individual implementations define their own exact format, and provide combinations or subsets of the basic functions in single calls.

11.1.1. Initialize

```
INITIALIZE ([local port],[local eligible address list])  
-> local SCTP instance name
```

This primitive allows SCTP to initialize its internal data structures and allocate necessary resources for setting up its operation environment. Once SCTP is initialized, ULP can communicate directly with other endpoints without re-invoking this primitive.

SCTP will return a local SCTP instance name to the ULP.

Mandatory attributes:
None.

Optional attributes:
local port: SCTP port number, if ULP wants it to be specified.
local eligible address list: an address list that the local SCTP endpoint binds. By default, if an address list is not included, all IP addresses assigned to the host are used by the local endpoint.

Implementation Note: If this optional attribute is supported by an implementation, it will be the responsibility of the implementation to enforce that the IP source address field of any SCTP packets sent by this endpoint contains one of the IP addresses indicated in the local eligible address list.

11.1.2. Associate

```
ASSOCIATE(local SCTP instance name,  
initial destination transport addr list, outbound stream count)  
-> association id [,destination transport addr list]  
[,outbound stream count]
```

This primitive allows the upper layer to initiate an association to a specific peer endpoint.

The peer endpoint is specified by one or more of the transport addresses that defines the endpoint (see Section 2.3). If the local SCTP instance has not been initialized, the ASSOCIATE is considered an error.

An association id, which is a local handle to the SCTP association, will be returned on successful establishment of the association. If SCTP is not able to open an SCTP association with the peer endpoint, an error is returned.

Other association parameters can be returned, including the complete destination transport addresses of the peer as well as the outbound stream count of the local endpoint. One of the transport addresses from the returned destination addresses will be selected by the local endpoint as default primary path for sending SCTP packets to this peer. The returned "destination transport addr list" can be used by the ULP to change the default primary path or to force sending a packet to a specific transport address.

Implementation Note: If ASSOCIATE primitive is implemented as a blocking function call, the ASSOCIATE primitive can return association parameters in addition to the association id upon successful establishment. If ASSOCIATE primitive is implemented as a non-blocking call, only the association id is returned and association parameters are passed using the COMMUNICATION UP notification.

Mandatory attributes:

local SCTP instance name: obtained from the INITIALIZE operation.

initial destination transport addr list: a non-empty list of

transport addresses of the peer endpoint with which the association is to be established.

outbound stream count: the number of outbound streams the ULP would like to open towards this peer endpoint.

Optional attributes:
None.

11.1.3. Shutdown

SHUTDOWN(association id) -> result

Gracefully closes an association. Any locally queued user data will be delivered to the peer. The association will be terminated only after the peer acknowledges all the SCTP packets sent. A success code will be returned on successful termination of the association. If attempting to terminate the association results in a failure, an error code is returned.

Mandatory attributes:
association id: local handle to the SCTP association.

Optional attributes:
None.

11.1.4. Abort

ABORT(association id [, Upper Layer Abort Reason]) -> result

Ungracefully closes an association. Any locally queued user data will be discarded, and an ABORT chunk is sent to the peer. A success code will be returned on successful abort of the association. If attempting to abort the association results in a failure, an error code is returned.

Mandatory attributes:
association id: local handle to the SCTP association.

Optional attributes:
Upper Layer Abort Reason: reason of the abort to be passed to the peer.

11.1.5. Send

```
SEND(association id, buffer address, byte count [,context]  
[,stream id] [,life time] [,destination transport address]  
[,unordered flag] [,no-bundle flag] [,payload protocol-id]  
[,sack-immediately flag]) -> result
```

This is the main method to send user data via SCTP.

Mandatory attributes:

association id: local handle to the SCTP association.

buffer address: the location where the user message to be transmitted is stored.

byte count: the size of the user data in number of bytes.

Optional attributes:

context: an optional information provided that will be carried in the sending failure notification to the ULP if the transportation of this user message fails.

stream id: to indicate which stream to send the data on. If not specified, stream 0 will be used.

life time: specifies the life time of the user data. The user data will not be sent by SCTP after the life time expires. This parameter can be used to avoid efforts to transmit stale user messages. SCTP notifies the ULP if the data cannot be initiated to transport (i.e., sent to the destination via SCTP's SEND primitive) within the life time variable. However, the user data will be transmitted if SCTP has attempted to transmit a chunk before the life time expired.

Implementation Note: In order to better support the data life time option, the transmitter can hold back the assigning of the TSN number to an outbound DATA chunk to the last moment. And, for implementation simplicity, once a TSN number has been assigned the sender considers the send of this DATA chunk as committed, overriding any life time option attached to the DATA chunk.

destination transport address: specified as one of the destination transport addresses of the peer endpoint to which this packet is sent. Whenever possible, SCTP uses this destination transport address for sending the packets, instead of the current primary path.

unordered flag: this flag, if present, indicates that the user

would like the data delivered in an unordered fashion to the peer (i.e., the U flag is set to 1 on all DATA chunks carrying this message).

no-bundle flag: instructs SCTP not to delay the sending of DATA chunks for this user data just to allow it to be bundled with other outbound DATA chunks. When faced with network congestion, SCTP might still bundle the data, even when this flag is present.

payload protocol-id: a 32-bit unsigned integer that is to be passed to the peer indicating the type of payload protocol data being transmitted. Note that the upper layer is responsible for the host to network byte order conversion of this field, which is passed by SCTP as 4 bytes of opaque data.

sack-immediately flag: set the I bit on the last DATA chunk used for the user message to be transmitted.

11.1.6. Set Primary

SETPRIMARY(association id, destination transport address,
[source transport address]) -> result

Instructs the local SCTP to use the specified destination transport address as the primary path for sending packets.

The result of attempting this operation is returned. If the specified destination transport address is not present in the "destination transport address list" returned earlier in an associate command or communication up notification, an error is returned.

Mandatory attributes:

association id: local handle to the SCTP association.

destination transport address: specified as one of the transport addresses of the peer endpoint, which is used as the primary address for sending packets. This overrides the current primary address information maintained by the local SCTP endpoint.

Optional attributes:

source transport address: optionally, some implementations can allow you to set the default source address placed in all outgoing IP datagrams.

11.1.7. Receive


```
RECEIVE(association id, buffer address, buffer size [,stream id])  
-> byte count [,transport address] [,stream id]  
[,stream sequence number] [,partial flag] [,payload protocol-id]
```

This primitive reads the first user message in the SCTP in-queue into the buffer specified by ULP, if there is one available. The size of the message read, in bytes, will be returned. It might, depending on the specific implementation, also return other information such as the sender's address, the stream id on which it is received, whether there are more messages available for retrieval, etc. For ordered messages, their Stream Sequence Number might also be returned.

Depending upon the implementation, if this primitive is invoked when no message is available the implementation returns an indication of this condition or blocks the invoking process until data does become available.

Mandatory attributes:

association id: local handle to the SCTP association

buffer address: the memory location indicated by the ULP to store the received message.

buffer size: the maximum size of data to be received, in bytes.

Optional attributes:

stream id: to indicate which stream to receive the data on.

stream sequence number: the Stream Sequence Number assigned by the sending SCTP peer.

partial flag: if this returned flag is set to 1, then this primitive contains a partial delivery of the whole message. When this flag is set, the stream id and stream sequence number accompanies this primitive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.

payload protocol-id: a 32-bit unsigned integer that is received from the peer indicating the type of payload protocol of the received data. Note that the upper layer is responsible for the host to network byte order conversion of this field, which is passed by SCTP as 4 bytes of opaque data.

11.1.8. Status

```
STATUS(association id) -> status data
```

This primitive returns a data block containing the following information:

- * association connection state,
- * destination transport address list,
- * destination transport address reachability states,
- * current receiver window size,
- * current congestion window sizes,
- * number of unacknowledged DATA chunks,
- * number of DATA chunks pending receipt,
- * primary path,
- * most recent SRTT on primary path,
- * RTO on primary path,
- * SRTT and RTO on other destination addresses, etc.

Mandatory attributes:

association id: local handle to the SCTP association.

Optional attributes:

None.

11.1.9. Change Heartbeat

CHANGE HEARTBEAT(association id, destination transport address, new state [,interval]) -> result

Instructs the local endpoint to enable or disable heartbeat on the specified destination transport address.

The result of attempting this operation is returned.

Note: Even when enabled, heartbeat will not take place if the destination transport address is not idle.

Mandatory attributes:

association id: local handle to the SCTP association.

destination transport address: specified as one of the transport

addresses of the peer endpoint.

new state: the new state of heartbeat for this destination transport address (either enabled or disabled).

Optional attributes:

interval: if present, indicates the frequency of the heartbeat if this is to enable heartbeat on a destination transport address. This value is added to the RTO of the destination transport address. This value, if present, affects all destinations.

11.1.10. Request Heartbeat

REQUESTHEARTBEAT(association id, destination transport address)
-> result

Instructs the local endpoint to perform a heartbeat on the specified destination transport address of the given association. The returned result indicates whether the transmission of the HEARTBEAT chunk to the destination address is successful.

Mandatory attributes:

association id: local handle to the SCTP association.

destination transport address: the transport address of the association on which a heartbeat is issued.

Optional attributes:

None.

11.1.11. Get SRTT Report

GETSRTTREPORT(association id, destination transport address)
-> srtt result

Instructs the local SCTP to report the current SRTT measurement on the specified destination transport address of the given association. The returned result can be an integer containing the most recent SRTT in milliseconds.

Mandatory attributes:

association id: local handle to the SCTP association.

destination transport address: the transport address of the association on which the SRTT measurement is to be reported.

Optional attributes:

None.

11.1.12. Set Failure Threshold

```
SETFAILURETHRESHOLD(association id, destination transport address,  
failure threshold) -> result
```

This primitive allows the local SCTP to customize the reachability failure detection threshold 'Path.Max.Retrans' for the specified destination address. Note that this can also be done using the SETPROTOCOLPARAMETERS primitive (Section 11.1.13).

Mandatory attributes:

association id: local handle to the SCTP association.

destination transport address: the transport address of the association on which the failure detection threshold is to be set.

failure threshold: the new value of 'Path.Max.Retrans' for the destination address.

Optional attributes:

None.

11.1.13. Set Protocol Parameters

```
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 16), or other parameters like the DSCP) that the SCTP user wishes to customize.

Optional attributes:

destination transport address: some of the protocol parameters might be set on a per destination transport address basis.

11.1.14. Receive Unsent Message

```
RECEIVE_UNSENT(data retrieval id, buffer address, buffer size  
[,stream id] [, stream sequence number] [,partial flag]  
[,payload protocol-id])
```

This primitive reads a user message, which has never been sent, into the buffer specified by ULP.

Mandatory attributes:

data retrieval id: the identification passed to the ULP in the failure notification.

buffer address: the memory location indicated by the ULP to store the received message.

buffer size: the maximum size of data to be received, in bytes.

Optional attributes:

stream id: this is a return value that is set to indicate which stream the data was sent to.

stream sequence number: this value is returned indicating the Stream Sequence Number that was associated with the message.

partial flag: if this returned flag is set to 1, then this message is a partial delivery of the whole message. When this flag is set, the stream id and stream sequence number accompanies this primitive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.

payload protocol-id: The 32 bit unsigned integer that was set to be sent to the peer indicating the type of payload protocol of the received data.

11.1.15. Receive Unacknowledged Message

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

This primitive reads a user message, which has been sent and has not been acknowledged by the peer, into the buffer specified by ULP.

Mandatory attributes:

data retrieval id: the identification passed to the ULP in the failure notification.

buffer address: the memory location indicated by the ULP to store

the received message.

buffer size: the maximum size of data to be received, in bytes.

Optional attributes:

stream id: this is a return value that is set to indicate which stream the data was sent to.

stream sequence number: this value is returned indicating the Stream Sequence Number that was associated with the message.

partial flag: if this returned flag is set to 1, then this message is a partial delivery of the whole message. When this flag is set, the stream id and stream sequence number accompanies this primitive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.

payload protocol-id: the 32-bit unsigned integer that was sent to the peer indicating the type of payload protocol of the received data.

11.1.16. Destroy SCTP Instance

DESTROY(local SCTP instance name)

Mandatory attributes:

local SCTP instance name: this is the value that was passed to the application in the initialize primitive and it indicates which SCTP instance is to be destroyed.

Optional attributes:

None.

11.2. SCTP-to-ULP

It is assumed that the operating system or application environment provides a means for the SCTP to asynchronously signal the ULP process. When SCTP does signal a ULP process, certain information is passed to the ULP.

Implementation Note: In some cases, this might be done through a separate socket or error channel.

11.2.1. DATA ARRIVE Notification

SCTP invokes this notification on the ULP when a user message is successfully received and ready for retrieval.

The following might optionally be passed with the notification:

association id: local handle to the SCTP association.

stream id: to indicate which stream the data is received on.

11.2.2. SEND FAILURE Notification

If a message cannot be delivered, SCTP invokes this notification on the ULP.

The following might optionally be passed with the notification:

association id: local handle to the SCTP association.

data retrieval id: an identification used to retrieve unsent and unacknowledged data.

mode: Indicate whether no part of the message never has been sent or if at least part of it has been sent but it is not completely acknowledged.

cause code: indicating the reason of the failure, e.g., size too large, message life time expiration, etc.

context: optional information associated with this message (see Section 11.1.5).

11.2.3. NETWORK STATUS CHANGE Notification

When a destination transport address is marked inactive (e.g., when SCTP detects a failure) or marked active (e.g., when SCTP detects a recovery), SCTP invokes this notification on the ULP.

The following is passed with the notification:

association id: local handle to the SCTP association.

destination transport address: this indicates the destination transport address of the peer endpoint affected by the change.

new-status: this indicates the new status.

11.2.4. COMMUNICATION UP Notification

This notification is used when SCTP becomes ready to send or receive user messages, or when a lost communication to an endpoint is restored.

Implementation Note: If the ASSOCIATE primitive is implemented as a blocking function call, the association parameters are returned as a result of the ASSOCIATE primitive itself. In that case, COMMUNICATION UP notification is optional at the association initiator's side.

The following is passed with the notification:

association id: local handle to the SCTP association.

status: This indicates what type of event has occurred.

destination transport address list: the complete set of transport addresses of the peer.

outbound stream count: the maximum number of streams allowed to be used in this association by the ULP.

inbound stream count: the number of streams the peer endpoint has requested with this association (this might not be the same number as 'outbound stream count').

11.2.5. COMMUNICATION LOST Notification

When SCTP loses communication to an endpoint completely (e.g., via Heartbeats) or detects that the endpoint has performed an abort operation, it invokes this notification on the ULP.

The following is passed with the notification:

association id: local handle to the SCTP association.

status: this indicates what type of event has occurred; the status might indicate that a failure OR a normal termination event occurred in response to a shutdown or abort request.

The following might be passed with the notification:

last-acked: the TSN last acked by that peer endpoint.

last-sent: the TSN last sent to that peer endpoint.

Upper Layer Abort Reason: the abort reason specified in case of a user-initiated abort.

11.2.6. COMMUNICATION ERROR Notification

When SCTP receives an ERROR chunk from its peer and decides to notify its ULP, it can invoke this notification on the ULP.

The following can be passed with the notification:

association id: local handle to the SCTP association.

error info: this indicates the type of error and optionally some additional information received through the ERROR chunk.

11.2.7. RESTART Notification

When SCTP detects that the peer has restarted, it might send this notification to its ULP.

The following can be passed with the notification:

association id: local handle to the SCTP association.

11.2.8. SHUTDOWN COMPLETE Notification

When SCTP completes the shutdown procedures (Section 9.2), this notification is passed to the upper layer.

The following can be passed with the notification:

association id: local handle to the SCTP association.

12. Security Considerations

12.1. Security Objectives

As a common transport protocol designed to reliably carry time-sensitive user messages, such as billing or signaling messages for telephony services, between two networked endpoints, SCTP has the following security objectives.

- * availability of reliable and timely data transport services
- * integrity of the user-to-user information carried by SCTP

12.2. SCTP Responses to Potential Threats

SCTP could potentially be used in a wide variety of risk situations. It is important for operators of systems running SCTP to analyze their particular situations and decide on the appropriate counter-measures.

Operators of systems running SCTP might consult [RFC2196] for guidance in securing their site.

12.2.1. Countering Insider Attacks

The principles of [RFC2196] might be applied to minimize the risk of theft of information or sabotage by insiders. Such procedures include publication of security policies, control of access at the physical, software, and network levels, and separation of services.

12.2.2. Protecting against Data Corruption in the Network

Where the risk of undetected errors in datagrams delivered by the lower-layer transport services is considered to be too great, additional integrity protection is required. If this additional protection were provided in the application layer, the SCTP header would remain vulnerable to deliberate integrity attacks. While the existing SCTP mechanisms for detection of packet replays are considered sufficient for normal operation, stronger protections are needed to protect SCTP when the operating environment contains significant risk of deliberate attacks from a sophisticated adversary.

The SCTP Authentication extension SCTP-AUTH [RFC4895] MAY be used when the threat environment requires stronger integrity protections, but does not require confidentiality.

12.2.3. Protecting Confidentiality

In most cases, the risk of breach of confidentiality applies to the signaling data payload, not to the SCTP or lower-layer protocol overheads. If that is true, encryption of the SCTP user data only might be considered. As with the supplementary checksum service, user data encryption MAY be performed by the SCTP user application. [RFC6083] MAY be used for this. Alternately, the user application MAY use an implementation-specific API to request that the IP Encapsulating Security Payload (ESP) [RFC4303] be used to provide confidentiality and integrity.

Particularly for mobile users, the requirement for confidentiality might include the masking of IP addresses and ports. In this case, ESP SHOULD be used instead of application-level confidentiality. If ESP is used to protect confidentiality of SCTP traffic, an ESP cryptographic transform that includes cryptographic integrity protection MUST be used, because if there is a confidentiality threat there will also be a strong integrity threat.

Regardless of where confidentiality is provided, the Internet Key Exchange Protocol version 2 (IKEv2) [RFC7296] SHOULD be used for key management of ESP.

Operators might consult [RFC4301] for more information on the security services available at and immediately above the Internet Protocol layer.

12.2.4. Protecting against Blind Denial-of-Service Attacks

A blind attack is one where the attacker is unable to intercept or otherwise see the content of data flows passing to and from the target SCTP node. Blind denial-of-service attacks can take the form of flooding, masquerade, or improper monopolization of services.

12.2.4.1. Flooding

The objective of flooding is to cause loss of service and incorrect behavior at target systems through resource exhaustion, interference with legitimate transactions, and exploitation of buffer-related software bugs. Flooding can be directed either at the SCTP node or at resources in the intervening IP Access Links or the Internet. Where the latter entities are the target, flooding will manifest itself as loss of network services, including potentially the breach of any firewalls in place.

In general, protection against flooding begins at the equipment design level, where it includes measures such as:

- * avoiding commitment of limited resources before determining that the request for service is legitimate.
- * giving priority to completion of processing in progress over the acceptance of new work.
- * identification and removal of duplicate or stale queued requests for service.
- * not responding to unexpected packets sent to non-unicast addresses.

Network equipment is expected to be capable of generating an alarm and log if a suspicious increase in traffic occurs. The log provides information such as the identity of the incoming link and source address(es) used, which will help the network or SCTP system operator to take protective measures. Procedures are expected to be in place for the operator to act on such alarms if a clear pattern of abuse emerges.

The design of SCTP is resistant to flooding attacks, particularly in its use of a four-way startup handshake, its use of a cookie to defer commitment of resources at the responding SCTP node until the handshake is completed, and its use of a Verification Tag to prevent insertion of extraneous packets into the flow of an established association.

ESP might be useful in reducing the risk of certain kinds of denial-of-service attacks.

Support for the Host Name Address parameter has been removed from the protocol. Endpoints receiving INIT or INIT ACK chunks containing the Host Name Address parameter MUST send an ABORT chunk in response and MAY include an "Unresolvable Address" error cause.

12.2.4.2. Blind Masquerade

Masquerade can be used to deny service in several ways:

- * by tying up resources at the target SCTP node to which the impersonated node has limited access. For example, the target node can by policy permit a maximum of one SCTP association with the impersonated SCTP node. The masquerading attacker can attempt to establish an association purporting to come from the impersonated node so that the latter cannot do so when it requires it.
- * by deliberately allowing the impersonation to be detected, thereby provoking counter-measures that cause the impersonated node to be locked out of the target SCTP node.
- * by interfering with an established association by inserting extraneous content such as a SHUTDOWN chunk.

SCTP reduces the risk of blind masquerade attacks through IP spoofing by use of the four-way startup handshake. Because the initial exchange is memory-less, no lockout mechanism is triggered by blind masquerade attacks. In addition, the packet containing the INIT ACK chunk with the State Cookie is transmitted back to the IP address from which it received the packet containing the INIT chunk. Thus,

the attacker would not receive the INIT ACK chunk containing the State Cookie. SCTP protects against insertion of extraneous packets into the flow of an established association by use of the Verification Tag.

Logging of received INIT chunks and abnormalities such as unexpected INIT ACK chunks might be considered as a way to detect patterns of hostile activity. However, the potential usefulness of such logging has to be weighed against the increased SCTP startup processing it implies, rendering the SCTP node more vulnerable to flooding attacks. Logging is pointless without the establishment of operating procedures to review and analyze the logs on a routine basis.

12.2.4.3. Improper Monopolization of Services

Attacks under this heading are performed openly and legitimately by the attacker. They are directed against fellow users of the target SCTP node or of the shared resources between the attacker and the target node. Possible attacks include the opening of a large number of associations between the attacker's node and the target, or transfer of large volumes of information within a legitimately established association.

Policy limits are expected to be placed on the number of associations per adjoining SCTP node. SCTP user applications are expected to be capable of detecting large volumes of illegitimate or "no-op" messages within a given association and either logging or terminating the association as a result, based on local policy.

12.3. SCTP Interactions with Firewalls

It is helpful for some firewalls if they can inspect just the first fragment of a fragmented SCTP packet and unambiguously determine whether it corresponds to an INIT chunk (for further information, please refer to [RFC1858]). Accordingly, we stress the requirements, as stated in Section 3.1, that (1) an INIT chunk MUST NOT be bundled with any other chunk in a packet and (2) a packet containing an INIT chunk MUST have a zero Verification Tag. The receiver of an INIT chunk MUST silently discard the INIT chunk and all further chunks if the INIT chunk is bundled with other chunks or the packet has a non-zero Verification Tag.

12.4. Protection of Non-SCTP-Capable Hosts

To provide a non-SCTP-capable host with the same level of protection against attacks as for SCTP-capable ones, all SCTP implementations MUST implement the ICMP handling described in Section 10.

When an SCTP implementation receives a packet containing multiple control or DATA chunks and the processing of the packet would result in sending multiple chunks in response, the sender of the response chunk(s) MUST NOT send more than one packet containing chunks other than DATA chunks. This requirement protects the network for triggering a packet burst in response to a single packet. If bundling is supported, multiple response chunks that fit into a single packet MAY be bundled together into one single response packet. If bundling is not supported, then the sender MUST NOT send more than one response chunk and MUST discard all other responses. Note that this rule does not apply to a SACK chunk, since a SACK chunk is, in itself, a response to DATA chunks and a SACK chunk does not require a response of more DATA chunks.

An SCTP implementation MUST abort the association if it receives a SACK chunk acknowledging a TSN that has not been sent.

An SCTP implementation that receives an INIT chunk that would require a large packet in response, due to the inclusion of multiple "Unrecognized Parameter" parameters, MAY (at its discretion) elect to omit some or all of the "Unrecognized Parameter" parameters to reduce the size of the INIT ACK chunk. Due to a combination of the size of the State Cookie parameter and the number of addresses a receiver of an INIT chunk indicates to a peer, it is always possible that the INIT ACK chunk will be larger than the original INIT chunk. An SCTP implementation SHOULD attempt to make the INIT ACK chunk as small as possible to reduce the possibility of byte amplification attacks.

13. Network Management Considerations

The MIB module for SCTP defined in [RFC3873] applies for the version of the protocol specified in this document.

14. Recommended Transmission Control Block (TCB) Parameters

This section details a set of parameters that are expected to be contained within the TCB for an implementation. This section is for illustrative purposes and is not considered to be requirements on an implementation or as an exhaustive list of all parameters inside an SCTP TCB. Each implementation might need its own additional parameters for optimization.

14.1. Parameters Necessary for the SCTP Instance

Associations: A list of current associations and mappings to the

data consumers for each association. This might be in the form of a hash table or other implementation-dependent structure. The data consumers might be process identification information such as file descriptors, named pipe pointer, or table pointers dependent on how SCTP is implemented.

Secret Key: A secret key used by this endpoint to compute the MAC. This SHOULD be a cryptographic quality random number with a sufficient length. Discussion in [RFC4086] can be helpful in selection of the key.

Address List: The list of IP addresses that this instance has bound. This information is passed to one's peer(s) in INIT and INIT ACK chunks.

SCTP Port: The local SCTP port number to which the endpoint is bound.

14.2. Parameters Necessary per Association (i.e., the TCB)

Peer Verification Tag: Tag value to be sent in every packet and is received in the INIT or INIT ACK chunk.

My Verification Tag: Tag expected in every inbound packet and sent in the INIT or INIT ACK chunk.

State: COOKIE-WAIT, COOKIE-ECHOED, ESTABLISHED, SHUTDOWN-PENDING, SHUTDOWN-SENT, SHUTDOWN-RECEIVED, SHUTDOWN-ACK-SENT.

Note: No "CLOSED" state is illustrated since if a association is "CLOSED" its TCB SHOULD be removed.

Peer Transport Address List: A list of SCTP transport addresses to which the peer is bound. This information is derived from the INIT or INIT ACK chunk and is used to associate an inbound packet with a given association. Normally, this information is hashed or keyed for quick lookup and access of the TCB.

Primary Path: This is the current primary destination transport address of the peer endpoint. It might also specify a source transport address on this endpoint.

Overall Error Count: The overall association error count.

Overall Error Threshold: The threshold for this association that if the Overall Error Count reaches will cause this association to be torn down.

Peer Rwnd: Current calculated value of the peer's rwnd.

Next TSN: The next TSN number to be assigned to a new DATA chunk. This is sent in the INIT or INIT ACK chunk to the peer and incremented each time a DATA chunk is assigned a TSN (normally just prior to transmit or during fragmentation).

Last Rcvd TSN: This is the last TSN received in sequence. This value is set initially by taking the peer's initial TSN, received in the INIT or INIT ACK chunk, and subtracting one from it.

Mapping Array: An array of bits or bytes indicating which out-of-order TSNs have been received (relative to the Last Rcvd TSN). If no gaps exist, i.e., no out-of-order packets have been received, this array will be set to all zero. This structure might be in the form of a circular buffer or bit array.

Ack State: This flag indicates if the next received packet is to be responded to with a SACK chunk. This is initialized to 0. When a packet is received it is incremented. If this value reaches 2 or more, a SACK chunk is sent and the value is reset to 0. Note: This is used only when no DATA chunks are received out of order. When DATA chunks are out of order, SACK chunks are not delayed (see Section 6).

Inbound Streams: An array of structures to track the inbound streams, normally including the next sequence number expected and possibly the stream number.

Outbound Streams: An array of structures to track the outbound streams, normally including the next sequence number to be sent on the stream.

Reasm Queue: A reassembly queue.

Receive Buffer: A buffer to store received user data which has not been delivered to the upper layer.

Local Transport Address List: The list of local IP addresses bound in to this association.

Association Maximum DATA Chunk Size: The smallest Path Maximum DATA Chunk Size of all destination addresses.

14.3. Per Transport Address Data

For each destination transport address in the peer's address list derived from the INIT or INIT ACK chunk, a number of data elements need to be maintained including:

Error Count: The current error count for this destination.

Error Threshold: Current error threshold for this destination, i.e., what value marks the destination down if error count reaches this value.

cwnd: The current congestion window.

ssthresh: The current ssthresh value.

RTO: The current retransmission timeout value.

SRTT: The current smoothed round-trip time.

RTTVAR: The current RTT variation.

partial bytes acked: The tracking method for increase of cwnd when in congestion avoidance mode (see Section 7.2.2).

state: The current state of this destination, i.e., DOWN, UP, ALLOW-HEARTBEAT, NO-HEARTBEAT, etc.

PMTU: The current known PMTU.

PMDCS: The current known PMDCS.

Per Destination Timer: A timer used by each destination.

RTO-Pending: A flag used to track if one of the DATA chunks sent to this address is currently being used to compute an RTT. If this flag is 0, the next DATA chunk sent to this destination is expected to be used to compute an RTT and this flag is expected to be set. Every time the RTT calculation completes (i.e., the DATA chunk is acknowledged), clear this flag.

last-time: The time to which this destination was last sent. This can be used to determine if the sending of a HEARTBEAT chunk is needed.

14.4. General Parameters Needed

Out Queue: A queue of outbound DATA chunks.

In Queue: A queue of inbound DATA chunks.

15. IANA Considerations

This document defines five registries that IANA maintains:

- * through definition of additional chunk types,
- * through definition of additional chunk flags,
- * through definition of additional parameter types,
- * through definition of additional cause codes within ERROR chunks,
or
- * through definition of additional payload protocol identifiers.

IANA is requested to perform the following updates for the above five registries:

- * In the Chunk Types Registry replace in the Reference section the reference to [RFC4960] and [RFC6096] by a reference to this document.

Replace in the Notes section the reference to Section 3.2 of [RFC6096] by a reference to Section 15.2 of this document.

Finally replace each reference to [RFC4960] by a reference to this document for the following chunk types:

- Payload Data (DATA)
- Initiation (INIT)
- Initiation Acknowledgement (INIT ACK)
- Selective Acknowledgement (SACK)
- Heartbeat Request (HEARTBEAT)
- Heartbeat Acknowledgement (HEARTBEAT ACK)
- Abort (ABORT)

- Shutdown (SHUTDOWN)
- Shutdown Acknowledgement (SHUTDOWN ACK)
- Operation Error (ERROR)
- State Cookie (COOKIE ECHO)
- Cookie Acknowledgement (COOKIE ACK)
- Reserved for Explicit Congestion Notification Echo (ECNE)
- Reserved for Congestion Window Reduced (CWR)
- Shutdown Complete (SHUTDOWN COMPLETE)
- Reserved for IETF-defined Chunk Extensions

- * In the Chunk Parameter Types Registry replace in the Reference section the reference to [RFC4960] by a reference to this document.

Replace each reference to [RFC4960] by a reference to this document for the following chunk parameter types:

- Heartbeat Info
- IPv4 Address
- IPv6 Address
- State Cookie
- Unrecognized Parameters
- Cookie Preservative
- Host Name Address
- Supported Address Types

Add a reference to this document for the following chunk parameter type:

- Reserved for ECN Capable (0x8000)

- * In the Chunk Flags Registry replace in the Reference section the reference to [RFC6096] by a reference to this document.

Replace each reference to [RFC4960] by a reference to this document for the following DATA chunk flags:

- E bit
- B bit
- U bit

Replace each reference to [RFC4960] by a reference to this document for the following ABORT chunk flags:

- T bit

Replace each reference to [RFC4960] by a reference to this document for the following SHUTDOWN COMPLETE chunk flags:

- T bit

- * In the Error Cause Codes Registry replace in the Reference section the reference to [RFC6096] by a reference to this document.

Replace each reference to [RFC4960] by a reference to this document for the following cause codes:

- Invalid Stream Identifier
- Missing Mandatory Parameter
- Stale Cookie Error
- Out of Resource
- Unresolvable Address
- Unrecognized Chunk Type
- Invalid Mandatory Parameter
- Unrecognized Parameters
- No User Data
- Cookie Received While Shutting Down
- Restart of an Association with New Addressess

Replace each reference to [RFC4460] by a reference to this document for the following cause codes:

- User Initiated Abort
- Protocol Violation
- * In the SCTP Payload Protocol Identifiers Registry replace in the Reference section the reference to [RFC6096] by a reference to this document.

Replace each reference to [RFC4960] by a reference to this document for the following SCTP payload protocol identifiers:

- Reserved by SCTP

SCTP requires that the IANA Port Numbers registry be opened for SCTP port registrations, Section 15.6 describes how. An IESG-appointed Expert Reviewer supports IANA in evaluating SCTP port allocation requests.

IANA is requested to perform the following update for the Port Number registry. Replace each reference to [RFC4960] by a reference to this document for the following SCTP port numbers:

- * 9 (discard)
- * 20 (ftp-data)
- * 21 (ftp)
- * 22 (ssh)
- * 80 (http)
- * 179 (bgp)
- * 443 (https)

Furthermore, IANA is requested to replace in the HTTP Digest Algorithm Values registry the reference to Appendix B of [RFC4960] to Appendix A of this document.

IANA is also requested to replace in the ONC RPC Netids registry, each of the reference to [RFC4960] by a reference to this document for the following netids:

- * sctp

- * sctp6

IANA is finally requested to replace in the IPFIX Information Elements registry, each of the reference to [RFC4960] by a reference to this document for the following elements with the name:

- * sourceTransportPort
- * destinationTransportPort
- * collectorTransportPort
- * exporterTransportPort
- * postNAPTSourceTransportPort
- * postNAPTDestinationTransportPort

15.1. IETF-Defined Chunk Extension

The assignment of new chunk type codes is done through an IETF Review action, as defined in [RFC8126]. Documentation for a new chunk MUST contain the following information:

- a) A long and short name for the new chunk type.
- b) A detailed description of the structure of the chunk, which MUST conform to the basic structure defined in Section 3.2.
- c) A detailed definition and description of intended use of each field within the chunk, including the chunk flags if any. Defined chunk flags will be used as initial entries in the chunk flags table for the new chunk type.
- d) A detailed procedural description of the use of the new chunk type within the operation of the protocol.

The last chunk type (255) is reserved for future extension if necessary.

For each new chunk type, IANA creates a registration table for the chunk flags of that type. The procedure for registering particular chunk flags is described in Section 15.2.

15.2. IETF Chunk Flags Registration

The assignment of new chunk flags is done through an RFC Required action, as defined in [RFC8126]. Documentation for the chunk flags MUST contain the following information:

- a) A name for the new chunk flag.
- b) A detailed procedural description of the use of the new chunk flag within the operation of the protocol. It MUST be considered that implementations not supporting the flag will send 0 on transmit and just ignore it on receipt.

IANA selects a chunk flags value. This MUST be one of 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, or 0x80, which MUST be unique within the chunk flag values for the specific chunk type.

15.3. IETF-Defined Chunk Parameter Extension

The assignment of new chunk parameter type codes is done through an IETF Review action as defined in [RFC8126]. Documentation of the chunk parameter MUST contain the following information:

- a) Name of the parameter type.
- b) Detailed description of the structure of the parameter field. This structure MUST conform to the general Type-Length-Value format described in Section 3.2.1.
- c) Detailed definition of each component of the parameter value.
- d) Detailed description of the intended use of this parameter type, and an indication of whether and under what circumstances multiple instances of this parameter type can be found within the same chunk.
- e) Each parameter type MUST be unique across all chunks.

15.4. IETF-Defined Additional Error Causes

Additional cause codes can be allocated in the range 11 to 65535 through a Specification Required action as defined in [RFC8126]. Provided documentation MUST include the following information:

- a) Name of the error condition.
- b) Detailed description of the conditions under which an SCTP endpoint issues an ERROR (or ABORT) chunk with this cause code.

- c) Expected action by the SCTP endpoint that receives an ERROR (or ABORT) chunk containing this cause code.
- d) Detailed description of the structure and content of data fields that accompany this cause code.

The initial word (32 bits) of a cause code parameter MUST conform to the format shown in Section 3.3.10, i.e.:

- * first 2 bytes contain the cause code value
- * last 2 bytes contain the length of the cause parameter.

15.5. Payload Protocol Identifiers

The assignment of payload protocol identifier is done using the First Come First Served policy as defined in [RFC8126].

Except for value 0, which is reserved to indicate an unspecified payload protocol identifier in a DATA chunk, an SCTP implementation will not be responsible for standardizing or verifying any payload protocol identifiers; An SCTP implementation simply receives the identifier from the upper layer and carries it with the corresponding payload data.

The upper layer, i.e., the SCTP user, SHOULD standardize any specific protocol identifier with IANA if it is so desired. The use of any specific payload protocol identifier is out of the scope of this specification.

15.6. Port Numbers Registry

SCTP services can use contact port numbers to provide service to unknown callers, as in TCP and UDP. 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].

16. Suggested SCTP Protocol Parameter Values

The following protocol parameters are RECOMMENDED:

RTO.Initial: 1 second

RTO.Min: 1 second

RTO.Max: 60 seconds

Max.Burst: 4

RTO.Alpha: 1/8

RTO.Beta: 1/4

Valid.Cookie.Life: 60 seconds

Association.Max.Retrans: 10 attempts

Path.Max.Retrans: 5 attempts (per destination address)

Max.Init.Retransmits: 8 attempts

HB.interval: 30 seconds

HB.Max.Burst: 1

SACK.Delay: 200 milliseconds

Implementation Note: The SCTP implementation can allow ULP to customize some of these protocol parameters (see Section 11).

'RTO.Min' SHOULD be set as described above in this section.

17. Acknowledgements

An undertaking represented by this updated document is not a small feat and represents the summation of the initial co-authors of [RFC2960]: Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson.

Add to that, the comments from everyone who contributed to [RFC2960]: Mark Allman, R. J. Atkinson, Richard Band, Scott Bradner, Steve Bellovin, Peter Butler, Ram Dantu, R. Ezhirpavai, Mike Fisk, Sally Floyd, Atsushi Fukumoto, Matt Holdrege, Henry Houh, Christian Huitema, Gary Lehecka, Jonathan Lee, David Lehmann, John Loughney, Daniel Luan, Barry Nagelberg, Thomas Narten, Erik Nordmark, Lyndon Ong, Shyamal Prasad, Kelvin Porter, Heinz Prantner, Jarno Rajahalme, Raymond E. Reeves, Renee Revis, Ivan Arias Rodriguez, A. Sankar, Greg Sidebottom, Brian Wyld, La Monte Yarroll, and many others for their invaluable comments.

Then, add the co-authors of [RFC4460]: I. Arias-Rodriguez, K. Poon, and A. Caro.

Then add to these the efforts of all the subsequent seven SCTP interoperability tests and those who commented on [RFC4460] as shown in its acknowledgements: Barry Zuckerman, La Monte Yarroll, Qiaobing Xie, Wang Xiaopeng, Jonathan Wood, Jeff Waskow, Mike Turner, John Townsend, Sabina Torrente, Cliff Thomas, Yuji Suzuki, Manoj Solanki, Sverre Slotte, Keyur Shah, Jan Rovins, Ben Robinson, Renee Revis, Ian Periam, RC Monee, Sanjay Rao, Sujith Radhakrishnan, Heinz Prantner, Biren Patel, Nathalie Mouellic, Mitch Miers, Bernward Meyknecht, Stan McClellan, Oliver Mayor, Tomas Orti Martin, Sandeep Mahajan, David Lehmann, Jonathan Lee, Philippe Langlois, Karl Knutson, Joe Keller, Gareth Keily, Andreas Jungmaier, Janardhan Iyengar, Mutsuya Irie, John Hebert, Kausar Hassan, Fred Hasle, Dan Harrison, Jon Grim, Laurent Glaude, Steven Furniss, Atsushi Fukumoto, Ken Fujita, Steve Dimig, Thomas Curran, Serkan Cil, Melissa Campbell, Peter Butler, Rob Brennan, Harsh Bhondwe, Brian Bidulock, Caitlin Bestler, Jon Berger, Robby Benedyk, Stephen Baucke, Sandeep Balani, and Ronnie Sellar.

A special thanks to Mark Allman, who should actually be a co-author for his work on the max-burst, but managed to wiggle out due to a technicality.

Also, we would like to acknowledge Lyndon Ong and Phil Conrad for their valuable input and many contributions.

Furthermore, you have [RFC4960], and those who have commented upon that including Alfred Hönes and Ronnie Sellars.

Then, add the co-author of [RFC8540]: Maksim Proshin.

And people who have commented on [RFC8540]: Pontus Andersson, Eric W. Biederman, Cedric Bonnet, Spencer Dawkins, Gorrry Fairhurst, Benjamin Kaduk, Mirja Kühlewind, Peter Lei, Gyula Marosi, Lionel Morand, Jeff Morriss, Tom Petch, Kacheong Poon, Julien Pourtet, Irene Rüngeler, Michael Welzl, and Qiaobing Xie.

And finally the people who have provided comments for this document including Gorrry Fairhurst, Martin Duke, Benjamin Kaduk, Tero Kivinen, Eliot Lear, Marcelo Ricardo Leitner, David Mandelberg, John Mattsson, Claudio Porfiri, Maksim Proshin, Ines Robles, Timo Völker, Magnus Westerlund, and Zhouming.

Our thanks cannot be adequately expressed to all of you who have participated in the coding, testing, and updating process of this document. All we can say is, Thank You!

18. Normative References

- [ITU.V42.1994] International Telecommunications Union, "Error-correcting Procedures for DCEs Using Asynchronous-to-Synchronous Conversion", ITU-T Recommendation V.42, 1994.
- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.
- [RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <<https://www.rfc-editor.org/info/rfc1123>>.
- [RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990, <<https://www.rfc-editor.org/info/rfc1191>>.
- [RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, DOI 10.17487/RFC1982, August 1996, <<https://www.rfc-editor.org/info/rfc1982>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <<https://www.rfc-editor.org/info/rfc4291>>.
- [RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <<https://www.rfc-editor.org/info/rfc4303>>.
- [RFC4895] Tuexen, M., Stewart, R., Lei, P., and E. Rescorla, "Authenticated Chunks for the Stream Control Transmission Protocol (SCTP)", RFC 4895, DOI 10.17487/RFC4895, August 2007, <<https://www.rfc-editor.org/info/rfc4895>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/info/rfc5681>>.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and

Transport Protocol Port Number Registry", BCP 165,
RFC 6335, DOI 10.17487/RFC6335, August 2011,
<<https://www.rfc-editor.org/info/rfc6335>>.

- [RFC6083] Tuexen, M., Seggelmann, R., and E. Rescorla, "Datagram Transport Layer Security (DTLS) for Stream Control Transmission Protocol (SCTP)", RFC 6083, DOI 10.17487/RFC6083, January 2011, <<https://www.rfc-editor.org/info/rfc6083>>.
- [RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <<https://www.rfc-editor.org/info/rfc7296>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <<https://www.rfc-editor.org/info/rfc8200>>.
- [RFC8201] McCann, J., Deering, S., Mogul, J., and R. Hinden, Ed., "Path MTU Discovery for IP version 6", STD 87, RFC 8201, DOI 10.17487/RFC8201, July 2017, <<https://www.rfc-editor.org/info/rfc8201>>.
- [RFC8899] Fairhurst, G., Jones, T., Tüxen, M., Rüngeler, I., and T. Völker, "Packetization Layer Path MTU Discovery for Datagram Transports", RFC 8899, DOI 10.17487/RFC8899, September 2020, <<https://www.rfc-editor.org/info/rfc8899>>.

19. Informative References

- [FALL96] Fall, K. and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", SIGCOM 99, V. 26, N. 3, pp 5-21, July 1996.
- [SAVAGE99] Savage, S., Cardwell, N., Wetherall, D., and T. Anderson, "TCP Congestion Control with a Misbehaving Receiver", ACM Computer Communications Review 29(5), October 1999.

- [ALLMAN99] Allman, M. and V. Paxson, "On Estimating End-to-End Network Path Properties", SIGCOM 99, 1999.
- [WILLIAMS93]
Williams, R., "A PAINLESS GUIDE TO CRC ERROR DETECTION ALGORITHMS", SIGCOM 99, August 1993,
<<http://www.geocities.com/SiliconValley/Pines/8659/crc.htm>>.
- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980,
<<https://www.rfc-editor.org/info/rfc768>>.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981,
<<https://www.rfc-editor.org/info/rfc793>>.
- [RFC1858] Ziemba, G., Reed, D., and P. Traina, "Security Considerations for IP Fragment Filtering", RFC 1858, DOI 10.17487/RFC1858, October 1995,
<<https://www.rfc-editor.org/info/rfc1858>>.
- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997,
<<https://www.rfc-editor.org/info/rfc2104>>.
- [RFC2196] Fraser, B., "Site Security Handbook", FYI 8, RFC 2196, DOI 10.17487/RFC2196, September 1997,
<<https://www.rfc-editor.org/info/rfc2196>>.
- [RFC2522] Karn, P. and W. Simpson, "Photuris: Session-Key Management Protocol", RFC 2522, DOI 10.17487/RFC2522, March 1999,
<<https://www.rfc-editor.org/info/rfc2522>>.
- [RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, DOI 10.17487/RFC2960, October 2000,
<<https://www.rfc-editor.org/info/rfc2960>>.
- [RFC3465] Allman, M., "TCP Congestion Control with Appropriate Byte Counting (ABC)", RFC 3465, DOI 10.17487/RFC3465, February 2003, <<https://www.rfc-editor.org/info/rfc3465>>.

- [RFC3873] Pastor, J. and M. Belinchon, "Stream Control Transmission Protocol (SCTP) Management Information Base (MIB)", RFC 3873, DOI 10.17487/RFC3873, September 2004, <<https://www.rfc-editor.org/info/rfc3873>>.
- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <<https://www.rfc-editor.org/info/rfc4086>>.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<https://www.rfc-editor.org/info/rfc4301>>.
- [RFC4460] Stewart, R., Arias-Rodriguez, I., Poon, K., Caro, A., and M. Tuexen, "Stream Control Transmission Protocol (SCTP) Specification Errata and Issues", RFC 4460, DOI 10.17487/RFC4460, April 2006, <<https://www.rfc-editor.org/info/rfc4460>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.
- [RFC6096] Tuexen, M. and R. Stewart, "Stream Control Transmission Protocol (SCTP) Chunk Flags Registration", RFC 6096, DOI 10.17487/RFC6096, January 2011, <<https://www.rfc-editor.org/info/rfc6096>>.
- [RFC6458] Stewart, R., Tuexen, M., Poon, K., Lei, P., and V. Yasevich, "Sockets API Extensions for the Stream Control Transmission Protocol (SCTP)", RFC 6458, DOI 10.17487/RFC6458, December 2011, <<https://www.rfc-editor.org/info/rfc6458>>.
- [RFC6951] Tuexen, M. and R. Stewart, "UDP Encapsulation of Stream Control Transmission Protocol (SCTP) Packets for End-Host to End-Host Communication", RFC 6951, DOI 10.17487/RFC6951, May 2013, <<https://www.rfc-editor.org/info/rfc6951>>.
- [RFC7053] Tuexen, M., Ruengeler, I., and R. Stewart, "SACK-IMMEDIATELY Extension for the Stream Control Transmission Protocol", RFC 7053, DOI 10.17487/RFC7053, November 2013, <<https://www.rfc-editor.org/info/rfc7053>>.

- [RFC8260] Stewart, R., Tuexen, M., Loreto, S., and R. Seggelmann, "Stream Schedulers and User Message Interleaving for the Stream Control Transmission Protocol", RFC 8260, DOI 10.17487/RFC8260, November 2017, <<https://www.rfc-editor.org/info/rfc8260>>.
- [RFC8261] Tuexen, M., Stewart, R., Jesup, R., and S. Loreto, "Datagram Transport Layer Security (DTLS) Encapsulation of SCTP Packets", RFC 8261, DOI 10.17487/RFC8261, November 2017, <<https://www.rfc-editor.org/info/rfc8261>>.
- [RFC8540] Stewart, R., Tuexen, M., and M. Proshin, "Stream Control Transmission Protocol: Errata and Issues in RFC 4960", RFC 8540, DOI 10.17487/RFC8540, February 2019, <<https://www.rfc-editor.org/info/rfc8540>>.

Appendix A. CRC32c Checksum Calculation

We define a 'reflected value' as one that is the opposite of the normal bit order of the machine. The 32-bit CRC (Cyclic Redundancy Check) is calculated as described for CRC32c and uses the polynomial code 0x11EDC6F41 (Castagnoli93) or $x^{32}+x^{28}+x^{27}+x^{26}+x^{25}+x^{23}+x^{22}+x^{20}+x^{19}+x^{18}+x^{14}+x^{13}+x^{11}+x^{10}+x^9+x^8+x^6+x^0$. The CRC is computed using a procedure similar to ETHERNET CRC [ITU.V42.1994], modified to reflect transport-level usage.

CRC computation uses polynomial division. A message bit-string M is transformed to a polynomial, $M(X)$, and the CRC is calculated from $M(X)$ using polynomial arithmetic.

When CRCs are used at the link layer, the polynomial is derived from on-the-wire bit ordering: the first bit 'on the wire' is the high-order coefficient. Since SCTP is a transport-level protocol, it cannot know the actual serial-media bit ordering. Moreover, different links in the path between SCTP endpoints can use different link-level bit orders.

A convention therefore is established for mapping SCTP transport messages to polynomials for purposes of CRC computation. The bit-ordering for mapping SCTP messages to polynomials is that bytes are taken most-significant first, but within each byte, bits are taken least-significant first. The first byte of the message provides the eight highest coefficients. Within each byte, the least-significant SCTP bit gives the most-significant polynomial coefficient within that byte, and the most-significant SCTP bit is the least-significant polynomial coefficient in that byte. (This bit ordering is sometimes called 'mirrored' or 'reflected' [WILLIAMS93].) CRC polynomials are to be transformed back into SCTP transport-level byte values, using a consistent mapping.

The SCTP transport-level CRC value can be calculated as follows:

- * CRC input data are assigned to a byte stream, numbered from 0 to N-1.
- * The transport-level byte stream is mapped to a polynomial value. An N-byte PDU with j bytes numbered 0 to N-1 is considered as coefficients of a polynomial $M(x)$ of order $8*N-1$, with bit 0 of byte j being coefficient $x^{(8*(N-j)-8)}$, and bit 7 of byte j being coefficient $x^{(8*(N-j)-1)}$.
- * The CRC remainder register is initialized with all 1s and the CRC is computed with an algorithm that simultaneously multiplies by x^{32} and divides by the CRC polynomial.
- * The polynomial is multiplied by x^{32} and divided by $G(x)$, the generator polynomial, producing a remainder $R(x)$ of degree less than or equal to 31.
- * The coefficients of $R(x)$ are considered a 32-bit sequence.
- * The bit sequence is complemented. The result is the CRC polynomial.
- * The CRC polynomial is mapped back into SCTP transport-level bytes. The coefficient of x^{31} gives the value of bit 7 of SCTP byte 0, and the coefficient of x^{24} gives the value of bit 0 of byte 0. The coefficient of x^7 gives bit 7 of byte 3, and the coefficient of x^0 gives bit 0 of byte 3. The resulting 4-byte transport-level sequence is the 32-bit SCTP checksum value.

Implementation Note: Standards documents, textbooks, and vendor literature on CRCs often follow an alternative formulation, in which the register used to hold the remainder of the long-division algorithm is initialized to zero rather than all ones, and instead

the first 32 bits of the message are complemented. The long-division algorithm used in our formulation is specified such that the initial multiplication by 2^{32} and the long-division are combined into one simultaneous operation. For such algorithms, and for messages longer than 64 bits, the two specifications are precisely equivalent. That equivalence is the intent of this document.

Implementors of SCTP are warned that both specifications are to be found in the literature, sometimes with no restriction on the long-division algorithm. The choice of formulation in this document is to permit non-SCTP usage, where the same CRC algorithm can be used to protect messages shorter than 64 bits.

There can be a computational advantage in validating the association against the Verification Tag, prior to performing a checksum, as invalid tags will result in the same action as a bad checksum in most cases. The exceptions for this technique would be packets containing INIT chunks and some SHUTDOWN-COMplete chunks, as well as a stale COOKIE ECHO chunks. These special-case exchanges represent small packets and will minimize the effect of the checksum calculation.

The following non-normative sample code is taken from an open-source CRC generator [WILLIAMS93], using the "mirroring" technique and yielding a lookup table for SCTP CRC32c with 256 entries, each 32 bits wide. While neither especially slow nor especially fast, as software table-lookup CRCs go, it has the advantage of working on both big-endian and little-endian CPUs, using the same (host-order) lookup tables, and using only the predefined `ntohl()` and `htonl()` operations. The code is somewhat modified from [WILLIAMS93], to ensure portability between big-endian and little-endian architectures, use fixed sized types to allow portability between 32-bit and 64-bit platforms, and general C code improvements. (Note that if the byte endian-ness of the target architecture is known to be little-endian, the final bit-reversal and byte-reversal steps can be folded into a single operation.)

```
<CODE BEGINS>
/*****
/* Note: The definitions for Ross Williams's table generator */
/* would be TB_WIDTH=4, TB_POLY=0x1EDC6F41, TB_REVER=TRUE. */
/* For Mr. Williams's direct calculation code, use the settings */
/* cm_width=32, cm_poly=0x1EDC6F41, cm_init=0xFFFFFFFF, */
/* cm_refin=TRUE, cm_refot=TRUE, cm_xorot=0x00000000. */
*****/

/* Example of the crc table file */
#ifndef __crc32cr_h__
#define __crc32cr_h__
```

```

#define CRC32C_POLY 0x1EDC6F41UL
#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])

uint32_t crc_c[256] = {
    0x00000000UL, 0xF26B8303UL, 0xE13B70F7UL, 0x1350F3F4UL,
    0xC79A971FUL, 0x35F1141CUL, 0x26A1E7E8UL, 0xD4CA64EBUL,
    0x8AD958CFUL, 0x78B2DBCCUL, 0x6BE22838UL, 0x9989AB3BUL,
    0x4D43CFD0UL, 0xBF284CD3UL, 0xAC78BF27UL, 0x5E133C24UL,
    0x105EC76FUL, 0xE235446CUL, 0xF165B798UL, 0x030E349BUL,
    0xD7C45070UL, 0x25AFD373UL, 0x36FF2087UL, 0xC494A384UL,
    0x9A879FA0UL, 0x68EC1CA3UL, 0x7BBCEF57UL, 0x89D76C54UL,
    0x5D1D08BFUL, 0xAF768BBCUL, 0xBC267848UL, 0x4E4DFB4BUL,
    0x20BD8EDEUL, 0xD2D60DDFUL, 0xC186FE29UL, 0x33ED7D2AUL,
    0xE72719C1UL, 0x154C9AC2UL, 0x061C6936UL, 0xF477EA35UL,
    0xAA64D611UL, 0x580F5512UL, 0x4B5FA6E6UL, 0xB93425E5UL,
    0x6DFE410EUL, 0x9F95C20DUL, 0x8CC531F9UL, 0x7EAE2FAUL,
    0x30E349B1UL, 0xC288CAB2UL, 0xD1D83946UL, 0x23B3BA45UL,
    0xF779DEAEUL, 0x05125DADUL, 0x1642AE59UL, 0xE4292D5AUL,
    0xBA3A117EUL, 0x4851927DUL, 0x5B016189UL, 0xA96AE28AUL,
    0x7DA08661UL, 0x8FCB0562UL, 0x9C9BF696UL, 0x6EF07595UL,
    0x417B1DBCUL, 0xB3109EBFUL, 0xA0406D4BUL, 0x522BEE48UL,
    0x86E18AA3UL, 0x748A09A0UL, 0x67DAFA54UL, 0x95B17957UL,
    0xCBA24573UL, 0x39C9C670UL, 0x2A993584UL, 0xD8F2B687UL,
    0x0C38D26CUL, 0xFE53516FUL, 0xED03A29BUL, 0x1F682198UL,
    0x5125DAD3UL, 0xA34E59D0UL, 0xB01EAA24UL, 0x42752927UL,
    0x96BF4DCCUL, 0x64D4CECFUL, 0x77843D3BUL, 0x85EFBE38UL,
    0xDBFC821CUL, 0x2997011FUL, 0x3AC7F2EBUL, 0xC8AC71E8UL,
    0x1C661503UL, 0xEE0D9600UL, 0xFD5D65F4UL, 0x0F36E6F7UL,
    0x61C69362UL, 0x93AD1061UL, 0x80FDE395UL, 0x72966096UL,
    0xA65C047DUL, 0x5437877EUL, 0x4767748AUL, 0xB50CF789UL,
    0xEB1FCBADUL, 0x197448AEUL, 0x0A24BB5AUL, 0xF84F3859UL,
    0x2C855CB2UL, 0xDEEEDFB1UL, 0xCDBE2C45UL, 0x3FD5AF46UL,
    0x7198540DUL, 0x83F3D70EUL, 0x90A324FAUL, 0x62C8A7F9UL,
    0xB602C312UL, 0x44694011UL, 0x5739B3E5UL, 0xA55230E6UL,
    0xFB410CC2UL, 0x092A8FC1UL, 0x1A7A7C35UL, 0xE811FF36UL,
    0x3CDB9BDDUL, 0xCEB018DEUL, 0xDDE0EB2AUL, 0x2F8B6829UL,
    0x82F63B78UL, 0x709DB87BUL, 0x63CD4B8FUL, 0x91A6C88CUL,
    0x456CAC67UL, 0xB7072F64UL, 0xA457DC90UL, 0x563C5F93UL,
    0x082F63B7UL, 0xFA44E0B4UL, 0xE9141340UL, 0x1B7F9043UL,
    0xCFB5F4A8UL, 0x3DDE77ABUL, 0x2E8E845FUL, 0xDCE5075CUL,
    0x92A8FC17UL, 0x60C37F14UL, 0x73938CE0UL, 0x81F80FE3UL,
    0x55326B08UL, 0xA759E80BUL, 0xB4091BFFUL, 0x466298FCUL,
    0x1871A4D8UL, 0xEA1A27DBUL, 0xF94AD42FUL, 0x0B21572CUL,
    0xDFEB33C7UL, 0x2D80B0C4UL, 0x3ED04330UL, 0xCCBBC033UL,
    0xA24BB5A6UL, 0x502036A5UL, 0x4370C551UL, 0xB11B4652UL,
    0x65D122B9UL, 0x97BAA1BAUL, 0x84EA524EUL, 0x7681D14DUL,
    0x2892ED69UL, 0xDAF96E6AUL, 0xC9A99D9EUL, 0x3BC21E9DUL,
    0xEF087A76UL, 0x1D63F975UL, 0x0E330A81UL, 0xFC588982UL,

```

```

0xB21572C9UL, 0x407EF1CAUL, 0x532E023EUL, 0xA145813DUL,
0x758FE5D6UL, 0x87E466D5UL, 0x94B49521UL, 0x66DF1622UL,
0x38CC2A06UL, 0xCA7A905UL, 0xD9F75AF1UL, 0x2B9CD9F2UL,
0xFF56BD19UL, 0x0D3D3E1AUL, 0x1E6DCDEEUL, 0xEC064EEDUL,
0xC38D26C4UL, 0x31E6A5C7UL, 0x22B65633UL, 0xD0DDD530UL,
0x0417B1DBUL, 0xF67C32D8UL, 0xE52CC12CUL, 0x1747422FUL,
0x49547E0BUL, 0xBB3FFD08UL, 0xA86F0EFCUL, 0x5A048DFFUL,
0x8ECEE914UL, 0x7CA56A17UL, 0x6FF599E3UL, 0x9D9E1AE0UL,
0xD3D3E1ABUL, 0x21B862A8UL, 0x32E8915CUL, 0xC083125FUL,
0x144976B4UL, 0xE622F5B7UL, 0xF5720643UL, 0x07198540UL,
0x590AB964UL, 0xAB613A67UL, 0xB831C993UL, 0x4A5A4A90UL,
0x9E902E7BUL, 0x6CFBAD78UL, 0x7FAB5E8CUL, 0x8DC0DD8FUL,
0xE330A81AUL, 0x115B2B19UL, 0x020BD8EDUL, 0xF0605BEEUL,
0x24AA3F05UL, 0xD6C1BC06UL, 0xC5914FF2UL, 0x37FACCF1UL,
0x69E9F0D5UL, 0x9B8273D6UL, 0x88D28022UL, 0x7AB90321UL,
0xAE7367CAUL, 0x5C18E4C9UL, 0x4F48173DUL, 0xBD23943EUL,
0xF36E6F75UL, 0x0105EC76UL, 0x12551F82UL, 0xE03E9C81UL,
0x34F4F86AUL, 0xC69F7B69UL, 0xD5CF889DUL, 0x27A40B9EUL,
0x79B737BAUL, 0x8BDCB4B9UL, 0x988C474DUL, 0x6AE7C44EUL,
0xBE2DA0A5UL, 0x4C4623A6UL, 0x5F16D052UL, 0xAD7D5351UL,
};

```

```
#endif
```

```
/* 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 |= 1UL << (31 - i);
        b >>= 1;
    }
    return (rw);
}

```

```

static uint32_t
build_crc_table (int index)
{
    int i;
    uint32_t rb;

    rb = reflect_32(index);

    for (i = 0; i < 8; i++) {
        if (rb & 0x80000000UL)
            rb = (rb << 1) ^ (uint32_t)CRC32C_POLY;
        else
            rb <<= 1;
    }
    return (reflect_32(rb));
}

int
main (void)
{
    int i;

    printf("\nGenerating CRC32c table file <%s>.\n",
        OUTPUT_FILE);
    if ((tf = fopen(OUTPUT_FILE, "w")) == NULL) {
        printf("Unable to open %s.\n", OUTPUT_FILE);
        exit (1);
    }
    fprintf(tf, "#ifndef __crc32cr_h__\n");
    fprintf(tf, "#define __crc32cr_h__\n\n");
    fprintf(tf, "#define CRC32C_POLY 0x%08XUL\n",
        (uint32_t)CRC32C_POLY);
    fprintf(tf,
        "#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])\n");
    fprintf(tf, "\nuint32_t crc_c[256] =\n{\n");
    for (i = 0; i < 256; i++) {
        fprintf(tf, "0x%08XUL,", build_crc_table (i));
        if ((i & 3) == 3)
            fprintf(tf, "\n");
        else
            fprintf(tf, " ");
    }
    fprintf(tf, "};\n\n#endif\n");

    if (fclose(tf) != 0)
        printf("Unable to close <%s>.\n", OUTPUT_FILE);
    else
        printf("\nThe CRC32c table has been written to <%s>.\n",

```

```

        OUTPUT_FILE);
    return (0);
}

/* Example of crc insertion */

#include "crc32cr.h"

uint32_t
generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    uint32_t crc32 = 0xffffffffUL;
    uint32_t result;
    uint32_t byte0, byte1, byte2, byte3;

    for (i = 0; i < length; i++) {
        CRC32C(crc32, buffer[i]);
    }

    result = ~crc32;

    /* result now holds the negated polynomial remainder,
     * since the table and algorithm are "reflected" [williams95].
     * That is, result has the same value as if we mapped the message
     * to a polynomial, computed the host-bit-order polynomial
     * remainder, performed final negation, and then did an
     * end-for-end bit-reversal.
     * Note that a 32-bit bit-reversal is identical to four in-place
     * 8-bit bit-reversals followed by an end-for-end byteswap.
     * In other words, the bits of each byte are in the right order,
     * but the bytes have been byteswapped. So, we now do an explicit
     * byteswap. On a little-endian machine, this byteswap and
     * the final ntohl cancel out and could be elided.
     */

    byte0 = result & 0xff;
    byte1 = (result>>8) & 0xff;
    byte2 = (result>>16) & 0xff;
    byte3 = (result>>24) & 0xff;
    crc32 = ((byte0 << 24) |
              (byte1 << 16) |
              (byte2 << 8) |
              byte3);
    return (crc32);
}

int
```

```
insert_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    uint32_t crc32;

    message = (SCTP_message *)buffer;
    message->common_header.checksum = 0UL;
    crc32 = generate_crc32c(buffer,length);
    /* and insert it into the message */
    message->common_header.checksum = htonl(crc32);
    return (1);
}

int
validate_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    unsigned int i;
    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>
```

Authors' Addresses

Randall R. Stewart
Netflix, Inc.
2455 Heritage Green Ave
Davenport, FL 33837
United States

Email: randall@lakerest.net

Michael Tüxen
Münster University of Applied Sciences
Stegerwaldstrasse 39
48565 Steinfurt
Germany

Email: tuexen@fh-muenster.de

Karen E. E. Nielsen
Kamstrup A/S
Industrivej 28
DK-8660 Skanderborg
Denmark

Email: kee@kamstrup.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 25, 2019

R. Stewart
Netflix, Inc.
M. Tuexen
Muenster Univ. of Appl. Sciences
M. Proshin
Ericsson
October 22, 2018

RFC 4960 Errata and Issues
draft-ietf-tsvwg-rfc4960-errata-08.txt

Abstract

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

Status of This Memo

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

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

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

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

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

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

Table of Contents

1. Introduction	3
2. Conventions	4
3. Corrections to RFC 4960	4
3.1. Path Error Counter Threshold Handling	4
3.2. Upper Layer Protocol Shutdown Request Handling	5
3.3. Registration of New Chunk Types	6
3.4. Variable Parameters for INIT Chunks	7
3.5. CRC32c Sample Code on 64-bit Platforms	8
3.6. Endpoint Failure Detection	9
3.7. Data Transmission Rules	10
3.8. T1-Cookie Timer	11
3.9. Miscellaneous Typos	12
3.10. CRC32c Sample Code	19
3.11. partial_bytes_acked after T3-rtx Expiration	20
3.12. Order of Adjustments of partial_bytes_acked and cwnd	21
3.13. HEARTBEAT ACK and the association error counter	22
3.14. Path for Fast Retransmission	23
3.15. Transmittal in Fast Recovery	24
3.16. Initial Value of ssthresh	25
3.17. Automatically Confirmed Addresses	26
3.18. Only One Packet after Retransmission Timeout	27
3.19. INIT ACK Path for INIT in COOKIE-WAIT State	28
3.20. Zero Window Probing and Unreachable Primary Path	29
3.21. Normative Language in Section 10	30
3.22. Increase of partial_bytes_acked in Congestion Avoidance	33
3.23. Inconsistency in Notifications Handling	34
3.24. SACK.Delay Not Listed as a Protocol Parameter	40
3.25. Processing of Chunks in an Incoming SCTP Packet	42
3.26. CWND Increase in Congestion Avoidance Phase	43
3.27. Refresh of cwnd and ssthresh after Idle Period	46
3.28. Window Updates After Receiver Window Opens Up	47
3.29. Path of DATA and Reply Chunks	48
3.30. Outstanding Data, Flightsize and Data In Flight Key Terms	50
3.31. CWND Degradation due to Max.Burst	52
3.32. Reduction of RTO.Initial	53
3.33. Ordering of Bundled SACK and ERROR Chunks	55
3.34. Undefined Parameter Returned by RECEIVE Primitive	56
3.35. DSCP Changes	57

3.36. Inconsistent Handling of ICMPv4 and ICMPv6 Messages . . .	58
3.37. Handling of Soft Errors	60
3.38. Honoring CWND	60
3.39. Zero Window Probing	62
3.40. Updating References Regarding ECN	64
3.41. Host Name Address Parameter Deprecated	66
3.42. Conflicting Text Regarding the Supported Address Types Parameter	70
3.43. Integration of RFC 6096	71
3.44. Integration of RFC 6335	73
3.45. Integration of RFC 7053	75
3.46. CRC32c Code Improvements	79
3.47. Clarification of Gap Ack Blocks in SACK Chunks	89
3.48. Handling of SSN Wrap Arounds	91
3.49. Update RFC 2119 Boilerplate	92
3.50. Missed Text Removal	93
4. IANA Considerations	94
5. Security Considerations	94
6. Acknowledgments	94
7. References	95
7.1. Normative References	95
7.2. Informative References	95
Authors' Addresses	96

1. Introduction

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

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

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

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

Note that when reading this document one must use care to assure that a field or item is not updated further on within the document. Since this document is a historical record of the sequential changes that

have been found necessary at various inter-op events and through discussion on the list, the last delta in the text is the one which should be applied.

2. Conventions

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

3. Corrections to RFC 4960

[NOTE to RFC-Editor:

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

]

3.1. Path Error Counter Threshold Handling

3.1.1. Description of the Problem

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

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

3.1.2. Text Changes to the Document

Old text: (Section 8.3)

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

New text: (Section 8.3)

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

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

3.1.3. Solution Description

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

3.2. Upper Layer Protocol Shutdown Request Handling

3.2.1. Description of the Problem

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

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

3.2.2. Text Changes to the Document

Old text: (Section 9.2)

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

New text: (Section 9.2)

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

This text is in final form, and is not further updated in this document.

3.2.3. Solution Description

The text never intended the SCTP endpoint to ignore SHUTDOWN chunks from its peer. If it did, the endpoints could never gracefully terminate associations in some cases.

3.3. Registration of New Chunk Types

3.3.1. Description of the Problem

Section 14.1 of [RFC4960] should deal with new chunk types, however, the text refers to parameter types.

This issue was reported as an Errata for [RFC4960] with Errata ID 2592.

3.3.2. Text Changes to the Document

Old text: (Section 14.1)

The assignment of new chunk parameter type codes is done through an IETF Consensus action, as defined in [RFC2434]. Documentation of the chunk parameter MUST contain the following information:

New text: (Section 14.1)

The assignment of new chunk type codes is done through an IETF Consensus action, as defined in [RFC8126]. Documentation of the chunk type MUST contain the following information:

This text has been modified by multiple errata. It is further updated in Section 3.43.

3.3.3. Solution Description

Refer to chunk types as intended and change reference to [RFC8126].

3.4. Variable Parameters for INIT Chunks

3.4.1. Description of the Problem

Newlines in wrong places break the layout of the table of variable parameters for the INIT chunk in Section 3.3.2 of [RFC4960].

This issue was reported as an Errata for [RFC4960] with Errata ID 3291 and Errata ID 3804.

3.4.2. Text Changes to the Document

 Old text: (Section 3.3.2)

Variable Parameters	Status	Type Value
IPv4 Address (Note 1)	Optional	5
IPv6 Address (Note 1)	Optional	6
Cookie Preservative	Optional	9
Reserved for ECN Capable (Note 2)	Optional	32768 (0x8000)
Host Name Address (Note 3)	Optional	11
Supported Address Types (Note 4)	Optional	12

 New text: (Section 3.3.2)

Variable Parameters	Status	Type Value
IPv4 Address (Note 1)	Optional	5
IPv6 Address (Note 1)	Optional	6
Cookie Preservative	Optional	9
Reserved for ECN Capable (Note 2)	Optional	32768 (0x8000)
Host Name Address (Note 3)	Optional	11
Supported Address Types (Note 4)	Optional	12

This text is in final form, and is not further updated in this document.

3.4.3. Solution Description

Fix the formatting of the table.

3.5. CRC32c Sample Code on 64-bit Platforms

3.5.1. Description of the Problem

The sample code for computing the CRC32c provided in [RFC4960] assumes that a variable of type unsigned long uses 32 bits. This is not true on some 64-bit platforms (for example the ones using LP64).

This issue was reported as an Errata for [RFC4960] with Errata ID 3423.

3.5.2. Text Changes to the Document

Old text: (Appendix C)

```
unsigned long
generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    unsigned long crc32 = ~0L;
```

New text: (Appendix C)

```
unsigned long
generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    unsigned long crc32 = 0xffffffffL;
```

This text has been modified by multiple errata. It is further updated in Section 3.10 and in Section 3.46.

3.5.3. Solution Description

Use 0xffffffffL instead of ~0L which gives the same value on platforms using 32 bits or 64 bits for variables of type unsigned long.

3.6. Endpoint Failure Detection

3.6.1. Description of the Problem

The handling of the association error counter defined in Section 8.1 of [RFC4960] can result in an association failure even if the path used for data transmission is available, but idle.

This issue was reported as an Errata for [RFC4960] with Errata ID 3788.

3.6.2. Text Changes to the Document

Old text: (Section 8.1)

An endpoint shall keep a counter on the total number of consecutive retransmissions to its peer (this includes retransmissions to all the destination transport addresses of the peer if it is multi-homed), including unacknowledged HEARTBEAT chunks.

New text: (Section 8.1)

An endpoint SHOULD keep a counter on the total number of consecutive retransmissions to its peer (this includes data retransmissions to all the destination transport addresses of the peer if it is multi-homed), including the number of unacknowledged HEARTBEAT chunks observed on the path which is currently used for data transfer. Unacknowledged HEARTBEAT chunks observed on paths different from the path currently used for data transfer SHOULD NOT increment the association error counter, as this could lead to association closure even if the path which is currently used for data transfer is available (but idle).

This text has been modified by multiple errata. It is further updated in Section 3.23.

3.6.3. Solution Description

A more refined handling for the association error counter is defined.

3.7. Data Transmission Rules

3.7.1. Description of the Problem

When integrating the changes to Section 6.1 A) of [RFC2960] as described in Section 2.15.2 of [RFC4460] some text was duplicated and became the final paragraph of Section 6.1 A) of [RFC4960].

This issue was reported as an Errata for [RFC4960] with Errata ID 4071.

3.7.2. Text Changes to the Document

Old text: (Section 6.1 A)

The sender MUST also have an algorithm for sending new DATA chunks to avoid silly window syndrome (SWS) as described in [RFC0813]. The algorithm can be similar to the one described in Section 4.2.3.4 of [RFC1122].

However, regardless of the value of `rwnd` (including if it is 0), the data sender can always have one DATA chunk in flight to the receiver if allowed by `cwnd` (see rule B below). This rule allows the sender to probe for a change in `rwnd` that the sender missed due to the SACK having been lost in transit from the data receiver to the data sender.

New text: (Section 6.1 A)

The sender MUST also have an algorithm for sending new DATA chunks to avoid silly window syndrome (SWS) as described in [RFC1122]. The algorithm can be similar to the one described in Section 4.2.3.4 of [RFC1122].

This text is in final form, and is not further updated in this document.

3.7.3. Solution Description

Last paragraph of Section 6.1 A) removed as intended in Section 2.15.2 of [RFC4460].

3.8. T1-Cookie Timer

3.8.1. Description of the Problem

Figure 4 of [RFC4960] illustrates the SCTP association setup. However, it incorrectly shows that the `T1-init` timer is used in the `COOKIE-ECHOED` state whereas the `T1-cookie` timer should have been used instead.

This issue was reported as an Errata for [RFC4960] with Errata ID 4400.

3.8.2. Text Changes to the Document

 Old text: (Section 5.1.6, Figure 4)

```

COOKIE ECHO [Cookie_Z] -----\
(Start T1-init timer)           \
(Enter COOKIE-ECHOED state)      \---> (build TCB enter ESTABLISHED
                                         state)
                                   /---- COOKIE-ACK
                                   /
  (Cancel T1-init timer, <-----/
    Enter ESTABLISHED state)

```

 New text: (Section 5.1.6, Figure 4)

```

COOKIE ECHO [Cookie_Z] -----\
(Start T1-cookie timer)        \
(Enter COOKIE-ECHOED state)     \---> (build TCB enter ESTABLISHED
                                         state)
                                   /---- COOKIE-ACK
                                   /
  (Cancel T1-cookie timer, <----/
    Enter ESTABLISHED state)

```

This text has been modified by multiple errata. It is further updated in Section 3.9.

3.8.3. Solution Description

Change the figure such that the T1-cookie timer is used instead of the T1-init timer.

3.9. Miscellaneous Typos

3.9.1. Description of the Problem

While processing [RFC4960] some typos were not caught.

One typo was reported as an Errata for [RFC4960] with Errata ID 5003.

3.9.2. Text Changes to the Document

Old text: (Section 1.6)

Transmission Sequence Numbers wrap around when they reach $2^{32} - 1$. That is, the next TSN a DATA chunk MUST use after transmitting $TSN = 2^{32} - 1$ is $TSN = 0$.

New text: (Section 1.6)

Transmission Sequence Numbers wrap around when they reach $2^{32} - 1$. That is, the next TSN a DATA chunk MUST use after transmitting $TSN = 2^{32} - 1$ is $TSN = 0$.

This text is in final form, and is not further updated in this document.

Old text: (Section 3.3.10.9)

No User Data: This error cause is returned to the originator of a DATA chunk if a received DATA chunk has no user data.

New text: (Section 3.3.10.9)

No User Data: This error cause is returned to the originator of a DATA chunk if a received DATA chunk has no user data.

This text is in final form, and is not further updated in this document.

 Old text: (Section 6.7, Figure 9)

```

Endpoint A                                Endpoint Z {App
sends 3 messages; strm 0} DATA [TSN=6,Strm=0,Seq=2] -----
-----> (ack delayed) (Start T3-rtx timer)

DATA [TSN=7,Strm=0,Seq=3] -----> X (lost)

DATA [TSN=8,Strm=0,Seq=4] -----> (gap detected,
                                   immediately send ack)
                                   /----- SACK [TSN Ack=6,Block=1,
                                   /
                                   <-----/ (remove 6 from out-queue,
                                   and mark 7 as "1" missing report)

```

 New text: (Section 6.7, Figure 9)

```

Endpoint A                                Endpoint Z
{App sends 3 messages; strm 0}
DATA [TSN=6,Strm=0,Seq=2] -----> (ack delayed)
(Start T3-rtx timer)

DATA [TSN=7,Strm=0,Seq=3] -----> X (lost)

DATA [TSN=8,Strm=0,Seq=4] -----> (gap detected,
                                   immediately send ack)
                                   /----- SACK [TSN Ack=6,Block=1,
                                   /
                                   <-----/
(remove 6 from out-queue,
 and mark 7 as "1" missing report)

```

This text is in final form, and is not further updated in this document.

Old text: (Section 6.10)

An endpoint bundles chunks by simply including multiple chunks in one outbound SCTP packet. The total size of the resultant IP datagram, including the SCTP packet and IP headers, MUST be less than or equal to the current Path MTU.

New text: (Section 6.10)

An endpoint bundles chunks by simply including multiple chunks in one outbound SCTP packet. The total size of the resultant IP datagram, including the SCTP packet and IP headers, MUST be less than or equal to the current PMTU.

This text is in final form, and is not further updated in this document.

Old text: (Section 10.1 O))

o Receive Unacknowledged Message

Format: RECEIVE_UNACKED(data retrieval id, buffer address, buffer size, [,stream id] [, stream sequence number] [,partial flag] [,payload protocol-id])

New text: (Section 10.1 O))

O) Receive Unacknowledged Message

Format: RECEIVE_UNACKED(data retrieval id, buffer address, buffer size [,stream id] [,stream sequence number] [,partial flag] [,payload protocol-id])

This text is in final form, and is not further updated in this document.

Old text: (Section 10.1 M)

M) Set Protocol Parameters

Format: SETPROTOCOLPARAMETERS(association id,
[,destination transport address,]
protocol parameter list)

New text: (Section 10.1 M)

M) Set Protocol Parameters

Format: SETPROTOCOLPARAMETERS(association id,
[destination transport address,]
protocol parameter list)

This text is in final form, and is not further updated in this document.

Old text: (Appendix C)

ICMP2) An implementation MAY ignore all ICMPv6 messages where the type field is not "Destination Unreachable", "Parameter Problem", or "Packet Too Big".

New text: (Appendix C)

ICMP2) An implementation MAY ignore all ICMPv6 messages where the type field is not "Destination Unreachable", "Parameter Problem", or "Packet Too Big".

This text is in final form, and is not further updated in this document.

Old text: (Appendix C)

ICMP7) If the ICMP message is either a v6 "Packet Too Big" or a v4 "Fragmentation Needed", an implementation MAY process this information as defined for PATH MTU discovery.

New text: (Appendix C)

ICMP7) If the ICMP message is either a v6 "Packet Too Big" or a v4 "Fragmentation Needed", an implementation MAY process this information as defined for PMTU discovery.

This text is in final form, and is not further updated in this document.

Old text: (Section 5.4)

2) For the receiver of the COOKIE ECHO, the only CONFIRMED address is the one to which the INIT-ACK was sent.

New text: (Section 5.4)

2) For the receiver of the COOKIE ECHO, the only CONFIRMED address is the one to which the INIT ACK was sent.

This text is in final form, and is not further updated in this document.

 Old text: (Section 5.1.6, Figure 4)

```

COOKIE ECHO [Cookie_Z] -----\
(Start T1-init timer)           \
(Enter COOKIE-ECHOED state)      \---> (build TCB enter ESTABLISHED
                                         state)
                                   /---- COOKIE-ACK
                                   /
(Cancel T1-init timer, <-----/
  Enter ESTABLISHED state)

```

 New text: (Section 5.1.6, Figure 4)

```

COOKIE ECHO [Cookie_Z] -----\
(Start T1-cookie timer)        \
(Enter COOKIE-ECHOED state)     \---> (build TCB enter ESTABLISHED
                                         state)
                                   /---- COOKIE ACK
                                   /
(Cancel T1-cookie timer, <---/
  Enter ESTABLISHED state)

```

This text has been modified by multiple errata. It includes modifications from Section 3.8. It is in final form, and is not further updated in this document.

 Old text: (Section 5.2.5)

5.2.5. Handle Duplicate COOKIE-ACK.

 New text: (Section 5.2.5)

5.2.5. Handle Duplicate COOKIE ACK.

This text is in final form, and is not further updated in this document.

Old text: (Section 8.3)

By default, an SCTP endpoint SHOULD monitor the reachability of the idle destination transport address(es) of its peer by sending a HEARTBEAT chunk periodically to the destination transport address(es). HEARTBEAT sending MAY begin upon reaching the ESTABLISHED state and is discontinued after sending either SHUTDOWN or SHUTDOWN-ACK. A receiver of a HEARTBEAT MUST respond to a HEARTBEAT with a HEARTBEAT-ACK after entering the COOKIE-ECHOED state (INIT sender) or the ESTABLISHED state (INIT receiver), up until reaching the SHUTDOWN-SENT state (SHUTDOWN sender) or the SHUTDOWN-ACK-SENT state (SHUTDOWN receiver).

New text: (Section 8.3)

By default, an SCTP endpoint SHOULD monitor the reachability of the idle destination transport address(es) of its peer by sending a HEARTBEAT chunk periodically to the destination transport address(es). HEARTBEAT sending MAY begin upon reaching the ESTABLISHED state and is discontinued after sending either SHUTDOWN or SHUTDOWN ACK. A receiver of a HEARTBEAT MUST respond to a HEARTBEAT with a HEARTBEAT ACK after entering the COOKIE-ECHOED state (INIT sender) or the ESTABLISHED state (INIT receiver), up until reaching the SHUTDOWN-SENT state (SHUTDOWN sender) or the SHUTDOWN-ACK-SENT state (SHUTDOWN receiver).

This text is in final form, and is not further updated in this document.

3.9.3. Solution Description

Typos fixed.

3.10. CRC32c Sample Code

3.10.1. Description of the Problem

The CRC32c computation is described in Appendix B of [RFC4960]. However, the corresponding sample code and its explanation appears at the end of Appendix C, which deals with ICMP handling.

3.10.2. Text Changes to the Document

Move all of Appendix C starting with the following sentence to the end of Appendix B.

The following non-normative sample code is taken from an open-source CRC generator [WILLIAMS93], using the "mirroring" technique and yielding a lookup table for SCTP CRC32c with 256 entries, each 32 bits wide.

This text has been modified by multiple errata. It includes modifications from Section 3.5. It is further updated in Section 3.46.

3.10.3. Solution Description

Text moved to the appropriate location.

3.11. partial_bytes_acked after T3-rtx Expiration

3.11.1. Description of the Problem

Section 7.2.3 of [RFC4960] explicitly states that partial_bytes_acked should be reset to 0 after packet loss detection from SACK but the same is missed for T3-rtx timer expiration.

3.11.2. Text Changes to the Document

Old text: (Section 7.2.3)

When the T3-rtx timer expires on an address, SCTP should perform slow start by:

```
ssthresh = max(cwnd/2, 4*MTU)
cwnd = 1*MTU
```

New text: (Section 7.2.3)

When the T3-rtx timer expires on an address, SCTP SHOULD perform slow start by:

```
ssthresh = max(cwnd/2, 4*MTU)
cwnd = 1*MTU
partial_bytes_acked = 0
```

This text is in final form, and is not further updated in this document.

3.11.3. Solution Description

Specify that `partial_bytes_acked` should be reset to 0 after `T3-rtx` timer expiration.

3.12. Order of Adjustments of `partial_bytes_acked` and `cwnd`

3.12.1. Description of the Problem

Section 7.2.2 of [RFC4960] likely implies the wrong order of adjustments applied to `partial_bytes_acked` and `cwnd` in the congestion avoidance phase.

3.12.2. Text Changes to the Document

Old text: (Section 7.2.2)

- o When `partial_bytes_acked` is equal to or greater than `cwnd` and before the arrival of the SACK the sender had `cwnd` or more bytes of data outstanding (i.e., before arrival of the SACK, `flightsize` was greater than or equal to `cwnd`), increase `cwnd` by MTU, and reset `partial_bytes_acked` to `(partial_bytes_acked - cwnd)`.

New text: (Section 7.2.2)

- o When `partial_bytes_acked` is equal to or greater than `cwnd` and before the arrival of the SACK the sender had `cwnd` or more bytes of data outstanding (i.e., before arrival of the SACK, `flightsize` was greater than or equal to `cwnd`), `partial_bytes_acked` is reset to `(partial_bytes_acked - cwnd)`. Next, `cwnd` is increased by `1*MTU`.

This text has been modified by multiple errata. It is further updated in Section 3.26.

3.12.3. Solution Description

The new text defines the exact order of adjustments of `partial_bytes_acked` and `cwnd` in the congestion avoidance phase.

3.13. HEARTBEAT ACK and the association error counter

3.13.1. Description of the Problem

Section 8.1 and Section 8.3 of [RFC4960] prescribe that the receiver of a HEARTBEAT ACK must reset the association overall error counter. In some circumstances, e.g. when a router discards DATA chunks but not HEARTBEAT chunks due to the larger size of the DATA chunk, it might be better to not clear the association error counter on reception of the HEARTBEAT ACK and reset it only on reception of the SACK to avoid stalling the association.

3.13.2. Text Changes to the Document

Old text: (Section 8.1)

The counter shall be reset each time a DATA chunk sent to that peer endpoint is acknowledged (by the reception of a SACK) or a HEARTBEAT ACK is received from the peer endpoint.

New text: (Section 8.1)

The counter MUST be reset each time a DATA chunk sent to that peer endpoint is acknowledged (by the reception of a SACK). When a HEARTBEAT ACK is received from the peer endpoint, the counter SHOULD also be reset. The receiver of the HEARTBEAT ACK MAY choose not to clear the counter if there is outstanding data on the association. This allows for handling the possible difference in reachability based on DATA chunks and HEARTBEAT chunks.

This text is in final form, and is not further updated in this document.

Old text: (Section 8.3)

Upon the receipt of the HEARTBEAT ACK, the sender of the HEARTBEAT should clear the error counter of the destination transport address to which the HEARTBEAT was sent, and mark the destination transport address as active if it is not so marked. The endpoint may optionally report to the upper layer when an inactive destination address is marked as active due to the reception of the latest HEARTBEAT ACK. The receiver of the HEARTBEAT ACK must also clear the association overall error count as well (as defined in Section 8.1).

New text: (Section 8.3)

Upon the receipt of the HEARTBEAT ACK, the sender of the HEARTBEAT MUST clear the error counter of the destination transport address to which the HEARTBEAT was sent, and mark the destination transport address as active if it is not so marked. The endpoint MAY optionally report to the upper layer when an inactive destination address is marked as active due to the reception of the latest HEARTBEAT ACK. The receiver of the HEARTBEAT ACK SHOULD also clear the association overall error counter (as defined in Section 8.1).

This text has been modified by multiple errata. It is further updated in Section 3.23.

3.13.3. Solution Description

The new text provides a possibility to not reset the association overall error counter when a HEARTBEAT ACK is received if there are valid reasons for it.

3.14. Path for Fast Retransmission

3.14.1. Description of the Problem

[RFC4960] clearly describes where to retransmit data that is timed out when the peer is multi-homed but the same is not stated for fast retransmissions.

3.14.2. Text Changes to the Document

Old text: (Section 6.4)

Furthermore, when its peer is multi-homed, an endpoint SHOULD try to retransmit a chunk that timed out to an active destination transport address that is different from the last destination address to which the DATA chunk was sent.

New text: (Section 6.4)

Furthermore, when its peer is multi-homed, an endpoint SHOULD try to retransmit a chunk that timed out to an active destination transport address that is different from the last destination address to which the DATA chunk was sent.

When its peer is multi-homed, an endpoint SHOULD send fast retransmissions to the same destination transport address where the original data was sent to. If the primary path has been changed and the original data was sent to the old primary path before the fast retransmit, the implementation MAY send it to the new primary path.

This text is in final form, and is not further updated in this document.

3.14.3. Solution Description

The new text clarifies where to send fast retransmissions.

3.15. Transmittal in Fast Recovery

3.15.1. Description of the Problem

The Fast Retransmit on Gap Reports algorithm intends that only the very first packet may be sent regardless of cwnd in the Fast Recovery phase but rule 3) of [RFC4960], Section 7.2.4, misses this clarification.

3.15.2. Text Changes to the Document

Old text: (Section 7.2.4)

- 3) Determine how many of the earliest (i.e., lowest TSN) DATA chunks marked for retransmission will fit into a single packet, subject to constraint of the path MTU of the destination transport address to which the packet is being sent. Call this value K. Retransmit those K DATA chunks in a single packet. When a Fast Retransmit is being performed, the sender SHOULD ignore the value of cwnd and SHOULD NOT delay retransmission for this single packet.

New text: (Section 7.2.4)

- 3) If not in Fast Recovery, determine how many of the earliest (i.e., lowest TSN) DATA chunks marked for retransmission will fit into a single packet, subject to constraint of the PMTU of the destination transport address to which the packet is being sent. Call this value K. Retransmit those K DATA chunks in a single packet. When a Fast Retransmit is being performed, the sender SHOULD ignore the value of cwnd and SHOULD NOT delay retransmission for this single packet.

This text is in final form, and is not further updated in this document.

3.15.3. Solution Description

The new text explicitly specifies to send only the first packet in the Fast Recovery phase disregarding cwnd limitations.

3.16. Initial Value of ssthresh

3.16.1. Description of the Problem

The initial value of ssthresh should be set arbitrarily high. Using the advertised receiver window of the peer is inappropriate if the peer increases its window after the handshake. Furthermore, use a higher requirements level, since not following the advice may result in performance problems.

3.16.2. Text Changes to the Document

Old text: (Section 7.2.1)

- o The initial value of ssthresh MAY be arbitrarily high (for example, implementations MAY use the size of the receiver advertised window).

New text: (Section 7.2.1)

- o The initial value of ssthresh SHOULD be arbitrarily high (e.g., the size of the largest possible advertised window).

This text is in final form, and is not further updated in this document.

3.16.3. Solution Description

Use the same value as suggested in [RFC5681], Section 3.1, as an appropriate initial value. Furthermore, use the same requirements level.

3.17. Automatically Confirmed Addresses

3.17.1. Description of the Problem

The Path Verification procedure of [RFC4960] prescribes that any address passed to the sender of the INIT by its upper layer is automatically CONFIRMED. This, however, is unclear if only addresses in the request to initiate association establishment are considered or any addresses provided by the upper layer in any requests (e.g. in 'Set Primary').

3.17.2. Text Changes to the Document

Old text: (Section 5.4)

- 1) Any address passed to the sender of the INIT by its upper layer is automatically considered to be CONFIRMED.

New text: (Section 5.4)

- 1) Any addresses passed to the sender of the INIT by its upper layer in the request to initialize an association are automatically considered to be CONFIRMED.

This text is in final form, and is not further updated in this document.

3.17.3. Solution Description

The new text clarifies that only addresses provided by the upper layer in the request to initialize an association are automatically confirmed.

3.18. Only One Packet after Retransmission Timeout

3.18.1. Description of the Problem

[RFC4960] is not completely clear when it describes data transmission after T3-rtx timer expiration. Section 7.2.1 does not specify how many packets are allowed to be sent after T3-rtx timer expiration if more than one packet fit into cwnd. At the same time, Section 7.2.3 has the text without normative language saying that SCTP should ensure that no more than one packet will be in flight after T3-rtx timer expiration until successful acknowledgment. It makes the text inconsistent.

3.18.2. Text Changes to the Document

Old text: (Section 7.2.1)

- o The initial cwnd after a retransmission timeout MUST be no more than 1*MTU.

New text: (Section 7.2.1)

- o The initial cwnd after a retransmission timeout MUST be no more than 1*MTU and only one packet is allowed to be in flight until successful acknowledgement.

This text is in final form, and is not further updated in this document.

3.18.3. Solution Description

The new text clearly specifies that only one packet is allowed to be sent after T3-rtx timer expiration until successful acknowledgement.

3.19. INIT ACK Path for INIT in COOKIE-WAIT State

3.19.1. Description of the Problem

In case of an INIT received in the COOKIE-WAIT state [RFC4960] prescribes to send an INIT ACK to the same destination address to which the original INIT has been sent. This text does not address the possibility of the upper layer to provide multiple remote IP addresses while requesting the association establishment. If the upper layer has provided multiple IP addresses and only a subset of these addresses are supported by the peer then the destination address of the original INIT may be absent in the incoming INIT and sending INIT ACK to that address is useless.

3.19.2. Text Changes to the Document

Old text: (Section 5.2.1)

Upon receipt of an INIT in the COOKIE-WAIT state, an endpoint MUST respond with an INIT ACK using the same parameters it sent in its original INIT chunk (including its Initiate Tag, unchanged). When responding, the endpoint MUST send the INIT ACK back to the same address that the original INIT (sent by this endpoint) was sent.

New text: (Section 5.2.1)

Upon receipt of an INIT in the COOKIE-WAIT state, an endpoint MUST respond with an INIT ACK using the same parameters it sent in its original INIT chunk (including its Initiate Tag, unchanged). When responding, the following rules MUST be applied:

- 1) The INIT ACK MUST only be sent to an address passed by the upper layer in the request to initialize the association.
- 2) The INIT ACK MUST only be sent to an address reported in the incoming INIT.
- 3) The INIT ACK SHOULD be sent to the source address of the received INIT.

This text is in final form, and is not further updated in this document.

3.19.3. Solution Description

The new text requires sending INIT ACK to a destination address that is passed by the upper layer and reported in the incoming INIT. If the source address of the INIT meets these conditions, sending the INIT ACK to the source address of the INIT is the preferred behavior.

3.20. Zero Window Probing and Unreachable Primary Path

3.20.1. Description of the Problem

Section 6.1 of [RFC4960] states that when sending zero window probes, SCTP should neither increment the association counter nor increment the destination address error counter if it continues to receive new packets from the peer. However, the reception of new packets from the peer does not guarantee the peer's reachability and, if the destination address becomes unreachable during zero window probing,

SCTP cannot get an updated `rwnd` until it switches the destination address for probes.

3.20.2. Text Changes to the Document

Old text: (Section 6.1)

If the sender continues to receive new packets from the receiver while doing zero window probing, the unacknowledged window probes should not increment the error counter for the association or any destination transport address. This is because the receiver MAY keep its window closed for an indefinite time. Refer to Section 6.2 on the receiver behavior when it advertises a zero window.

New text: (Section 6.1)

If the sender continues to receive SACKs from the peer while doing zero window probing, the unacknowledged window probes SHOULD NOT increment the error counter for the association or any destination transport address. This is because the receiver could keep its window closed for an indefinite time. Section 6.2 describes the receiver behavior when it advertises a zero window.

This text is in final form, and is not further updated in this document.

3.20.3. Solution Description

The new text clarifies that if the receiver continues to send SACKs, the sender of probes should not increment the error counter of the association and the destination address even if the SACKs do not acknowledge the probes.

3.21. Normative Language in Section 10

3.21.1. Description of the Problem

Section 10 of [RFC4960] is informative and, therefore, normative language such as MUST and MAY cannot be used there. However, there are several places in Section 10 where MUST and MAY are used.

3.21.2. Text Changes to the Document

Old text: (Section 10.1 E))

- o no-bundle flag - instructs SCTP not to bundle this user data with other outbound DATA chunks. SCTP MAY still bundle even when this flag is present, when faced with network congestion.

New text: (Section 10.1 E))

- o no-bundle flag - instructs SCTP not to bundle this user data with other outbound DATA chunks. SCTP may still bundle even when this flag is present, when faced with network congestion.

This text is in final form, and is not further updated in this document.

Old text: (Section 10.1 G))

- o Stream Sequence Number - the Stream Sequence Number assigned by the sending SCTP peer.
- o partial flag - if this returned flag is set to 1, then this Receive contains a partial delivery of the whole message. When this flag is set, the stream id and Stream Sequence Number MUST accompany this receive. When this flag is set to 0, it indicates that no more deliveries will be received for this Stream Sequence Number.

New text: (Section 10.1 G))

- o stream sequence number - the Stream Sequence Number assigned by the sending SCTP peer.
- o partial flag - if this returned flag is set to 1, then this primitive contains a partial delivery of the whole message. When this flag is set, the stream id and stream sequence number must accompany this primitive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.

This text is in final form, and is not further updated in this document.

Old text: (Section 10.1 N)

- o Stream Sequence Number - this value is returned indicating the Stream Sequence Number that was associated with the message.
- o partial flag - if this returned flag is set to 1, then this message is a partial delivery of the whole message. When this flag is set, the stream id and Stream Sequence Number MUST accompany this receive. When this flag is set to 0, it indicates that no more deliveries will be received for this Stream Sequence Number.

New text: (Section 10.1 N)

- o stream sequence number - this value is returned indicating the Stream Sequence Number that was associated with the message.
- o partial flag - if this returned flag is set to 1, then this message is a partial delivery of the whole message. When this flag is set, the stream id and stream sequence number must accompany this primitive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.

This text is in final form, and is not further updated in this document.

Old text: (Section 10.1 O)

- o Stream Sequence Number - this value is returned indicating the Stream Sequence Number that was associated with the message.
- o partial flag - if this returned flag is set to 1, then this message is a partial delivery of the whole message. When this flag is set, the stream id and Stream Sequence Number MUST accompany this receive. When this flag is set to 0, it indicates that no more deliveries will be received for this Stream Sequence Number.

New text: (Section 10.1 O)

- o stream sequence number - this value is returned indicating the Stream Sequence Number that was associated with the message.
- o partial flag - if this returned flag is set to 1, then this message is a partial delivery of the whole message. When this flag is set, the stream id and stream sequence number must accompany this primitive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.

This text is in final form, and is not further updated in this document.

3.21.3. Solution Description

The normative language is removed from Section 10. In addition, the consistency of the text has been improved.

3.22. Increase of partial_bytes_acked in Congestion Avoidance

3.22.1. Description of the Problem

Two issues have been discovered with the partial_bytes_acked handling described in Section 7.2.2 of [RFC4960]:

- o If the Cumulative TSN Ack Point is not advanced but the SACK chunk acknowledges new TSNs in the Gap Ack Blocks, these newly acknowledged TSNs are not considered for partial_bytes_acked although these TSNs were successfully received by the peer.

- o Duplicate TSNs are not considered in `partial_bytes_acked` although they confirm that the DATA chunks were successfully received by the peer.

3.22.2. Text Changes to the Document

Old text: (Section 7.2.2)

- o Whenever `cwnd` is greater than `ssthresh`, upon each SACK arrival that advances the Cumulative TSN Ack Point, increase `partial_bytes_acked` by the total number of bytes of all new chunks acknowledged in that SACK including chunks acknowledged by the new Cumulative TSN Ack and by Gap Ack Blocks.

New text: (Section 7.2.2)

- o Whenever `cwnd` is greater than `ssthresh`, upon each SACK arrival, increase `partial_bytes_acked` by the total number of bytes of all new chunks acknowledged in that SACK including chunks acknowledged by the new Cumulative TSN Ack, by Gap Ack Blocks and by the number of bytes of duplicated chunks reported in Duplicate TSNs.

This text has been modified by multiple errata. It is further updated in Section 3.26.

3.22.3. Solution Description

Now `partial_bytes_acked` is increased by TSNs reported as duplicated as well as TSNs newly acknowledged in Gap Ack Blocks even if the Cumulative TSN Ack Point is not advanced.

3.23. Inconsistency in Notifications Handling

3.23.1. Description of the Problem

[RFC4960] uses inconsistent normative and non-normative language when describing rules for sending notifications to the upper layer. E.g. Section 8.2 of [RFC4960] says that when a destination address becomes inactive due to an unacknowledged DATA chunk or HEARTBEAT chunk, SCTP SHOULD send a notification to the upper layer while Section 8.3 of [RFC4960] says that when a destination address becomes inactive due to an unacknowledged HEARTBEAT chunk, SCTP may send a notification to the upper layer.

This makes the text inconsistent.

3.23.2. Text Changes to the Document

Old text: (Section 8.1)

An endpoint shall keep a counter on the total number of consecutive retransmissions to its peer (this includes retransmissions to all the destination transport addresses of the peer if it is multi-homed), including unacknowledged HEARTBEAT chunks.

New text: (Section 8.1)

An endpoint SHOULD keep a counter on the total number of consecutive retransmissions to its peer (this includes data retransmissions to all the destination transport addresses of the peer if it is multi-homed), including the number of unacknowledged HEARTBEAT chunks observed on the path which currently is used for data transfer. Unacknowledged HEARTBEAT chunks observed on paths different from the path currently used for data transfer SHOULD NOT increment the association error counter, as this could lead to association closure even if the path which currently is used for data transfer is available (but idle). If the value of this counter exceeds the limit indicated in the protocol parameter 'Association.Max.Retrans', the endpoint SHOULD consider the peer endpoint unreachable and SHALL stop transmitting any more data to it (and thus the association enters the CLOSED state). In addition, the endpoint SHOULD report the failure to the upper layer and optionally report back all outstanding user data remaining in its outbound queue. The association is automatically closed when the peer endpoint becomes unreachable.

This text has been modified by multiple errata. It includes modifications from Section 3.6. It is in final form, and is not further updated in this document.

Old text: (Section 8.2)

When an outstanding TSN is acknowledged or a HEARTBEAT sent to that address is acknowledged with a HEARTBEAT ACK, the endpoint shall clear the error counter of the destination transport address to which the DATA chunk was last sent (or HEARTBEAT was sent). When the peer endpoint is multi-homed and the last chunk sent to it was a retransmission to an alternate address, there exists an ambiguity as to whether or not the acknowledgement should be credited to the address of the last chunk sent. However, this ambiguity does not seem to bear any significant consequence to SCTP behavior. If this ambiguity is undesirable, the transmitter may choose not to clear the error counter if the last chunk sent was a retransmission.

New text: (Section 8.2)

When an outstanding TSN is acknowledged or a HEARTBEAT sent to that address is acknowledged with a HEARTBEAT ACK, the endpoint SHOULD clear the error counter of the destination transport address to which the DATA chunk was last sent (or HEARTBEAT was sent), and SHOULD also report to the upper layer when an inactive destination address is marked as active. When the peer endpoint is multi-homed and the last chunk sent to it was a retransmission to an alternate address, there exists an ambiguity as to whether or not the acknowledgement could be credited to the address of the last chunk sent. However, this ambiguity does not seem to bear any significant consequence to SCTP behavior. If this ambiguity is undesirable, the transmitter MAY choose not to clear the error counter if the last chunk sent was a retransmission.

This text is in final form, and is not further updated in this document.

Old text: (Section 8.3)

When the value of this counter reaches the protocol parameter 'Path.Max.Retrans', the endpoint should mark the corresponding destination address as inactive if it is not so marked, and may also optionally report to the upper layer the change of reachability of this destination address. After this, the endpoint should continue HEARTBEAT on this destination address but should stop increasing the counter.

New text: (Section 8.3)

When the value of this counter exceeds the protocol parameter 'Path.Max.Retrans', the endpoint SHOULD mark the corresponding destination address as inactive if it is not so marked, and SHOULD also report to the upper layer the change of reachability of this destination address. After this, the endpoint SHOULD continue HEARTBEAT on this destination address but SHOULD stop increasing the counter.

This text has been modified by multiple errata. It includes modifications from Section 3.1. It is in final form, and is not further updated in this document.

Old text: (Section 8.3)

Upon the receipt of the HEARTBEAT ACK, the sender of the HEARTBEAT should clear the error counter of the destination transport address to which the HEARTBEAT was sent, and mark the destination transport address as active if it is not so marked. The endpoint may optionally report to the upper layer when an inactive destination address is marked as active due to the reception of the latest HEARTBEAT ACK. The receiver of the HEARTBEAT ACK must also clear the association overall error count as well (as defined in Section 8.1).

New text: (Section 8.3)

Upon the receipt of the HEARTBEAT ACK, the sender of the HEARTBEAT SHOULD clear the error counter of the destination transport address to which the HEARTBEAT was sent, and mark the destination transport address as active if it is not so marked. The endpoint SHOULD report to the upper layer when an inactive destination address is marked as active due to the reception of the latest HEARTBEAT ACK. The receiver of the HEARTBEAT ACK SHOULD also clear the association overall error counter (as defined in Section 8.1).

This text has been modified by multiple errata. It includes modifications from Section 3.13. It is in final form, and is not further updated in this document.

Old text: (Section 9.2)

An endpoint should limit the number of retransmissions of the SHUTDOWN chunk to the protocol parameter 'Association.Max.Retrans'. If this threshold is exceeded, the endpoint should destroy the TCB and MUST report the peer endpoint unreachable to the upper layer (and thus the association enters the CLOSED state).

New text: (Section 9.2)

An endpoint SHOULD limit the number of retransmissions of the SHUTDOWN chunk to the protocol parameter 'Association.Max.Retrans'. If this threshold is exceeded, the endpoint SHOULD destroy the TCB and SHOULD report the peer endpoint unreachable to the upper layer (and thus the association enters the CLOSED state).

This text is in final form, and is not further updated in this document.

Old text: (Section 9.2)

The sender of the SHUTDOWN ACK should limit the number of retransmissions of the SHUTDOWN ACK chunk to the protocol parameter 'Association.Max.Retrans'. If this threshold is exceeded, the endpoint should destroy the TCB and may report the peer endpoint unreachable to the upper layer (and thus the association enters the CLOSED state).

New text: (Section 9.2)

The sender of the SHUTDOWN ACK SHOULD limit the number of retransmissions of the SHUTDOWN ACK chunk to the protocol parameter 'Association.Max.Retrans'. If this threshold is exceeded, the endpoint SHOULD destroy the TCB and SHOULD report the peer endpoint unreachable to the upper layer (and thus the association enters the CLOSED state).

This text is in final form, and is not further updated in this document.

3.23.3. Solution Description

The inconsistencies are removed by using consistently SHOULD.

3.24. SACK.Delay Not Listed as a Protocol Parameter

3.24.1. Description of the Problem

SCTP as specified in [RFC4960] supports delaying SACKs. The timer value for this is a parameter and Section 6.2 of [RFC4960] specifies a default and maximum value for it. However, defining a name for this parameter and listing it in the table of protocol parameters in Section 15 of [RFC4960] is missing.

This issue was reported as an Errata for [RFC4960] with Errata ID 4656.

3.24.2. Text Changes to the Document

Old text: (Section 6.2)

An implementation MUST NOT allow the maximum delay to be configured to be more than 500 ms. In other words, an implementation MAY lower this value below 500 ms but MUST NOT raise it above 500 ms.

New text: (Section 6.2)

An implementation MUST NOT allow the maximum delay (protocol parameter 'SACK.Delay') to be configured to be more than 500 ms. In other words, an implementation MAY lower the value of SACK.Delay below 500 ms but MUST NOT raise it above 500 ms.

This text is in final form, and is not further updated in this document.

Old text: (Section 15)

The following protocol parameters are RECOMMENDED:

RTO.Initial - 3 seconds
RTO.Min - 1 second
RTO.Max - 60 seconds
Max.Burst - 4
RTO.Alpha - 1/8
RTO.Beta - 1/4
Valid.Cookie.Life - 60 seconds
Association.Max.Retrans - 10 attempts
Path.Max.Retrans - 5 attempts (per destination address)
Max.Init.Retransmits - 8 attempts
HB.interval - 30 seconds
HB.Max.Burst - 1

New text: (Section 15)

The following protocol parameters are RECOMMENDED:

RTO.Initial - 3 seconds
RTO.Min - 1 second
RTO.Max - 60 seconds
Max.Burst - 4
RTO.Alpha - 1/8
RTO.Beta - 1/4
Valid.Cookie.Life - 60 seconds
Association.Max.Retrans - 10 attempts
Path.Max.Retrans - 5 attempts (per destination address)
Max.Init.Retransmits - 8 attempts
HB.interval - 30 seconds
HB.Max.Burst - 1
SACK.Delay - 200 milliseconds

This text has been modified by multiple errata. It is further updated in Section 3.32.

3.24.3. Solution Description

The parameter was given a name and added to the list of protocol parameters.

3.25. Processing of Chunks in an Incoming SCTP Packet

3.25.1. Description of the Problem

There are a few places in [RFC4960] where the receiver of a packet must discard it while processing the chunks of the packet. It is unclear whether the receiver has to rollback state changes already performed while processing the packet or not.

The intention of [RFC4960] is to process an incoming packet chunk by chunk and not to perform any prescreening of chunks in the received packet. Thus, by discarding one chunk the receiver also causes discarding of all further chunks.

3.25.2. Text Changes to the Document

Old text: (Section 3.2)

- 00 - Stop processing this SCTP packet and discard it, do not process any further chunks within it.
- 01 - Stop processing this SCTP packet and discard it, do not process any further chunks within it, and report the unrecognized chunk in an 'Unrecognized Chunk Type'.

New text: (Section 3.2)

- 00 - Stop processing this SCTP packet, discard the unrecognized chunk and all further chunks.
- 01 - Stop processing this SCTP packet, discard the unrecognized chunk and all further chunks, and report the unrecognized chunk in an 'Unrecognized Chunk Type'.

This text is in final form, and is not further updated in this document.

Old text: (Section 11.3)

It is helpful for some firewalls if they can inspect just the first fragment of a fragmented SCTP packet and unambiguously determine whether it corresponds to an INIT chunk (for further information, please refer to [RFC1858]). Accordingly, we stress the requirements, stated in Section 3.1, that (1) an INIT chunk MUST NOT be bundled with any other chunk in a packet, and (2) a packet containing an INIT chunk MUST have a zero Verification Tag. Furthermore, we require that the receiver of an INIT chunk MUST enforce these rules by silently discarding an arriving packet with an INIT chunk that is bundled with other chunks or has a non-zero verification tag and contains an INIT-chunk.

New text: (Section 11.3)

It is helpful for some firewalls if they can inspect just the first fragment of a fragmented SCTP packet and unambiguously determine whether it corresponds to an INIT chunk (for further information, please refer to [RFC1858]). Accordingly, we stress the requirements, stated in Section 3.1, that (1) an INIT chunk MUST NOT be bundled with any other chunk in a packet, and (2) a packet containing an INIT chunk MUST have a zero Verification Tag. The receiver of an INIT chunk MUST silently discard the INIT chunk and all further chunks if the INIT chunk is bundled with other chunks or the packet has a non-zero verification tag.

This text is in final form, and is not further updated in this document.

3.25.3. Solution Description

The new text makes it clear that chunks can be processed from the beginning to the end and no rollback or pre-screening is required.

3.26. CWND Increase in Congestion Avoidance Phase

3.26.1. Description of the Problem

[RFC4960] in Section 7.2.2 prescribes to increase cwnd by 1*MTU per RTT if the sender has cwnd or more bytes of data outstanding to the corresponding address in the Congestion Avoidance phase. However, this is described without normative language. Moreover, Section 7.2.2 includes an algorithm how an implementation can achieve

this but this algorithm is underspecified and actually allows increasing cwnd by more than 1*MTU per RTT.

3.26.2. Text Changes to the Document

Old text: (Section 7.2.2)

When cwnd is greater than ssthresh, cwnd should be incremented by 1*MTU per RTT if the sender has cwnd or more bytes of data outstanding for the corresponding transport address.

New text: (Section 7.2.2)

When cwnd is greater than ssthresh, cwnd SHOULD be incremented by 1*MTU per RTT if the sender has cwnd or more bytes of data outstanding for the corresponding transport address. The basic guidelines for incrementing cwnd during congestion avoidance are:

- o Sctp MAY increment cwnd by 1*MTU.
- o Sctp SHOULD increment cwnd by one 1*MTU once per RTT when the sender has cwnd or more bytes of data outstanding for the corresponding transport address.
- o Sctp MUST NOT increment cwnd by more than 1*MTU per RTT.

This text is in final form, and is not further updated in this document.

Old text: (Section 7.2.2)

- o Whenever cwnd is greater than ssthresh, upon each SACK arrival that advances the Cumulative TSN Ack Point, increase partial_bytes_acked by the total number of bytes of all new chunks acknowledged in that SACK including chunks acknowledged by the new Cumulative TSN Ack and by Gap Ack Blocks.
- o When partial_bytes_acked is equal to or greater than cwnd and before the arrival of the SACK the sender had cwnd or more bytes of data outstanding (i.e., before arrival of the SACK, flightsize was greater than or equal to cwnd), increase cwnd by MTU, and reset partial_bytes_acked to (partial_bytes_acked - cwnd).

New text: (Section 7.2.2)

- o Whenever cwnd is greater than ssthresh, upon each SACK arrival, increase partial_bytes_acked by the total number of bytes of all new chunks acknowledged in that SACK including chunks acknowledged by the new Cumulative TSN Ack, by Gap Ack Blocks and by the number of bytes of duplicated chunks reported in Duplicate TSNs.
- o When partial_bytes_acked is greater than cwnd and before the arrival of the SACK the sender had less than cwnd bytes of data outstanding (i.e., before arrival of the SACK, flightsize was less than cwnd), reset partial_bytes_acked to cwnd.
- o When partial_bytes_acked is equal to or greater than cwnd and before the arrival of the SACK the sender had cwnd or more bytes of data outstanding (i.e., before arrival of the SACK, flightsize was greater than or equal to cwnd), partial_bytes_acked is reset to (partial_bytes_acked - cwnd). Next, cwnd is increased by 1*MTU.

This text has been modified by multiple errata. It includes modifications from Section 3.12 and Section 3.22. It is in final form, and is not further updated in this document.

3.26.3. Solution Description

The basic guidelines for incrementing cwnd during the congestion avoidance phase are added into Section 7.2.2. The guidelines include the normative language and are aligned with [RFC5681].

The algorithm from Section 7.2.2 is improved to not allow increasing cwnd by more than 1*MTU per RTT.

3.27. Refresh of cwnd and ssthresh after Idle Period

3.27.1. Description of the Problem

[RFC4960] prescribes to adjust cwnd per RTO if the endpoint does not transmit data on a given transport address. In addition to that, it prescribes to set cwnd to the initial value after a sufficiently long idle period. The latter is excessive. Moreover, it is unclear what is a sufficiently long idle period.

[RFC4960] doesn't specify the handling of ssthresh in the idle case. If ssthresh is reduced due to a packet loss, ssthresh is never recovered. So traffic can end up in Congestion Avoidance all the time, resulting in a low sending rate and bad performance. The problem is even more serious for SCTP because in a multi-homed SCTP association traffic that switches back to the previously failed primary path will also lead to the situation where traffic ends up in Congestion Avoidance.

3.27.2. Text Changes to the Document

Old text: (Section 7.2.1)

- o The initial cwnd before DATA transmission or after a sufficiently long idle period MUST be set to min(4*MTU, max (2*MTU, 4380 bytes)).

New text: (Section 7.2.1)

- o The initial cwnd before DATA transmission MUST be set to min(4*MTU, max (2*MTU, 4380 bytes)).

Old text: (Section 7.2.1)

- o When the endpoint does not transmit data on a given transport address, the cwnd of the transport address should be adjusted to $\max(\text{cwnd}/2, 4 \cdot \text{MTU})$ per RTO.

New text: (Section 7.2.1)

- o While the endpoint does not transmit data on a given transport address, the cwnd of the transport address SHOULD be adjusted to $\max(\text{cwnd}/2, 4 \cdot \text{MTU})$ once per RTO. Before the first cwnd adjustment, the ssthresh of the transport address SHOULD be set to the cwnd.

This text is in final form, and is not further updated in this document.

3.27.3. Solution Description

A rule about cwnd adjustment after a sufficiently long idle period is removed.

The text is updated to describe the ssthresh handling. When the idle period is detected, the cwnd value is stored to the ssthresh value.

3.28. Window Updates After Receiver Window Opens Up

3.28.1. Description of the Problem

The sending of SACK chunks for window updates is only indirectly referenced in [RFC4960], Section 6.2, where it is stated that an SCTP receiver must not generate more than one SACK for every incoming packet, other than to update the offered window.

However, the sending of window updates when the receiver window opens up is necessary to avoid performance problems.

3.28.2. Text Changes to the Document

Old text: (Section 6.2)

An SCTP receiver MUST NOT generate more than one SACK for every incoming packet, other than to update the offered window as the receiving application consumes new data.

New text: (Section 6.2)

An SCTP receiver MUST NOT generate more than one SACK for every incoming packet, other than to update the offered window as the receiving application consumes new data. When the window opens up, an SCTP receiver SHOULD send additional SACK chunks to update the window even if no new data is received. The receiver MUST avoid sending a large number of window updates, in particular large bursts of them. One way to achieve this is to send a window update only if the window can be increased by at least a quarter of the receive buffer size of the association.

This text is in final form, and is not further updated in this document.

3.28.3. Solution Description

The new text makes clear that additional SACK chunks for window updates should be sent as long as excessive bursts are avoided.

3.29. Path of DATA and Reply Chunks

3.29.1. Description of the Problem

Section 6.4 of [RFC4960] describes the transmission policy for multi-homed SCTP endpoints. However, there are the following issues with it:

- o It states that a SACK should be sent to the source address of an incoming DATA. However, it is known that other SACK policies (e.g. sending SACKs always to the primary path) may be more beneficial in some situations.
- o Initially it states that an endpoint should always transmit DATA chunks to the primary path. Then it states that the rule for transmittal of reply chunks should also be followed if the endpoint is bundling DATA chunks together with the reply chunk which contradicts with the first statement to always transmit DATA

chunks to the primary path. Some implementations were having problems with it and sent DATA chunks bundled with reply chunks to a different destination address than the primary path that caused many gaps.

3.29.2. Text Changes to the Document

Old text: (Section 6.4)

An endpoint SHOULD transmit reply chunks (e.g., SACK, HEARTBEAT ACK, etc.) to the same destination transport address from which it received the DATA or control chunk to which it is replying. This rule should also be followed if the endpoint is bundling DATA chunks together with the reply chunk.

However, when acknowledging multiple DATA chunks received in packets from different source addresses in a single SACK, the SACK chunk may be transmitted to one of the destination transport addresses from which the DATA or control chunks being acknowledged were received.

New text: (Section 6.4)

An endpoint SHOULD transmit reply chunks (e.g., INIT ACK, COOKIE ACK, HEARTBEAT ACK, etc.) in response to control chunks to the same destination transport address from which it received the control chunk to which it is replying.

The selection of the destination transport address for packets containing SACK chunks is implementation dependent. However, an endpoint SHOULD NOT vary the destination transport address of a SACK when it receives DATA chunks coming from the same source address.

When acknowledging multiple DATA chunks received in packets from different source addresses in a single SACK, the SACK chunk MAY be transmitted to one of the destination transport addresses from which the DATA or control chunks being acknowledged were received.

This text is in final form, and is not further updated in this document.

3.29.3. Solution Description

The SACK transmission policy is left implementation dependent but it is specified to not vary the destination address of a packet containing a SACK chunk unless there are reasons for it as it may negatively impact RTT measurement.

A confusing statement that prescribes to follow the rule for transmittal of reply chunks when the endpoint is bundling DATA chunks together with the reply chunk is removed.

3.30. Outstanding Data, Flightsize and Data In Flight Key Terms

3.30.1. Description of the Problem

[RFC4960] uses outstanding data, flightsize and data in flight key terms in formulas and statements but their definitions are not provided in Section 1.3. Furthermore, outstanding data does not include DATA chunks which are classified as lost but which have not been retransmitted yet and there is a paragraph in Section 6.1 of [RFC4960] where this statement is broken.

3.30.2. Text Changes to the Document

Old text: (Section 1.3)

- o Congestion window (cwnd): An SCTP variable that limits the data, in number of bytes, a sender can send to a particular destination transport address before receiving an acknowledgement.

...

- o Outstanding TSN (at an SCTP endpoint): A TSN (and the associated DATA chunk) that has been sent by the endpoint but for which it has not yet received an acknowledgement.

New text: (Section 1.3)

- o Outstanding TSN (at an SCTP endpoint): A TSN (and the associated DATA chunk) that has been sent by the endpoint but for which it has not yet received an acknowledgement.
- o Outstanding data (or Data outstanding or Data in flight): The total amount of the DATA chunks associated with outstanding TSNs. A retransmitted DATA chunk is counted once in outstanding data. A DATA chunk which is classified as lost but which has not been retransmitted yet is not in outstanding data.
- o Flightsize: The amount of bytes of outstanding data to a particular destination transport address at any given time.
- o Congestion window (cwnd): An SCTP variable that limits outstanding data, in number of bytes, a sender can send to a particular destination transport address before receiving an acknowledgement.

This text is in final form, and is not further updated in this document.

Old text: (Section 6.1)

- C) When the time comes for the sender to transmit, before sending new DATA chunks, the sender MUST first transmit any outstanding DATA chunks that are marked for retransmission (limited by the current cwnd).

New text: (Section 6.1)

- C) When the time comes for the sender to transmit, before sending new DATA chunks, the sender MUST first transmit any DATA chunks that are marked for retransmission (limited by the current cwnd).

This text is in final form, and is not further updated in this document.

3.30.3. Solution Description

Now Section 1.3, Key Terms, includes explanations of outstanding data, data in flight and flightsize key terms. Section 6.1 is corrected to properly use the outstanding data term.

3.31. CWND Degradation due to Max.Burst

3.31.1. Description of the Problem

Some implementations were experiencing a degradation of cwnd because of the Max.Burst limit. This was due to misinterpretation of the suggestion in [RFC4960], Section 6.1, on how to use the Max.Burst parameter when calculating the number of packets to transmit.

3.31.2. Text Changes to the Document

Old text: (Section 6.1)

- D) When the time comes for the sender to transmit new DATA chunks, the protocol parameter Max.Burst SHOULD be used to limit the number of packets sent. The limit MAY be applied by adjusting cwnd as follows:

```
if((flightsize + Max.Burst*MTU) < cwnd) cwnd = flightsize +
Max.Burst*MTU
```

Or it MAY be applied by strictly limiting the number of packets emitted by the output routine.

New text: (Section 6.1)

- D) When the time comes for the sender to transmit new DATA chunks, the protocol parameter Max.Burst SHOULD be used to limit the number of packets sent. The limit MAY be applied by adjusting cwnd temporarily as follows:

```
if ((flightsize + Max.Burst*MTU) < cwnd)
    cwnd = flightsize + Max.Burst*MTU
```

Or it MAY be applied by strictly limiting the number of packets emitted by the output routine. When calculating the number of packets to transmit and particularly using the formula above, cwnd SHOULD NOT be changed permanently.

This text is in final form, and is not further updated in this document.

3.31.3. Solution Description

The new text clarifies that cwnd should not be changed when applying the Max.Burst limit. This mitigates packet bursts related to the reception of SACK chunks, but not bursts related to an application sending a burst of user messages.

3.32. Reduction of RTO.Initial

3.32.1. Description of the Problem

[RFC4960] uses 3 seconds as the default value for RTO.Initial in accordance with Section 4.3.2.1 of [RFC1122]. [RFC6298] updates [RFC1122] and lowers the initial value of the retransmission timer from 3 seconds to 1 second.

3.32.2. Text Changes to the Document

Old text: (Section 15)

The following protocol parameters are RECOMMENDED:

RTO.Initial - 3 seconds
RTO.Min - 1 second
RTO.Max - 60 seconds
Max.Burst - 4
RTO.Alpha - 1/8
RTO.Beta - 1/4
Valid.Cookie.Life - 60 seconds
Association.Max.Retrans - 10 attempts
Path.Max.Retrans - 5 attempts (per destination address)
Max.Init.Retransmits - 8 attempts
HB.interval - 30 seconds
HB.Max.Burst - 1

New text: (Section 15)

The following protocol parameters are RECOMMENDED:

RTO.Initial - 1 second
RTO.Min - 1 second
RTO.Max - 60 seconds
Max.Burst - 4
RTO.Alpha - 1/8
RTO.Beta - 1/4
Valid.Cookie.Life - 60 seconds
Association.Max.Retrans - 10 attempts
Path.Max.Retrans - 5 attempts (per destination address)
Max.Init.Retransmits - 8 attempts
HB.interval - 30 seconds
HB.Max.Burst - 1
SACK.Delay - 200 milliseconds

This text has been modified by multiple errata. It includes modifications from Section 3.24. It is in final form, and is not further updated in this document.

3.32.3. Solution Description

The value `RTO.Initial` has been lowered to 1 second to be in tune with [RFC6298].

3.33. Ordering of Bundled SACK and ERROR Chunks

3.33.1. Description of the Problem

When an SCTP endpoint receives a DATA chunk with an invalid stream identifier it shall acknowledge it by sending a SACK chunk and indicate that the stream identifier was invalid by sending an ERROR chunk. These two chunks may be bundled. However, [RFC4960] requires in case of bundling that the ERROR chunk follows the SACK chunk. This restriction of the ordering is not necessary and might only limit interoperability.

3.33.2. Text Changes to the Document

Old text: (Section 6.5)

Every DATA chunk MUST carry a valid stream identifier. If an endpoint receives a DATA chunk with an invalid stream identifier, it shall acknowledge the reception of the DATA chunk following the normal procedure, immediately send an ERROR chunk with cause set to "Invalid Stream Identifier" (see Section 3.3.10), and discard the DATA chunk. The endpoint may bundle the ERROR chunk in the same packet as the SACK as long as the ERROR follows the SACK.

New text: (Section 6.5)

Every DATA chunk MUST carry a valid stream identifier. If an endpoint receives a DATA chunk with an invalid stream identifier, it SHOULD acknowledge the reception of the DATA chunk following the normal procedure, immediately send an ERROR chunk with cause set to "Invalid Stream Identifier" (see Section 3.3.10), and discard the DATA chunk. The endpoint MAY bundle the ERROR chunk and the SACK Chunk in the same packet.

This text is in final form, and is not further updated in this document.

3.33.3. Solution Description

The unnecessary restriction regarding the ordering of the SACK and ERROR chunk has been removed.

3.34. Undefined Parameter Returned by RECEIVE Primitive

3.34.1. Description of the Problem

[RFC4960] provides a description of an abstract API. In the definition of the RECEIVE primitive an optional parameter with name "delivery number" is mentioned. However, no definition of this parameter is given in [RFC4960] and the parameter is unnecessary.

3.34.2. Text Changes to the Document

Old text: (Section 10.1 G))

G) Receive

Format: RECEIVE(association id, buffer address, buffer size
[,stream id])

-> byte count [,transport address] [,stream id] [,stream sequence
number] [,partial flag] [,delivery number] [,payload protocol-id]

New text: (Section 10.1 G))

G) Receive

Format: RECEIVE(association id, buffer address, buffer size
[,stream id])

-> byte count [,transport address] [,stream id] [,stream sequence
number] [,partial flag] [,payload protocol-id]

This text is in final form, and is not further updated in this document.

3.34.3. Solution Description

The undefined parameter has been removed.

3.35. DSCP Changes

3.35.1. Description of the Problem

The upper layer can change the Differentiated Services Code Point (DSCP) used for packets being sent. A change of the DSCP can result in packets hitting different queues on the path and, therefore, the congestion control should be initialized when the DSCP is changed by the upper layer. This is not described in [RFC4960].

3.35.2. Text Changes to the Document

New text: (Section 7.2.5)

7.2.5. Change of Differentiated Services Code Points

SCTP implementations MAY allow an application to configure the Differentiated Services Code Point (DSCP) used for sending packets. If a DSCP change might result in outgoing packets being queued in different queues, the congestion control parameters for all affected destination addresses MUST be reset to their initial values.

This text is in final form, and is not further updated in this document.

Old text: (Section 10.1 M)

Mandatory attributes:

- o association id - local handle to the SCTP association.
- o protocol parameter list - the specific names and values of the protocol parameters (e.g., Association.Max.Retrans; see Section 15) that the SCTP user wishes to customize.

New text: (Section 10.1 M)

Mandatory attributes:

- o association id - local handle to the SCTP association.
- o protocol parameter list - the specific names and values of the protocol parameters (e.g., Association.Max.Retrans; see Section 15, or other parameters like the DSCP) that the SCTP user wishes to customize.

This text is in final form, and is not further updated in this document.

3.35.3. Solution Description

Text describing the required action on DSCP changes has been added.

3.36. Inconsistent Handling of ICMPv4 and ICMPv6 Messages

3.36.1. Description of the Problem

Appendix C of [RFC4960] describes the handling of ICMPv4 and ICMPv6 messages. The handling of ICMP messages indicating that the port number is unreachable described in the enumeration is not consistent with the description given in [RFC4960] after the enumeration. Furthermore, the text explicitly describes the handling of ICMPv6 packets indicating reachability problems, but does not do the same for the corresponding ICMPv4 packets.

3.36.2. Text Changes to the Document

Old text: (Appendix C)

ICMP3) An implementation MAY ignore any ICMPv4 messages where the code does not indicate "Protocol Unreachable" or "Fragmentation Needed".

New text: (Appendix C)

ICMP3) An implementation SHOULD ignore any ICMP messages where the code indicates "Port Unreachable".

This text is in final form, and is not further updated in this document.

Old text: (Appendix C)

ICMP9) If the ICMPv6 code is "Destination Unreachable", the implementation MAY mark the destination into the unreachable state or alternatively increment the path error counter.

New text: (Appendix C)

ICMP9) If the ICMP type is "Destination Unreachable", the implementation MAY mark the destination into the unreachable state or alternatively increment the path error counter.

This text has been modified by multiple errata. It is further updated in Section 3.37.

3.36.3. Solution Description

The text has been changed to describe the intended handling of ICMP messages indicating that the port number is unreachable by replacing the third rule. Furthermore, remove the limitation to ICMPv6 in the ninth rule.

3.37. Handling of Soft Errors

3.37.1. Description of the Problem

[RFC1122] defines the handling of soft errors and hard errors for TCP. Appendix C of [RFC4960] only deals with hard errors.

3.37.2. Text Changes to the Document

Old text: (Appendix C)

ICMP9) If the ICMPv6 code is "Destination Unreachable", the implementation MAY mark the destination into the unreachable state or alternatively increment the path error counter.

New text: (Appendix C)

ICMP9) If the ICMP type is "Destination Unreachable", the implementation MAY mark the destination into the unreachable state or alternatively increment the path error counter. SCTP MAY provide information to the upper layer indicating the reception of ICMP messages when reporting a network status change.

This text has been modified by multiple errata. It includes modifications from Section 3.36. It is in final form, and is not further updated in this document.

3.37.3. Solution Description

Text has been added allowing SCTP to notify the application in case of soft errors.

3.38. Honoring CWND

3.38.1. Description of the Problem

When using the slow start algorithm, SCTP increases the congestion window only when it is being fully utilized. Since SCTP uses DATA chunks and does not use the congestion window to fragment user messages, this requires that some overbooking of the congestion window is allowed.

3.38.2. Text Changes to the Document

Old text: (Section 6.1)

- B) At any given time, the sender MUST NOT transmit new data to a given transport address if it has cwnd or more bytes of data outstanding to that transport address.

New text: (Section 6.1)

- B) At any given time, the sender MUST NOT transmit new data to a given transport address if it has cwnd + (PMTU - 1) or more bytes of data outstanding to that transport address. If data is available the sender SHOULD exceed cwnd by up to (PMTU-1) bytes on a new data transmission if the flightsize does not currently reach cwnd. The breach of cwnd MUST constitute one packet only.

This text is in final form, and is not further updated in this document.

Old text: (Section 7.2.1)

- o Whenever cwnd is greater than zero, the endpoint is allowed to have cwnd bytes of data outstanding on that transport address.

New text: (Section 7.2.1)

- o Whenever cwnd is greater than zero, the endpoint is allowed to have cwnd bytes of data outstanding on that transport address. A limited overbooking as described in B) of Section 6.1 SHOULD be supported.

This text is in final form, and is not further updated in this document.

3.38.3. Solution Description

Text was added to clarify how the cwnd limit should be handled.

3.39. Zero Window Probing

3.39.1. Description of the Problem

The text describing zero window probing was not clearly handling the case where the window was not zero, but too small for the next DATA chunk to be transmitted. Even in this case, zero window probing has to be performed to avoid deadlocks.

3.39.2. Text Changes to the Document

Old text: (Section 6.1)

- A) At any given time, the data sender MUST NOT transmit new data to any destination transport address if its peer's `rwnd` indicates that the peer has no buffer space (i.e., `rwnd` is 0; see Section 6.2.1). However, regardless of the value of `rwnd` (including if it is 0), the data sender can always have one DATA chunk in flight to the receiver if allowed by `cwnd` (see rule B, below). This rule allows the sender to probe for a change in `rwnd` that the sender missed due to the SACK's having been lost in transit from the data receiver to the data sender.

When the receiver's advertised window is zero, this probe is called a zero window probe. Note that a zero window probe SHOULD only be sent when all outstanding DATA chunks have been cumulatively acknowledged and no DATA chunks are in flight. Zero window probing MUST be supported.

New text: (Section 6.1)

- A) At any given time, the data sender MUST NOT transmit new data to any destination transport address if its peer's `rwnd` indicates that the peer has no buffer space (i.e., `rwnd` is smaller than the size of the next DATA chunk; see Section 6.2.1). However, regardless of the value of `rwnd` (including if it is 0), the data sender can always have one DATA chunk in flight to the receiver if allowed by `cwnd` (see rule B, below). This rule allows the sender to probe for a change in `rwnd` that the sender missed due to the SACK's having been lost in transit from the data receiver to the data sender.

When the receiver has no buffer space, this probe is called a zero window probe. Note that a zero window probe SHOULD only be sent when all outstanding DATA chunks have been cumulatively acknowledged and no DATA chunks are in flight. Zero window probing MUST be supported.

This text is in final form, and is not further updated in this document.

3.39.3. Solution Description

The terminology is used in a cleaner way.

3.40. Updating References Regarding ECN

3.40.1. Description of the Problem

[RFC4960] refers for ECN only to [RFC3168], which will be updated by [RFC8311]. This needs to be reflected when referring to ECN.

3.40.2. Text Changes to the Document

Old text: (Appendix A)

ECN [RFC3168] describes a proposed extension to IP that details a method to become aware of congestion outside of datagram loss.

New text: (Appendix A)

ECN as specified in [RFC3168] updated by [RFC8311] describes an extension to IP that details a method to become aware of congestion outside of datagram loss.

This text is in final form, and is not further updated in this document.

Old text: (Appendix A)

In general, [RFC3168] should be followed with the following exceptions.

New text: (Appendix A)

In general, [RFC3168] updated by [RFC8311] SHOULD be followed with the following exceptions.

This text is in final form, and is not further updated in this document.

Old text: (Appendix A)

[RFC3168] details negotiation of ECN during the SYN and SYN-ACK stages of a TCP connection.

New text: (Appendix A)

[RFC3168] updated by [RFC8311] details negotiation of ECN during the SYN and SYN-ACK stages of a TCP connection.

This text is in final form, and is not further updated in this document.

Old text: (Appendix A)

[RFC3168] details a specific bit for a receiver to send back in its TCP acknowledgements to notify the sender of the Congestion Experienced (CE) bit having arrived from the network.

New text: (Appendix A)

[RFC3168] updated by [RFC8311] details a specific bit for a receiver to send back in its TCP acknowledgements to notify the sender of the Congestion Experienced (CE) bit having arrived from the network.

This text is in final form, and is not further updated in this document.

Old text: (Appendix A)

[RFC3168] details a specific bit for a sender to send in the header of its next outbound TCP segment to indicate to its peer that it has reduced its congestion window.

New text: (Appendix A)

[RFC3168] updated by [RFC8311] details a specific bit for a sender to send in the header of its next outbound TCP segment to indicate to its peer that it has reduced its congestion window.

This text is in final form, and is not further updated in this document.

3.40.3. Solution Description

References to [RFC8311] have been added. While there, some wordsmithing has been performed.

3.41. Host Name Address Parameter Deprecated

3.41.1. Description of the Problem

[RFC4960] defines three types of address parameters to be used with INIT and INIT ACK chunks:

1. IPv4 Address parameters.
2. IPv6 Address parameters.
3. Host Name Address parameters.

The first two are supported by the SCTP kernel implementations of FreeBSD, Linux and Solaris, but the third one is not. In addition, the first two were successfully tested in all nine interoperability tests for SCTP, but the third one has never been successfully tested. Therefore, the Host Name Address parameter should be deprecated.

3.41.2. Text Changes to the Document

Old text: (Section 3.3.2)

Note 3: An INIT chunk MUST NOT contain more than one Host Name Address parameter. Moreover, the sender of the INIT MUST NOT combine any other address types with the Host Name Address in the INIT. The receiver of INIT MUST ignore any other address types if the Host Name Address parameter is present in the received INIT chunk.

New text: (Section 3.3.2)

Note 3: An INIT chunk MUST NOT contain the Host Name Address parameter. The receiver of an INIT chunk containing a Host Name Address parameter MUST send an ABORT and MAY include an Error Cause indicating an Unresolvable Address.

This text is in final form, and is not further updated in this document.

Old text: (Section 3.3.2.1)

The sender of INIT uses this parameter to pass its Host Name (in place of its IP addresses) to its peer. The peer is responsible for resolving the name. Using this parameter might make it more likely for the association to work across a NAT box.

New text: (Section 3.3.2.1)

The sender of an INIT chunk MUST NOT include this parameter. The usage of the Host Name Address parameter is deprecated.

This text is in final form, and is not further updated in this document.

Old text: (Section 3.3.2.1)

Address Type: 16 bits (unsigned integer)

This is filled with the type value of the corresponding address TLV (e.g., IPv4 = 5, IPv6 = 6, Host name = 11).

New text: (Section 3.3.2.1)

Address Type: 16 bits (unsigned integer)

This is filled with the type value of the corresponding address TLV (e.g., IPv4 = 5, IPv6 = 6). The value indicating the Host Name Address parameter (Host name = 11) MUST NOT be used.

This text is in final form, and is not further updated in this document.

Old text: (Section 3.3.3)

Note 3: The INIT ACK chunks MUST NOT contain more than one Host Name Address parameter. Moreover, the sender of the INIT ACK MUST NOT combine any other address types with the Host Name Address in the INIT ACK. The receiver of the INIT ACK MUST ignore any other address types if the Host Name Address parameter is present.

New text: (Section 3.3.3)

Note 3: An INIT ACK chunk MUST NOT contain the Host Name Address parameter. The receiver of INIT ACK chunks containing a Host Name Address parameter MUST send an ABORT and MAY include an Error Cause indicating an Unresolvable Address.

This text is in final form, and is not further updated in this document.

Old text: (Section 5.1.2)

B) If there is a Host Name parameter present in the received INIT or

INIT ACK chunk, the endpoint shall resolve that host name to a list of IP address(es) and derive the transport address(es) of this peer by combining the resolved IP address(es) with the SCTP source port.

The endpoint MUST ignore any other IP Address parameters if they are also present in the received INIT or INIT ACK chunk.

The time at which the receiver of an INIT resolves the host name has potential security implications to SCTP. If the receiver of an INIT resolves the host name upon the reception of the chunk, and the mechanism the receiver uses to resolve the host name involves potential long delay (e.g., DNS query), the receiver may open itself up to resource attacks for the period of time while it is waiting for the name resolution results before it can build the State Cookie and release local resources.

Therefore, in cases where the name translation involves potential long delay, the receiver of the INIT MUST postpone the name resolution till the reception of the COOKIE ECHO chunk from the peer. In such a case, the receiver of the INIT SHOULD build the State Cookie using the received Host Name (instead of destination transport addresses) and send the INIT ACK to the source IP address from which the INIT was received.

The receiver of an INIT ACK shall always immediately attempt to resolve the name upon the reception of the chunk.

The receiver of the INIT or INIT ACK MUST NOT send user data (piggy-backed or stand-alone) to its peer until the host name is successfully resolved.

If the name resolution is not successful, the endpoint MUST immediately send an ABORT with "Unresolvable Address" error cause to its peer. The ABORT shall be sent to the source IP address from which the last peer packet was received.

New text: (Section 5.1.2)

- B) If there is a Host Name parameter present in the received INIT or INIT ACK chunk, the endpoint MUST immediately send an ABORT and MAY include an Error Cause indicating an Unresolvable Address to its peer. The ABORT SHALL be sent to the source IP address from which the last peer packet was received.

This text is in final form, and is not further updated in this document.

Old text: (Section 11.2.4.1)

The use of the host name feature in the INIT chunk could be used to flood a target DNS server. A large backlog of DNS queries, resolving the host name received in the INIT chunk to IP addresses, could be accomplished by sending INITs to multiple hosts in a given domain. In addition, an attacker could use the host name feature in an indirect attack on a third party by sending large numbers of INITs to random hosts containing the host name of the target. In addition to the strain on DNS resources, this could also result in large numbers of INIT ACKs being sent to the target. One method to protect against this type of attack is to verify that the IP addresses received from DNS include the source IP address of the original INIT. If the list of IP addresses received from DNS does not include the source IP address of the INIT, the endpoint MAY silently discard the INIT. This last option will not protect against the attack against the DNS.

New text: (Section 11.2.4.1)

The support of the Host Name Address parameter has been removed from the protocol. Endpoints receiving INIT or INIT ACK chunks containing the Host Name Address parameter MUST send an ABORT chunk in response and MAY include an Error Cause indicating an Unresolvable Address.

This text is in final form, and is not further updated in this document.

3.41.3. Solution Description

The usage of the Host Name Address parameter has been deprecated.

3.42. Conflicting Text Regarding the Supported Address Types Parameter

3.42.1. Description of the Problem

When receiving an SCTP packet containing an INIT chunk sent from an address for which the corresponding address type is not listed in the Supported Address Types, there is conflicting text in Section 5.1.2 of [RFC4960]. It is stated that the association MUST be aborted and also that the association SHOULD be established and there SHOULD NOT be any error indication.

3.42.2. Text Changes to the Document

Old text: (Section 5.1.2)

The sender of INIT may include a 'Supported Address Types' parameter in the INIT to indicate what types of address are acceptable. When this parameter is present, the receiver of INIT (initiate) MUST either use one of the address types indicated in the Supported Address Types parameter when responding to the INIT, or abort the association with an "Unresolvable Address" error cause if it is unwilling or incapable of using any of the address types indicated by its peer.

New text: (Section 5.1.2)

The sender of INIT chunks MAY include a 'Supported Address Types' parameter in the INIT to indicate what types of addresses are acceptable.

This text is in final form, and is not further updated in this document.

3.42.3. Solution Description

The conflicting text has been removed.

3.43. Integration of RFC 6096

3.43.1. Description of the Problem

[RFC6096] updates [RFC4960] by adding a Chunk Flags Registry. This should be integrated into the base specification.

3.43.2. Text Changes to the Document

Old text: (Section 14.1)

14.1. IETF-Defined Chunk Extension

The assignment of new chunk parameter type codes is done through an IETF Consensus action, as defined in [RFC2434]. Documentation of the chunk parameter MUST contain the following information:

- a) A long and short name for the new chunk type.
- b) A detailed description of the structure of the chunk, which MUST conform to the basic structure defined in Section 3.2.
- c) A detailed definition and description of the intended use of each field within the chunk, including the chunk flags if any.
- d) A detailed procedural description of the use of the new chunk type within the operation of the protocol.

The last chunk type (255) is reserved for future extension if necessary.

New text: (Section 14.1)

14.1. IETF-Defined Chunk Extension

The assignment of new chunk type codes is done through an IETF Review action, as defined in [RFC8126]. Documentation of a new chunk MUST contain the following information:

- a) A long and short name for the new chunk type;
- b) A detailed description of the structure of the chunk, which MUST conform to the basic structure defined in Section 3.2 of [RFC4960];
- c) A detailed definition and description of the intended use of each field within the chunk, including the chunk flags if any. Defined chunk flags will be used as initial entries in the chunk flags table for the new chunk type;
- d) A detailed procedural description of the use of the new chunk type within the operation of the protocol.

The last chunk type (255) is reserved for future extension if necessary.

For each new chunk type, IANA creates a registration table for the chunk flags of that type. The procedure for registering particular chunk flags is described in the following Section 14.2.

This text has been modified by multiple errata. It includes modifications from Section 3.3. It is in final form, and is not further updated in this document.

New text: (Section 14.2)

14.2. New IETF Chunk Flags Registration

The assignment of new chunk flags is done through an RFC required action, as defined in [RFC8126]. Documentation of the chunk flags MUST contain the following information:

- a) A name for the new chunk flag;
- b) A detailed procedural description of the use of the new chunk flag within the operation of the protocol. It MUST be considered that implementations not supporting the flag will send '0' on transmit and just ignore it on receipt.

IANA selects a chunk flags value. This MUST be one of 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, or 0x80, which MUST be unique within the chunk flag values for the specific chunk type.

This text is in final form, and is not further updated in this document.

Please note that Sections 14.2, 14.3, 14.4, and 14.5 need to be renumbered.

3.43.3. Solution Description

[RFC6096] was integrated and the reference updated to [RFC8126].

3.44. Integration of RFC 6335

3.44.1. Description of the Problem

[RFC6335] updates [RFC4960] by updating Procedures for the Port Numbers Registry. This should be integrated into the base specification. While there, update the reference to the RFC giving guidelines for writing IANA sections to [RFC8126].

3.44.2. Text Changes to the Document

Old text: (Section 14.5)

SCTP services may use contact port numbers to provide service to unknown callers, as in TCP and UDP. IANA is therefore requested to

open the existing Port Numbers registry for SCTP using the following rules, which we intend to mesh well with existing Port Numbers registration procedures. An IESG-appointed Expert Reviewer supports IANA in evaluating SCTP port allocation requests, according to the procedure defined in [RFC2434].

Port numbers are divided into three ranges. The Well Known Ports are those from 0 through 1023, the Registered Ports are those from 1024 through 49151, and the Dynamic and/or Private Ports are those from 49152 through 65535. Well Known and Registered Ports are intended for use by server applications that desire a default contact point on a system. On most systems, Well Known Ports can only be used by system (or root) processes or by programs executed by privileged users, while Registered Ports can be used by ordinary user processes or programs executed by ordinary users. Dynamic and/or Private Ports are intended for temporary use, including client-side ports, out-of-band negotiated ports, and application testing prior to registration of a dedicated port; they MUST NOT be registered.

The Port Numbers registry should accept registrations for SCTP ports in the Well Known Ports and Registered Ports ranges. Well Known and Registered Ports SHOULD NOT be used without registration. Although in some cases -- such as porting an application from TCP to SCTP -- it may seem natural to use an SCTP port before registration completes, we emphasize that IANA will not guarantee registration of particular Well Known and Registered Ports. Registrations should be requested as early as possible.

Each port registration SHALL include the following information:

- o A short port name, consisting entirely of letters (A-Z and a-z), digits (0-9), and punctuation characters from "-_+./*" (not including the quotes).
- o The port number that is requested for registration.
- o A short English phrase describing the port's purpose.
- o Name and contact information for the person or entity performing the registration, and possibly a reference to a document defining the port's use. Registrations coming from IETF working groups need only name the working group, but indicating a contact person is recommended.

Registrants are encouraged to follow these guidelines when submitting a registration.

- o A port name SHOULD NOT be registered for more than one SCTP port


```

|  Type = 0      | Reserved|U|B|E|      Length      |
+-----+-----+-----+-----+-----+-----+
|                                     TSN                                     |
+-----+-----+-----+-----+-----+-----+
|  Stream Identifier S      | Stream Sequence Number n      |
+-----+-----+-----+-----+-----+-----+
|                                     Payload Protocol Identifier                                     |
+-----+-----+-----+-----+-----+-----+
\                                                                                   \
/                                     User Data (seq n of Stream S)                  /
\                                                                                   \
+-----+-----+-----+-----+-----+-----+

```

Reserved: 5 bits

Should be set to all '0's and ignored by the receiver.

New text: (Section 3.3.1)

```

0           1           2           3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|  Type = 0      | Res  |I|U|B|E|      Length      |
+-----+-----+-----+-----+-----+-----+-----+
|                                     TSN                                     |
+-----+-----+-----+-----+-----+-----+
|  Stream Identifier S      | Stream Sequence Number n      |
+-----+-----+-----+-----+-----+-----+
|                                     Payload Protocol Identifier                                     |
+-----+-----+-----+-----+-----+-----+
\                                                                                   \
/                                     User Data (seq n of Stream S)                  /
\                                                                                   \
+-----+-----+-----+-----+-----+-----+

```

Res: 4 bits

SHOULD be set to all '0's and ignored by the receiver.

I bit: 1 bit

The (I)mmmediate Bit MAY be set by the sender, whenever the sender of a DATA chunk can benefit from the corresponding SACK chunk being sent back without delay. See [RFC7053] for a discussion about

This text is in final form, and is not further updated in this document.

New text: (Append to Section 6.1)

Whenever the sender of a DATA chunk can benefit from the corresponding SACK chunk being sent back without delay, the sender MAY set the I bit in the DATA chunk header. Please note that why the sender has set the I bit is irrelevant to the receiver.

Reasons for setting the I bit include, but are not limited to (see Section 4 of [RFC7053] for the benefits):

- o The application requests to set the I bit of the last DATA chunk of a user message when providing the user message to the SCTP implementation (see Section 7).
- o The sender is in the SHUTDOWN-PENDING state.
- o The sending of a DATA chunk fills the congestion or receiver window.

This text is in final form, and is not further updated in this document.

Old text: (Section 6.2)

Note: The SHUTDOWN chunk does not contain Gap Ack Block fields. Therefore, the endpoint should use a SACK instead of the SHUTDOWN chunk to acknowledge DATA chunks received out of order.

New text: (Section 6.2)

Note: The SHUTDOWN chunk does not contain Gap Ack Block fields. Therefore, the endpoint SHOULD use a SACK instead of the SHUTDOWN chunk to acknowledge DATA chunks received out of order.

Upon receipt of an SCTP packet containing a DATA chunk with the I bit set, the receiver SHOULD NOT delay the sending of the corresponding SACK chunk, i.e., the receiver SHOULD immediately respond with the corresponding SACK chunk.

Please note that this change is only about adding a paragraph.

This text is in final form, and is not further updated in this document.

Old text: (Section 10.1 E))

E) Send

Format: SEND(association id, buffer address, byte count [,context]
[,stream id] [,life time] [,destination transport address]
[,unordered flag] [,no-bundle flag] [,payload protocol-id])
-> result

New text: (Section 10.1 E))

E) Send

Format: SEND(association id, buffer address, byte count [,context]
[,stream id] [,life time] [,destination transport address]
[,unordered flag] [,no-bundle flag] [,payload protocol-id]
[,sack immediately])
-> result

This text is in final form, and is not further updated in this document.

New text: (Append optional parameter in Subsection E of Section 10.1)

- o sack immediately - set the I bit on the last DATA chunk used for sending buffer.

This text is in final form, and is not further updated in this document.

3.45.3. Solution Description

[RFC7053] was integrated.

3.46. CRC32c Code Improvements

3.46.1. Description of the Problem

The code given for the CRC32c computations uses types like long which may have different length on different operating systems or processors. Therefore, the code is changed to use specific types like uint32_t.

While there, fix also some syntax errors and a comment.

3.46.2. Text Changes to the Document

```

-----
Old text: (Appendix C)
-----
/*****
/* Note Definition for Ross Williams table generator would */
/* be: TB_WIDTH=4, TB_POLLY=0x1EDC6F41, TB_REVER=TRUE */
/* For Mr. Williams direct calculation code use the settings */
/* cm_width=32, cm_poly=0x1EDC6F41, cm_init=0xFFFFFFFF, */
/* cm_refin=TRUE, cm_refot=TRUE, cm_xorort=0x00000000 */
*****/

/* Example of the crc table file */
#ifndef __crc32cr_table_h__
#define __crc32cr_table_h__

#define CRC32C_POLY 0x1EDC6F41
#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])

unsigned long crc_c[256] =
{
0x00000000L, 0xF26B8303L, 0xE13B70F7L, 0x1350F3F4L,
0xC79A971FL, 0x35F1141CL, 0x26A1E7E8L, 0xD4CA64EBL,
0x8AD958CFL, 0x78B2DBCCL, 0x6BE22838L, 0x9989AB3BL,
0x4D43CFD0L, 0xBF284CD3L, 0xAC78BF27L, 0x5E133C24L,
0x105EC76FL, 0xE235446CL, 0xF165B798L, 0x030E349BL,
0xD7C45070L, 0x25AFD373L, 0x36FF2087L, 0xC494A384L,
0x9A879FA0L, 0x68EC1CA3L, 0x7BBCEF57L, 0x89D76C54L,
0x5D1D08BFL, 0xAF768BBCL, 0xBC267848L, 0x4E4DFB4BL,
0x20BD8EDEL, 0xD2D60DDDL, 0xC186FE29L, 0x33ED7D2AL,
0xE72719C1L, 0x154C9AC2L, 0x061C6936L, 0xF477EA35L,
0xAA64D611L, 0x580F5512L, 0x4B5FA6E6L, 0xB93425E5L,
0x6DFE410EL, 0x9F95C20DL, 0x8CC531F9L, 0x7EAE2FAL,
0x30E349B1L, 0xC288CAB2L, 0xD1D83946L, 0x23B3BA45L,

```

0xF779DEAEL, 0x05125DADL, 0x1642AE59L, 0xE4292D5AL,
0xBA3A117EL, 0x4851927DL, 0x5B016189L, 0xA96AE28AL,
0x7DA08661L, 0x8FCB0562L, 0x9C9BF696L, 0x6EF07595L,
0x417B1DBCL, 0xB3109EBFL, 0xA0406D4BL, 0x522BEE48L,
0x86E18AA3L, 0x748A09A0L, 0x67DAFA54L, 0x95B17957L,
0xCBA24573L, 0x39C9C670L, 0x2A993584L, 0xD8F2B687L,
0x0C38D26CL, 0xFE53516FL, 0xED03A29BL, 0x1F682198L,
0x5125DAD3L, 0xA34E59D0L, 0xB01EAA24L, 0x42752927L,
0x96BF4DCCL, 0x64D4CECFL, 0x77843D3BL, 0x85EFBE38L,
0xDBFC821CL, 0x2997011FL, 0x3AC7F2EBL, 0xC8AC71E8L,
0x1C661503L, 0xEE0D9600L, 0xFD5D65F4L, 0x0F36E6F7L,
0x61C69362L, 0x93AD1061L, 0x80FDE395L, 0x72966096L,
0xA65C047DL, 0x5437877EL, 0x4767748AL, 0xB50CF789L,
0xEB1FCBADL, 0x197448AEL, 0x0A24BB5AL, 0xF84F3859L,
0x2C855CB2L, 0xDEEEDFB1L, 0xCDBE2C45L, 0x3FD5AF46L,
0x7198540DL, 0x83F3D70EL, 0x90A324FAL, 0x62C8A7F9L,
0xB602C312L, 0x44694011L, 0x5739B3E5L, 0xA55230E6L,
0xFB410CC2L, 0x092A8FC1L, 0x1A7A7C35L, 0xE811FF36L,
0x3CDB9BDDL, 0xCEB018DEL, 0xDDE0EB2AL, 0x2F8B6829L,
0x82F63B78L, 0x709DB87BL, 0x63CD4B8FL, 0x91A6C88CL,
0x456CAC67L, 0xB7072F64L, 0xA457DC90L, 0x563C5F93L,
0x082F63B7L, 0xFA44E0B4L, 0xE9141340L, 0x1B7F9043L,
0xCFB5F4A8L, 0x3DDE77ABL, 0x2E8E845FL, 0xDCE5075CL,
0x92A8FC17L, 0x60C37F14L, 0x73938CE0L, 0x81F80FE3L,
0x55326B08L, 0xA759E80BL, 0xB4091BFFL, 0x466298FCL,
0x1871A4D8L, 0xEA1A27DBL, 0xF94AD42FL, 0x0B21572CL,
0xDFEB33C7L, 0x2D80B0C4L, 0x3ED04330L, 0xCCBBC033L,
0xA24BB5A6L, 0x502036A5L, 0x4370C551L, 0xB11B4652L,
0x65D122B9L, 0x97BAA1BAL, 0x84EA524EL, 0x7681D14DL,
0x2892ED69L, 0xDAF96E6AL, 0xC9A99D9EL, 0x3BC21E9DL,
0xEF087A76L, 0x1D63F975L, 0x0E330A81L, 0xFC588982L,
0xB21572C9L, 0x407EF1CAL, 0x532E023EL, 0xA145813DL,
0x758FE5D6L, 0x87E466D5L, 0x94B49521L, 0x66DF1622L,
0x38CC2A06L, 0xCAA7A905L, 0xD9F75AF1L, 0x2B9CD9F2L,
0xFF56BD19L, 0x0D3D3E1AL, 0x1E6DCDEEL, 0xEC064EEDL,
0xC38D26C4L, 0x31E6A5C7L, 0x22B65633L, 0xD0DDD530L,
0x0417B1DBL, 0xF67C32D8L, 0xE52CC12CL, 0x1747422FL,
0x49547E0BL, 0xBB3FFD08L, 0xA86F0EFCL, 0x5A048DFFL,
0x8ECEE914L, 0x7CA56A17L, 0x6FF599E3L, 0x9D9E1AE0L,
0xD3D3E1ABL, 0x21B862A8L, 0x32E8915CL, 0xC083125FL,
0x144976B4L, 0xE622F5B7L, 0xF5720643L, 0x07198540L,
0x590AB964L, 0xAB613A67L, 0xB831C993L, 0x4A5A4A90L,
0x9E902E7BL, 0x6CFBAD78L, 0x7FAB5E8CL, 0x8DC0DD8FL,
0xE330A81AL, 0x115B2B19L, 0x020BD8EDL, 0xF0605BEEL,
0x24AA3F05L, 0xD6C1BC06L, 0xC5914FF2L, 0x37FACCF1L,
0x69E9F0D5L, 0x9B8273D6L, 0x88D28022L, 0x7AB90321L,
0xAE7367CAL, 0x5C18E4C9L, 0x4F48173DL, 0xBD23943EL,
0xF36E6F75L, 0x0105EC76L, 0x12551F82L, 0xE03E9C81L,

```

0x34F4F86AL, 0xC69F7B69L, 0xD5CF889DL, 0x27A40B9EL,
0x79B737BAL, 0x8BDCB4B9L, 0x988C474DL, 0x6AE7C44EL,
0xBE2DA0A5L, 0x4C4623A6L, 0x5F16D052L, 0xAD7D5351L,
};

#endif

```

```

-----
New text: (Appendix B)
-----

```

```

<CODE BEGINS>
/*****
/* Note Definition for Ross Williams table generator would */
/* be: TB_WIDTH=4, TB_POLLY=0x1EDC6F41, TB_REVER=TRUE */
/* For Mr. Williams direct calculation code use the settings */
/* cm_width=32, cm_poly=0x1EDC6F41, cm_init=0xFFFFFFFF, */
/* cm_refin=TRUE, cm_refot=TRUE, cm_xorort=0x00000000 */
*****/

/* Example of the crc table file */
#ifndef __crc32cr_h__
#define __crc32cr_h__

#define CRC32C_POLY 0x1EDC6F41UL
#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])

uint32_t crc_c[256] =
{
0x00000000UL, 0xF26B8303UL, 0xE13B70F7UL, 0x1350F3F4UL,
0xC79A971FUL, 0x35F1141CUL, 0x26A1E7E8UL, 0xD4CA64EBUL,
0x8AD958CFUL, 0x78B2DBCCUL, 0x6BE22838UL, 0x9989AB3BUL,
0x4D43CFD0UL, 0xBF284CD3UL, 0xAC78BF27UL, 0x5E133C24UL,
0x105EC76FUL, 0xE235446CUL, 0xF165B798UL, 0x030E349BUL,
0xD7C45070UL, 0x25AFD373UL, 0x36FF2087UL, 0xC494A384UL,
0x9A879FA0UL, 0x68EC1CA3UL, 0x7BBCEF57UL, 0x89D76C54UL,
0x5D1D08BFUL, 0xAF768BBCUL, 0xBC267848UL, 0x4E4DFB4BUL,
0x20BD8EDEUL, 0xD2D60DDDUL, 0xC186FE29UL, 0x33ED7D2AUL,
0xE72719C1UL, 0x154C9AC2UL, 0x061C6936UL, 0xF477EA35UL,
0xAA64D611UL, 0x580F5512UL, 0x4B5FA6E6UL, 0xB93425E5UL,
0x6DFE410EUL, 0x9F95C20DUL, 0x8CC531F9UL, 0x7EAE82FAUL,
0x30E349B1UL, 0xC288CAB2UL, 0xD1D83946UL, 0x23B3BA45UL,
0xF779DEAEUL, 0x05125DADUL, 0x1642AE59UL, 0xE4292D5AUL,
0xBA3A117EUL, 0x4851927DUL, 0x5B016189UL, 0xA96AE28AUL,
0x7DA08661UL, 0x8FCB0562UL, 0x9C9BF696UL, 0x6EF07595UL,
0x417B1DBCUL, 0xB3109EBFUL, 0xA0406D4BUL, 0x522BEE48UL,
0x86E18AA3UL, 0x748A09A0UL, 0x67DAFA54UL, 0x95B17957UL,
0xCBA24573UL, 0x39C9C670UL, 0x2A993584UL, 0xD8F2B687UL,

```



```
0x0C38D26CUL, 0xFE53516FUL, 0xED03A29BUL, 0x1F682198UL,  
0x5125DAD3UL, 0xA34E59D0UL, 0xB01EAA24UL, 0x42752927UL,  
0x96BF4DCCUL, 0x64D4CECFUL, 0x77843D3BUL, 0x85EFBE38UL,  
0xDBFC821CUL, 0x2997011FUL, 0x3AC7F2EBUL, 0xC8AC71E8UL,  
0x1C661503UL, 0xEE0D9600UL, 0xFD5D65F4UL, 0x0F36E6F7UL,  
0x61C69362UL, 0x93AD1061UL, 0x80FDE395UL, 0x72966096UL,  
0xA65C047DUL, 0x5437877EUL, 0x4767748AUL, 0xB50CF789UL,  
0xEB1FCBADUL, 0x197448AEUL, 0x0A24BB5AUL, 0xF84F3859UL,  
0x2C855CB2UL, 0xDEEEDFB1UL, 0xCDBE2C45UL, 0x3FD5AF46UL,  
0x7198540DUL, 0x83F3D70EUL, 0x90A324FAUL, 0x62C8A7F9UL,  
0xB602C312UL, 0x44694011UL, 0x5739B3E5UL, 0xA55230E6UL,  
0xFB410CC2UL, 0x092A8FC1UL, 0x1A7A7C35UL, 0xE811FF36UL,  
0x3CDB9BDDUL, 0xCEB018DEUL, 0xDDE0EB2AUL, 0x2F8B6829UL,  
0x82F63B78UL, 0x709DB87BUL, 0x63CD4B8FUL, 0x91A6C88CUL,  
0x456CAC67UL, 0xB7072F64UL, 0xA457DC90UL, 0x563C5F93UL,  
0x082F63B7UL, 0xFA44E0B4UL, 0xE9141340UL, 0x1B7F9043UL,  
0xCFB5F4A8UL, 0x3DDE77ABUL, 0x2E8E845FUL, 0xDCE5075CUL,  
0x92A8FC17UL, 0x60C37F14UL, 0x73938CE0UL, 0x81F80FE3UL,  
0x55326B08UL, 0xA759E80BUL, 0xB4091BFFUL, 0x466298FCUL,  
0x1871A4D8UL, 0xEA1A27DBUL, 0xF94AD42FUL, 0x0B21572CUL,  
0xDFEB33C7UL, 0x2D80B0C4UL, 0x3ED04330UL, 0xCCBBC033UL,  
0xA24BB5A6UL, 0x502036A5UL, 0x4370C551UL, 0xB11B4652UL,  
0x65D122B9UL, 0x97BAA1BAUL, 0x84EA524EUL, 0x7681D14DUL,  
0x2892ED69UL, 0xDAF96E6AUL, 0xC9A99D9EUL, 0x3BC21E9DUL,  
0xEF087A76UL, 0x1D63F975UL, 0x0E330A81UL, 0xFC588982UL,  
0xB21572C9UL, 0x407EF1CAUL, 0x532E023EUL, 0xA145813DUL,  
0x758FE5D6UL, 0x87E466D5UL, 0x94B49521UL, 0x66DF1622UL,  
0x38CC2A06UL, 0xCA7A905UL, 0xD9F75AF1UL, 0x2B9CD9F2UL,  
0xFF56BD19UL, 0x0D3D3E1AUL, 0x1E6DCDEEUL, 0xEC064EEDUL,  
0xC38D26C4UL, 0x31E6A5C7UL, 0x22B65633UL, 0xD0DDD530UL,  
0x0417B1DBUL, 0xF67C32D8UL, 0xE52CC12CUL, 0x1747422FUL,  
0x49547E0BUL, 0xBB3FFD08UL, 0xA86F0EFCUL, 0x5A048DFFUL,  
0x8ECEE914UL, 0x7CA56A17UL, 0x6FF599E3UL, 0x9D9E1AE0UL,  
0xD3D3E1ABUL, 0x21B862A8UL, 0x32E8915CUL, 0xC083125FUL,  
0x144976B4UL, 0xE622F5B7UL, 0xF5720643UL, 0x07198540UL,  
0x590AB964UL, 0xAB613A67UL, 0xB831C993UL, 0x4A5A4A90UL,  
0x9E902E7BUL, 0x6CFBAD78UL, 0x7FAB5E8CUL, 0x8DC0DD8FUL,  
0xE330A81AUL, 0x115B2B19UL, 0x020BD8EDUL, 0xF0605BEEUL,  
0x24AA3F05UL, 0xD6C1BC06UL, 0xC5914FF2UL, 0x37FACCF1UL,  
0x69E9F0D5UL, 0x9B8273D6UL, 0x88D28022UL, 0x7AB90321UL,  
0xAE7367CAUL, 0x5C18E4C9UL, 0x4F48173DUL, 0xBD23943EUL,  
0xF36E6F75UL, 0x0105EC76UL, 0x12551F82UL, 0xE03E9C81UL,  
0x34F4F86AUL, 0xC69F7B69UL, 0xD5CF889DUL, 0x27A40B9EUL,  
0x79B737BAUL, 0x8BDCB4B9UL, 0x988C474DUL, 0x6AE7C44EUL,  
0xBE2DA0A5UL, 0x4C4623A6UL, 0x5F16D052UL, 0xAD7D5351UL,  
};
```

```
#endif
```

This text has been modified by multiple errata. It includes modifications from Section 3.10. It is in final form, and is not further updated in this document.

Old text: (Appendix C)

```
/* Example of table build routine */

#include <stdio.h>
#include <stdlib.h>

#define OUTPUT_FILE    "crc32cr.h"
#define CRC32C_POLY    0x1EDC6F41L
FILE *tf;
unsigned long
reflect_32 (unsigned long b)
{
    int i;
    unsigned long rw = 0L;

    for (i = 0; i < 32; i++){
        if (b & 1)
            rw |= 1 << (31 - i);
        b >>= 1;
    }
    return (rw);
}

unsigned long
build_crc_table (int index)
{
    int i;
    unsigned long rb;

    rb = reflect_32 (index);

    for (i = 0; i < 8; i++){
        if (rb & 0x80000000L)
            rb = (rb << 1) ^ CRC32C_POLY;
        else
            rb <<= 1;
    }
    return (reflect_32 (rb));
}
```

```

main ()
{
    int i;

    printf ("\nGenerating CRC-32c table file <%s>\n",
        OUTPUT_FILE);
    if ((tf = fopen (OUTPUT_FILE, "w")) == NULL){
        printf ("Unable to open %s\n", OUTPUT_FILE);
        exit (1);
    }
    fprintf (tf, "#ifndef __crc32cr_table_h__\n");
    fprintf (tf, "#define __crc32cr_table_h__\n\n");
    fprintf (tf, "#define CRC32C_POLY 0x%08lX\n",
        CRC32C_POLY);
    fprintf (tf,
        "#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])\n");
    fprintf (tf, "\nunsigned long  crc_c[256] =\n{\n");
    for (i = 0; i < 256; i++){
        fprintf (tf, "0x%08lXL, ", build_crc_table (i));
        if ((i & 3) == 3)
            fprintf (tf, "\n");
    }
    fprintf (tf, "};\n\n#endif\n");

    if (fclose (tf) != 0)
        printf ("Unable to close <%s>." OUTPUT_FILE);
    else
        printf ("\nThe CRC-32c table has been written to <%s>.\n",
            OUTPUT_FILE);
}

```

```

-----
New text: (Appendix B)
-----

```

```

/* Example of table build routine */

#include <stdio.h>
#include <stdlib.h>

#define OUTPUT_FILE    "crc32cr.h"
#define CRC32C_POLY    0x1EDC6F41UL

static FILE *tf;

static uint32_t
reflect_32(uint32_t b)
{

```

```

    int i;
    uint32_t rw = 0UL;

    for (i = 0; i < 32; i++) {
        if (b & 1)
            rw |= 1 << (31 - i);
        b >>= 1;
    }
    return (rw);
}

static uint32_t
build_crc_table(int index)
{
    int i;
    uint32_t rb;

    rb = reflect_32(index);

    for (i = 0; i < 8; i++) {
        if (rb & 0x80000000UL)
            rb = (rb << 1) ^ (uint32_t)CRC32C_POLY;
        else
            rb <<= 1;
    }
    return (reflect_32(rb));
}

int
main (void)
{
    int i;

    printf("\nGenerating CRC-32c table file <%=s>\n",
        OUTPUT_FILE);
    if ((tf = fopen(OUTPUT_FILE, "w")) == NULL) {
        printf ("Unable to open %s\n", OUTPUT_FILE);
        exit (1);
    }
    fprintf(tf, "#ifndef __crc32cr_h__\n");
    fprintf(tf, "#define __crc32cr_h__\n\n");
    fprintf(tf, "#define CRC32C_POLY 0x%08XUL\n",
        (uint32_t)CRC32C_POLY);
    fprintf(tf,
        "#define CRC32C(c,d) (c=(c>>8)^crc_c[(c^(d))&0xFF])\n");
    fprintf(tf, "\nuint32_t crc_c[256] =\n{\n");
    for (i = 0; i < 256; i++) {
        fprintf(tf, "0x%08XUL,", build_crc_table (i));

```

```

        if ((i & 3) == 3)
            fprintf(tf, "\n");
        else
            fprintf(tf, " ");
    }
    fprintf(tf, "};\n\n#endif\n");

    if (fclose (tf) != 0)
        printf("Unable to close <%s>.", OUTPUT_FILE);
    else
        printf("\nThe CRC-32c table has been written to <%s>.\n",
            OUTPUT_FILE);
}

```

This text has been modified by multiple errata. It includes modifications from Section 3.10. It is in final form, and is not further updated in this document.

 Old text: (Appendix C)

```

/* Example of crc insertion */

#include "crc32cr.h"

unsigned long
generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    unsigned long crc32 = ~0L;
    unsigned long result;
    unsigned char byte0,byte1,byte2,byte3;

    for (i = 0; i < length; i++){
        CRC32C(crc32, buffer[i]);
    }

    result = ~crc32;

    /* result now holds the negated polynomial remainder;
     * since the table and algorithm is "reflected" [williams95].
     * That is, result has the same value as if we mapped the message
     * to a polynomial, computed the host-bit-order polynomial
     * remainder, performed final negation, then did an end-for-end
     * bit-reversal.
    */
}

```

```
* Note that a 32-bit bit-reversal is identical to four inplace
* 8-bit reversals followed by an end-for-end byteswap.
* In other words, the bytes of each bit are in the right order,
* but the bytes have been byteswapped. So we now do an explicit
* byteswap. On a little-endian machine, this byteswap and
* the final ntohl cancel out and could be elided.
*/

byte0 = result & 0xff;
byte1 = (result>>8) & 0xff;
byte2 = (result>>16) & 0xff;
byte3 = (result>>24) & 0xff;
crc32 = ((byte0 << 24) |
         (byte1 << 16) |
         (byte2 << 8)  |
         byte3);
return ( crc32 );
}

int
insert_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    unsigned long crc32;
    message = (SCTP_message *) buffer;
    message->common_header.checksum = 0L;
    crc32 = generate_crc32c(buffer,length);
    /* and insert it into the message */
    message->common_header.checksum = htonl(crc32);
    return 1;
}

int
validate_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    unsigned int i;
    unsigned long original_crc32;
    unsigned long crc32 = ~0L;

    /* save and zero checksum */
    message = (SCTP_message *) buffer;
    original_crc32 = ntohl(message->common_header.checksum);
    message->common_header.checksum = 0L;
    crc32 = generate_crc32c(buffer,length);
    return ((original_crc32 == crc32)? 1 : -1);
}
```

```
-----
New text: (Appendix B)
-----

/* Example of crc insertion */

#include "crc32cr.h"

uint32_t
generate_crc32c(unsigned char *buffer, unsigned int length)
{
    unsigned int i;
    uint32_t crc32 = 0xffffffffUL;
    uint32_t result;
    uint8_t byte0, byte1, byte2, byte3;

    for (i = 0; i < length; i++) {
        CRC32C(crc32, buffer[i]);
    }

    result = ~crc32;

    /* result now holds the negated polynomial remainder;
     * since the table and algorithm is "reflected" [williams95].
     * That is, result has the same value as if we mapped the message
     * to a polynomial, computed the host-bit-order polynomial
     * remainder, performed final negation, then did an end-for-end
     * bit-reversal.
     * Note that a 32-bit bit-reversal is identical to four inplace
     * 8-bit reversals followed by an end-for-end byteswap.
     * In other words, the bits of each byte are in the right order,
     * but the bytes have been byteswapped. So we now do an explicit
     * byteswap. On a little-endian machine, this byteswap and
     * the final ntohl cancel out and could be elided.
     */

    byte0 = result & 0xff;
    byte1 = (result>>8) & 0xff;
    byte2 = (result>>16) & 0xff;
    byte3 = (result>>24) & 0xff;
    crc32 = ((byte0 << 24) |
              (byte1 << 16) |
              (byte2 << 8) |
              byte3);
    return (crc32);
}

int
```

```
insert_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    uint32_t crc32;
    message = (SCTP_message *) buffer;
    message->common_header.checksum = 0UL;
    crc32 = generate_crc32c(buffer, length);
    /* and insert it into the message */
    message->common_header.checksum = htonl(crc32);
    return 1;
}

int
validate_crc32(unsigned char *buffer, unsigned int length)
{
    SCTP_message *message;
    unsigned int i;
    uint32_t original_crc32;
    uint32_t crc32;

    /* save and zero checksum */
    message = (SCTP_message *)buffer;
    original_crc32 = ntohl(message->common_header.checksum);
    message->common_header.checksum = 0L;
    crc32 = generate_crc32c(buffer, length);
    return ((original_crc32 == crc32)? 1 : -1);
}
<CODE ENDS>
```

This text has been modified by multiple errata. It includes modifications from Section 3.5 and Section 3.10. It is in final form, and is not further updated in this document.

3.46.3. Solution Description

The code was changed to use platform independent types.

3.47. Clarification of Gap Ack Blocks in SACK Chunks

3.47.1. Description of the Problem

The Gap Ack Blocks in the SACK chunk are intended to be isolated. However, this is not mentioned with normative text.

This issue was reported as part of an Errata for [RFC4960] with Errata ID 5202.

3.47.2. Text Changes to the Document

Old text: (Section 3.3.4)

The SACK also contains zero or more Gap Ack Blocks. Each Gap Ack Block acknowledges a subsequence of TSNs received following a break in the sequence of received TSNs. By definition, all TSNs acknowledged by Gap Ack Blocks are greater than the value of the Cumulative TSN Ack.

New text: (Section 3.3.4)

The SACK also contains zero or more Gap Ack Blocks. Each Gap Ack Block acknowledges a subsequence of TSNs received following a break in the sequence of received TSNs. The Gap Ack Blocks SHOULD be isolated. This means that the TSN just before each Gap Ack Block and the TSN just after each Gap Ack Block has not been received. By definition, all TSNs acknowledged by Gap Ack Blocks are greater than the value of the Cumulative TSN Ack.

This text is in final form, and is not further updated in this document.

Old text: (Section 3.3.4)

Gap Ack Blocks:

These fields contain the Gap Ack Blocks. They are repeated for each Gap Ack Block up to the number of Gap Ack Blocks defined in the Number of Gap Ack Blocks field. All DATA chunks with TSNs greater than or equal to (Cumulative TSN Ack + Gap Ack Block Start) and less than or equal to (Cumulative TSN Ack + Gap Ack Block End) of each Gap Ack Block are assumed to have been received correctly.

New text: (Section 3.3.4)

Gap Ack Blocks:

These fields contain the Gap Ack Blocks. They are repeated for each Gap Ack Block up to the number of Gap Ack Blocks defined in the Number of Gap Ack Blocks field. All DATA chunks with TSNs greater than or equal to (Cumulative TSN Ack + Gap Ack Block Start) and less than or equal to (Cumulative TSN Ack + Gap Ack Block End) of each Gap Ack Block are assumed to have been received correctly. Gap Ack Blocks SHOULD be isolated. That means that the DATA chunks with TSN equal to (Cumulative TSN Ack + Gap Ack Block Start - 1) and (Cumulative TSN Ack + Gap Ack Block End + 1) have not been received.

This text is in final form, and is not further updated in this document.

3.47.3. Solution Description

Normative text describing the intended usage of Gap Ack Blocks has been added.

3.48. Handling of SSN Wrap Arounds

3.48.1. Description of the Problem

The Stream Sequence Number (SSN) is used for preserving the ordering of user messages within each SCTP stream. The SSN is limited to 16 bits. Therefore, multiple wrap arounds of the SSN might happen within the current send window. To allow the receiver to deliver

ordered user messages in the correct sequence, the sender should limit the number of user messages per stream.

3.48.2. Text Changes to the Document

Old text: (Section 6.1)

Note: The data sender SHOULD NOT use a TSN that is more than $2^{31} - 1$ above the beginning TSN of the current send window.

New text: (Section 6.1)

Note: The data sender SHOULD NOT use a TSN that is more than $2^{31} - 1$ above the beginning TSN of the current send window.

Note: For each stream, the data sender SHOULD NOT have more than $2^{16} - 1$ ordered user messages in the current send window.

This text is in final form, and is not further updated in this document.

3.48.3. Solution Description

The data sender is required to limit the number of ordered user messages within the current send window.

3.49. Update RFC 2119 Boilerplate

3.49.1. Description of the Problem

The text to be used to refer to the [RFC2119] terms has been updated by [RFC8174].

3.49.2. Text Changes to the Document

Old text: (Section 2)

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

New text: (Section 2)

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This text is in final form, and is not further updated in this document.

3.49.3. Solution Description

The text has been updated to the one specified in [RFC8174].

3.50. Missed Text Removal

3.50.1. Description of the Problem

When integrating the changes to Section 7.2.4 of [RFC2960] as described in Section 2.8.2 of [RFC4460] some text was not removed and is therefore still in [RFC4960].

3.50.2. Text Changes to the Document

Old text: (Section 7.2.4)

A straightforward implementation of the above keeps a counter for each TSN hole reported by a SACK. The counter increments for each consecutive SACK reporting the TSN hole. After reaching 3 and starting the Fast-Retransmit procedure, the counter resets to 0. Because cwnd in SCTP indirectly bounds the number of outstanding TSN's, the effect of TCP Fast Recovery is achieved automatically with no adjustment to the congestion control window size.

New text: (Section 7.2.4)

This text is in final form, and is not further updated in this document.

3.50.3. Solution Description

The text has finally been removed.

4. IANA Considerations

Section 3.44 of this document updates the port number registry for SCTP to be consistent with [RFC6335]. IANA is requested to review Section 3.44.

IANA is only requested to check if it is OK to make the proposed text change in an upcoming standards track document that updates [RFC4960]. IANA is not asked to perform any other action and this document does not request IANA to make a change to any registry.

5. Security Considerations

This document does not add any security considerations to those given in [RFC4960].

6. Acknowledgments

The authors wish to thank Pontus Andersson, Eric W. Biederman, Cedric Bonnet, Spencer Dawkins, Gorrry Fairhurst, Benjamin Kaduk, Mirja Kuehlewind, Peter Lei, Gyula Marosi, Lionel Morand, Jeff Morriss, Karen E. E. Nielsen, Tom Petch, Kacheong Poon, Julien Pourtet, Irene Ruengeler, Michael Welzl, and Qiaobing Xie for their invaluable comments.

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.

7.2. Informative References

- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.
- [RFC1858] Ziemba, G., Reed, D., and P. Traina, "Security Considerations for IP Fragment Filtering", RFC 1858, DOI 10.17487/RFC1858, October 1995, <<https://www.rfc-editor.org/info/rfc1858>>.
- [RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, DOI 10.17487/RFC2960, October 2000, <<https://www.rfc-editor.org/info/rfc2960>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC4460] Stewart, R., Arias-Rodriguez, I., Poon, K., Caro, A., and M. Tuexen, "Stream Control Transmission Protocol (SCTP) Specification Errata and Issues", RFC 4460, DOI 10.17487/RFC4460, April 2006, <<https://www.rfc-editor.org/info/rfc4460>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/info/rfc5681>>.

- [RFC6096] Tuexen, M. and R. Stewart, "Stream Control Transmission Protocol (SCTP) Chunk Flags Registration", RFC 6096, DOI 10.17487/RFC6096, January 2011, <<https://www.rfc-editor.org/info/rfc6096>>.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <<https://www.rfc-editor.org/info/rfc6298>>.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <<https://www.rfc-editor.org/info/rfc6335>>.
- [RFC7053] Tuexen, M., Ruengeler, I., and R. Stewart, "SACK-IMMEDIATELY Extension for the Stream Control Transmission Protocol", RFC 7053, DOI 10.17487/RFC7053, November 2013, <<https://www.rfc-editor.org/info/rfc7053>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <<https://www.rfc-editor.org/info/rfc8311>>.

Authors' Addresses

Randall R. Stewart
Netflix, Inc.
Chapin, SC 29036
United States

Email: randall@lakerest.net

Michael Tuexen
Muenster University of Applied Sciences
Stegerwaldstrasse 39
48565 Steinfurt
Germany

Email: tuexen@fh-muenster.de

Maksim Proshin
Ericsson
Kistavaegen 25
Stockholm 164 80
Sweden

Email: mproshin@tieto.mera.ru

TSVWG
Internet-Draft
Intended status: Standards Track
Expires: December 20, 2019

V. Roca
B. Teibi
INRIA
June 18, 2019

Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC)
Schemes for FECFRAME
draft-ietf-tsvwg-rlc-fec-scheme-16

Abstract

This document describes two fully-specified Forward Erasure Correction (FEC) Schemes for Sliding Window Random Linear Codes (RLC), one for RLC over the Galois Field (A.K.A. Finite Field) $GF(2)$, a second one for RLC over the Galois Field $GF(2^{8})$, each time with the possibility of controlling the code density. They can protect arbitrary media streams along the lines defined by FECFRAME extended to sliding window FEC codes. These sliding window FEC codes rely on an encoding window that slides over the source symbols, generating new repair symbols whenever needed. Compared to block FEC codes, these sliding window FEC codes offer key advantages with real-time flows in terms of reduced FEC-related latency while often providing improved packet erasure recovery capabilities.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 20, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
1.1.	Limits of Block Codes with Real-Time Flows	4
1.2.	Lower Latency and Better Protection of Real-Time Flows with the Sliding Window RLC Codes	4
1.3.	Small Transmission Overheads with the Sliding Window RLC FEC Scheme	5
1.4.	Document Organization	6
2.	Definitions and Abbreviations	6
3.	Common Procedures	7
3.1.	Codec Parameters	7
3.2.	ADU, ADUI and Source Symbols Mappings	9
3.3.	Encoding Window Management	10
3.4.	Source Symbol Identification	11
3.5.	Pseudo-Random Number Generator (PRNG)	11
3.6.	Coding Coefficients Generation Function	13
3.7.	Finite Fields Operations	15
3.7.1.	Finite Field Definitions	15
3.7.2.	Linear Combination of Source Symbols Computation	15
4.	Sliding Window RLC FEC Scheme over $GF(2^{^8})$ for Arbitrary Packet Flows	16
4.1.	Formats and Codes	16
4.1.1.	FEC Framework Configuration Information	16
4.1.2.	Explicit Source FEC Payload ID	18
4.1.3.	Repair FEC Payload ID	18
4.2.	Procedures	20
5.	Sliding Window RLC FEC Scheme over $GF(2)$ for Arbitrary Packet Flows	20
5.1.	Formats and Codes	20
5.1.1.	FEC Framework Configuration Information	20
5.1.2.	Explicit Source FEC Payload ID	20
5.1.3.	Repair FEC Payload ID	20
5.2.	Procedures	21
6.	FEC Code Specification	21
6.1.	Encoding Side	21
6.2.	Decoding Side	22
7.	Implementation Status	22

8.	Security Considerations	23
8.1.	Attacks Against the Data Flow	23
8.1.1.	Access to Confidential Content	23
8.1.2.	Content Corruption	23
8.2.	Attacks Against the FEC Parameters	23
8.3.	When Several Source Flows are to be Protected Together .	25
8.4.	Baseline Secure FEC Framework Operation	25
8.5.	Additional Security Considerations for Numerical Computations	25
9.	Operations and Management Considerations	26
9.1.	Operational Recommendations: Finite Field GF(2) Versus GF(2 ⁸)	26
9.2.	Operational Recommendations: Coding Coefficients Density Threshold	26
10.	IANA Considerations	27
11.	Acknowledgments	27
12.	References	27
12.1.	Normative References	27
12.2.	Informative References	28
Appendix A.	TinyMT32 Validation Criteria (Normative)	30
Appendix B.	Assessing the PRNG Adequacy (Informational)	31
Appendix C.	Possible Parameter Derivation (Informational)	33
C.1.	Case of a CBR Real-Time Flow	34
C.2.	Other Types of Real-Time Flow	36
C.3.	Case of a Non Real-Time Flow	37
Appendix D.	Decoding Beyond Maximum Latency Optimization (Informational)	37
Authors' Addresses	38

1. Introduction

Application-Level Forward Erasure Correction (AL-FEC) codes, or simply FEC codes, are a key element of communication systems. They are used to recover from packet losses (or erasures) during content delivery sessions to a potentially large number of receivers (multicast/broadcast transmissions). This is the case with the FLUTE/ALC protocol [RFC6726] when used for reliable file transfers over lossy networks, and the FECFRAME protocol [RFC6363] when used for reliable continuous media transfers over lossy networks.

The present document only focuses on the FECFRAME protocol, used in multicast/broadcast delivery mode, in particular for contents that feature stringent real-time constraints: each source packet has a maximum validity period after which it will not be considered by the destination application.

1.1. Limits of Block Codes with Real-Time Flows

With FECFRAME, there is a single FEC encoding point (either an end-host/server (source) or a middlebox) and a single FEC decoding point per receiver (either an end-host (receiver) or middlebox). In this context, currently standardized AL-FEC codes for FECFRAME like Reed-Solomon [RFC6865], LDPC-Staircase [RFC6816], or Raptor/RaptorQ [RFC6681], are all linear block codes: they require the data flow to be segmented into blocks of a predefined maximum size.

To define this block size, it is required to find an appropriate balance between robustness and decoding latency: the larger the block size, the higher the robustness (e.g., in case of long packet erasure bursts), but also the higher the maximum decoding latency (i.e., the maximum time required to recover a lost (erased) packet thanks to FEC protection). Therefore, with a multicast/broadcast session where different receivers experience different packet loss rates, the block size should be chosen by considering the worst communication conditions one wants to support, but without exceeding the desired maximum decoding latency. This choice then impacts the FEC-related latency of all receivers, even those experiencing a good communication quality, since no FEC encoding can happen until all the source data of the block is available at the sender, which directly depends on the block size.

1.2. Lower Latency and Better Protection of Real-Time Flows with the Sliding Window RLC Codes

This document introduces two fully-specified FEC Schemes that do not follow the block code approach: the Sliding Window Random Linear Codes (RLC) over either Galois Fields (A.K.A. Finite Fields) $GF(2)$ (the "binary case") or $GF(2^{8})$, 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:

- o the Encoding Symbol Identifier (ESI) of the first source symbol in the encoding window as well as the number of symbols (since this number may vary with a variable size, elastic window). These two pieces of information enable each receiver to reconstruct the set of source symbols considered during encoding, the only constraint being that there cannot be any gap;
- o the seed and density threshold parameters used by a coding coefficients generation function (Section 3.6). These two pieces

of information enable each receiver to generate the same set of coding coefficients over $GF(2^m)$ as the sender;

Therefore, no matter the number of source symbols present in the encoding window, each FEC Repair Packet features a fixed 64-bit long header, called Repair FEC Payload ID (Figure 8). Similarly, each FEC Source Packet features a fixed 32-bit long trailer, called Explicit Source FEC Payload ID (Figure 6), that contains the ESI of the first source symbol (Section 3.2).

1.4. Document Organization

This fully-specified FEC Scheme follows the structure required by [RFC6363], section 5.6. "FEC Scheme Requirements", namely:

3. Procedures: This section describes procedures specific to this FEC Scheme, namely: RLC parameters derivation, ADUI and source symbols mapping, pseudo-random number generator, and coding coefficients generation function;
4. Formats and Codes: This section defines the Source FEC Payload ID and Repair FEC Payload ID formats, carrying the signaling information associated to each source or repair symbol. It also defines the FEC Framework Configuration Information (FFCI) carrying signaling information for the session;
5. FEC Code Specification: Finally this section provides the code specification.

2. Definitions and Abbreviations

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document uses the following definitions and abbreviations:

a^b a to the power of b

$GF(q)$ denotes a finite field (also known as the Galois Field) with q elements. We assume that $q = 2^m$ in this document

m defines the length of the elements in the finite field, in bits.

In this document, m is equal to 1 or 8

ADU: Application Data Unit

ADUI: Application Data Unit Information (includes the F , L and padding fields in addition to the ADU)

E : size of an encoding symbol (i.e., source or repair symbol), assumed fixed (in bytes)

br_in: transmission bitrate at the input of the FECFRAME sender, assumed fixed (in bits/s)
br_out: transmission bitrate at the output of the FECFRAME sender, assumed fixed (in bits/s)
max_lat: maximum FEC-related latency within FECFRAME (a decimal number expressed in seconds)
cr: RLC coding rate, ratio between the total number of source symbols and the total number of source plus repair symbols
ew_size: encoding window current size at a sender (in symbols)
ew_max_size: encoding window maximum size at a sender (in symbols)
dw_max_size: decoding window maximum size at a receiver (in symbols)
ls_max_size: linear system maximum size (or width) at a receiver (in symbols)
WSR: window size ratio parameter used to derive ew_max_size (encoder) and ls_max_size (decoder).
PRNG: pseudo-random number generator
TinyMT32: PRNG used in this specification.
DT: coding coefficients density threshold, an integer between 0 and 15 (inclusive) the controls the fraction of coefficients that are non zero

3. Common Procedures

This section introduces the procedures that are used by these FEC Schemes.

3.1. Codec Parameters

A codec implementing the Sliding Window RLC FEC Scheme relies on several parameters:

Maximum FEC-related latency budget, max_lat (a decimal number expressed in seconds) with real-time flows:
a source ADU flow can have real-time constraints, and therefore any FECFRAME related operation should take place within the validity period of each ADU (Appendix D describes an exception to this rule). When there are multiple flows with different real-time constraints, we consider the most stringent constraints (see [RFC6363], Section 10.2, item 6, for recommendations when several flows are globally protected). The maximum FEC-related latency budget, max_lat, accounts for all sources of latency added by FEC encoding (at a sender) and FEC decoding (at a receiver). Other sources of latency (e.g., added by network communications) are out of scope and must be considered separately (said differently, they have already been deducted from max_lat). max_lat can be regarded as the latency budget permitted for all FEC-related operations. This is an input parameter that enables a FECFRAME sender to derive other internal parameters (see Appendix C);

Encoding window current (resp. maximum) size, `ew_size` (resp. `ew_max_size`) (in symbols):

at a FECFRAME sender, during FEC encoding, a repair symbol is computed as a linear combination of the `ew_size` source symbols present in the encoding window. The `ew_max_size` is the maximum size of this window, while `ew_size` is the current size. For example, in the common case at session start, upon receiving new source ADUs, the `ew_size` progressively increases until it reaches its maximum value, `ew_max_size`. We have:

$$0 < \text{ew_size} \leq \text{ew_max_size}$$

Decoding window maximum size, `dw_max_size` (in symbols): at a FECFRAME receiver, `dw_max_size` is the maximum number of received or lost source symbols that are still within their latency budget;

Linear system maximum size, `ls_max_size` (in symbols): at a FECFRAME receiver, the linear system maximum size, `ls_max_size`, is the maximum number of received or lost source symbols in the linear system (i.e., the variables). It SHOULD NOT be smaller than `dw_max_size` since it would mean that, even after receiving a sufficient number of FEC Repair Packets, a lost ADU may not be recovered just because the associated source symbols have been prematurely removed from the linear system, which is usually counter-productive. On the opposite, the linear system MAY grow beyond the `dw_max_size` (Appendix D);

Symbol size, `E` (in bytes): the `E` parameter determines the source and repair symbol sizes (necessarily equal). This is an input parameter that enables a FECFRAME sender to derive other internal parameters, as explained below. An implementation at a sender MUST fix the `E` parameter and MUST communicate it as part of the FEC Scheme-Specific Information (Section 4.1.1.2).

Code rate, `cr`: The code rate parameter determines the amount of redundancy added to the flow. More precisely the `cr` is the ratio between the total number of source symbols and the total number of source plus repair symbols and by definition: $0 < \text{cr} \leq 1$. This is an input parameter that enables a FECFRAME sender to derive other internal parameters, as explained below. However, there is no need to communicate the `cr` parameter per se (it's not required to process a repair symbol at a receiver). This code rate parameter can be static. However, in specific use-cases (e.g., with unicast transmissions in presence of a feedback mechanism that estimates the communication quality, out of scope of FECFRAME), the code rate may be adjusted dynamically.

Appendix C proposes non normative techniques to derive those parameters, depending on the use-case specificities.

3.2. ADU, ADUI and Source Symbols Mappings

At a sender, an ADU coming from the application is not directly mapped to source symbols. When multiple source flows (e.g., media streams) are mapped onto the same FECFRAME instance, each flow is assigned its own Flow ID value (see below). This Flow ID is then prepended to each ADU before FEC encoding. This way, FEC decoding at a receiver also recovers this Flow ID and the recovered ADU can be assigned to the right source flow (note that the 5-tuple used to identify the right source flow of a received ADU is absent with a recovered ADU since it is not FEC protected).

Additionally, since ADUs are of variable size, padding is needed so that each ADU (with its flow identifier) contribute to an integral number of source symbols. This requires adding the original ADU length to each ADU before doing FEC encoding. Because of these requirements, an intermediate format, the ADUI, or ADU Information, is considered [RFC6363].

For each incoming ADU, an ADUI MUST be created as follows. First of all, 3 bytes are prepended (Figure 1):

Flow ID (F) (8-bit field): this unsigned byte contains the integer identifier associated to the source ADU flow to which this ADU belongs. It is assumed that a single byte is sufficient, which implies that no more than 256 flows will be protected by a single FECFRAME session instance.

Length (L) (16-bit field): this unsigned integer contains the length of this ADU, in network byte order (i.e., big endian). This length is for the ADU itself and does not include the F, L, or Pad fields.

Then, zero padding is added to the ADU if needed:

Padding (Pad) (variable size field): this field contains zero padding to align the F, L, ADU and padding up to a size that is multiple of E bytes (i.e., the source and repair symbol length).

The data unit resulting from the ADU and the F, L, and Pad fields is called ADUI. Since ADUs can have different sizes, this is also the case for ADUIs. However, an ADUI always contributes to an integral number of source symbols.

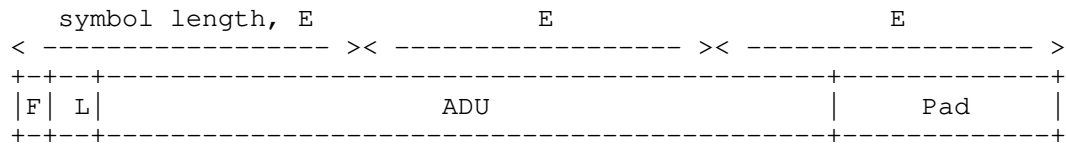


Figure 1: ADUI Creation example (here 3 source symbols are created for this ADUI).

Note that neither the initial 3 bytes nor the optional padding are sent over the network. However, they are considered during FEC encoding, and a receiver who lost a certain FEC Source Packet (e.g., the UDP datagram containing this FEC Source Packet when UDP is used as the transport protocol) will be able to recover the ADUI if FEC decoding succeeds. Thanks to the initial 3 bytes, this receiver will get rid of the padding (if any) and identify the corresponding ADU flow.

3.3. Encoding Window Management

Source symbols and the corresponding ADUs are removed from the encoding window:

- o when the sliding encoding window has reached its maximum size, `ew_max_size`. In that case the oldest symbol MUST be removed before adding a new symbol, so that the current encoding window size always remains inferior or equal to the maximum size: `ew_size <= ew_max_size`;
- o when an ADU has reached its maximum validity duration in case of a real-time flow. When this happens, all source symbols corresponding to the ADUI that expired SHOULD be removed from the encoding window;

Source symbols are added to the sliding encoding window each time a new ADU arrives, once the ADU-to-source symbols mapping has been performed (Section 3.2). The current size of the encoding window, `ew_size`, is updated after adding new source symbols. This process may require to remove old source symbols so that: `ew_size <= ew_max_size`.

Note that a FEC codec may feature practical limits in the number of source symbols in the encoding window (e.g., for computational complexity reasons). This factor may further limit the `ew_max_size` value, in addition to the maximum FEC-related latency budget (Section 3.1).

3.4. Source Symbol Identification

Each source symbol is identified by an Encoding Symbol ID (ESI), an unsigned integer. The ESI of source symbols MUST start with value 0 for the first source symbol and MUST be managed sequentially. Wrapping to zero happens after reaching the maximum value made possible by the ESI field size (this maximum value is FEC Scheme dependant, for instance, $2^{32}-1$ with FEC Schemes XXX and YYY).

No such consideration applies to repair symbols.

3.5. Pseudo-Random Number Generator (PRNG)

In order to compute coding coefficients (see Section 3.6), the RLC FEC Schemes rely on the TinyMT32 PRNG defined in [tinymt32] with two additional functions defined in this section.

This PRNG MUST first be initialized with a 32-bit unsigned integer, used as a seed, with:

```
void tinymt32_init (tinymt32_t * s, uint32_t seed);
```

With the FEC Schemes defined in this document, the seed is in practice restricted to a value between 0 and 0xFFFF inclusive (note that this PRNG accepts a seed value equal to 0), since this is the Repair_Key 16-bit field value of the Repair FEC Payload ID (Section 4.1.3). In practice, how to manage the seed and Repair_Key values (both are equal) is left to the implementer, using a monotonically increasing counter being one possibility (Section 6.1). In addition to the seed, this function takes as parameter a pointer to an instance of a tinymt32_t structure that is used to keep the internal state of the PRNG.

Then, each time a new pseudo-random integer between 0 and 15 inclusive (4-bit pseudo-random integer) is needed, the following function is used:

```
uint32_t tinymt32_rand16 (tinymt32_t * s);
```

This function takes as parameter a pointer to the same tinymt32_t structure (that is left unchanged between successive calls to the function).

Similarly, each time a new pseudo-random integer between 0 and 255 inclusive (8-bit pseudo-random integer) is needed, the following function is used:

```
uint32_t tinymt32_rand256 (tinymt32_t * s);
```

These two functions keep respectively the 4 or 8 less significant bits of the 32-bit pseudo-random number generated by the `tinymt32_generate_uint32()` function of [tinymt32]. This is done by computing the result of a binary AND between the `tinymt32_generate_uint32()` output and respectively the 0xF or 0xFF constants, using 32-bit unsigned integer operations. Figure 2 shows a possible implementation. This is a C language implementation, written for C99 [C99]. Test results discussed in Appendix B show that this simple technique, applied to this PRNG, is in line with the RLC FEC Schemes needs.

```
<CODE BEGINS>
/**
 * This function outputs a pseudo-random integer in [0 .. 15] range.
 *
 * @param s      pointer to tinymt internal state.
 * @return       unsigned integer between 0 and 15 inclusive.
 */
uint32_t tinymt32_rand16(tinymt32_t *s)
{
    return (tinymt32_generate_uint32(s) & 0xF);
}

/**
 * This function outputs a pseudo-random integer in [0 .. 255] range.
 *
 * @param s      pointer to tinymt internal state.
 * @return       unsigned integer between 0 and 255 inclusive.
 */
uint32_t tinymt32_rand256(tinymt32_t *s)
{
    return (tinymt32_generate_uint32(s) & 0xFF);
}
<CODE ENDS>
```

Figure 2: 4-bit and 8-bit mapping functions for TinyMT32

Any implementation of this PRNG MUST have the same output as that provided by the reference implementation of [tinymt32]. In order to increase the compliancy confidence, three criteria are proposed: the one described in [tinymt32] (for the TinyMT32 32-bit unsigned integer generator), and the two others detailed in Appendix A (for the mapping to 4-bit and 8-bit intervals). Because of the way the mapping functions work, it is unlikely that an implementation that fulfills the first criterion fails to fulfill the two others.

3.6. Coding Coefficients Generation Function

The coding coefficients, used during the encoding process, are generated at the RLC encoder by the `generate_coding_coefficients()` function each time a new repair symbol needs to be produced. The fraction of coefficients that are non zero (i.e., the density) is controlled by the DT (Density Threshold) parameter. DT has values between 0 (the minimum value) and 15 (the maximum value), and the average probability of having a non zero coefficient equals $(DT + 1) / 16$. In particular, when DT equals 15 the function guaranties that all coefficients are non zero (i.e., maximum density).

These considerations apply to both the RLC over GF(2) and RLC over GF($2^{^8}$), the only difference being the value of the m parameter. With the RLC over GF(2) FEC Scheme (Section 5), m is equal to 1. With RLC over GF($2^{^8}$) FEC Scheme (Section 4), m is equal to 8.

Figure 3 shows the reference `generate_coding_coefficients()` implementation. This is a C language implementation, written for C99 [C99].

```
<CODE BEGINS>
#include <string.h>

/*
 * Fills in the table of coding coefficients (of the right size)
 * provided with the appropriate number of coding coefficients to
 * use for the repair symbol key provided.
 *
 * (in) repair_key    key associated to this repair symbol. This
 *                    parameter is ignored (useless) if m=1 and dt=15
 * (in/out) cc_tab    pointer to a table of the right size to store
 *                    coding coefficients. All coefficients are
 *                    stored as bytes, regardless of the m parameter,
 *                    upon return of this function.
 * (in) cc_nb         number of entries in the cc_tab table. This
 *                    value is equal to the current encoding window
 *                    size.
 * (in) dt            integer between 0 and 15 (inclusive) that
 *                    controls the density. With value 15, all
 *                    coefficients are guaranteed to be non zero
 *                    (i.e. equal to 1 with GF(2) and equal to a
 *                    value in {1,... 255} with GF( $2^{^8}$ )), otherwise
 *                    a fraction of them will be 0.
 * (in) m             Finite Field GF( $2^{^m}$ ) parameter. In this
 *                    document only values 1 and 8 are considered.
 * (out)              returns 0 in case of success, an error code
 *                    different than 0 otherwise.
```

```
*/
int generate_coding_coefficients (uint16_t  repair_key,
                                uint8_t*   cc_tab,
                                uint16_t   cc_nb,
                                uint8_t    dt,
                                uint8_t    m)
{
    uint32_t    i;
    tinymt32_t  s;    /* PRNG internal state */

    if (dt > 15) {
        return -1; /* error, bad dt parameter */
    }
    switch (m) {
    case 1:
        if (dt == 15) {
            /* all coefficients are 1 */
            memset(cc_tab, 1, cc_nb);
        } else {
            /* here coefficients are either 0 or 1 */
            tinymt32_init(&s, repair_key);
            for (i = 0 ; i < cc_nb ; i++) {
                cc_tab[i] = (tinymt32_rand16(&s) <= dt) ? 1 : 0;
            }
        }
        break;

    case 8:
        tinymt32_init(&s, repair_key);
        if (dt == 15) {
            /* coefficient 0 is avoided here in order to include
             * all the source symbols */
            for (i = 0 ; i < cc_nb ; i++) {
                do {
                    cc_tab[i] = (uint8_t) tinymt32_rand256(&s);
                } while (cc_tab[i] == 0);
            }
        } else {
            /* here a certain number of coefficients should be 0 */
            for (i = 0 ; i < cc_nb ; i++) {
                if (tinymt32_rand16(&s) <= dt) {
                    do {
                        cc_tab[i] = (uint8_t) tinymt32_rand256(&s);
                    } while (cc_tab[i] == 0);
                } else {
                    cc_tab[i] = 0;
                }
            }
        }
    }
}
```

```

    }
    break;

default:
    return -2; /* error, bad parameter m */
}
return 0; /* success */
}
<CODE ENDS>

```

Figure 3: Coding Coefficients Generation Function Reference Implementation

3.7. Finite Fields Operations

3.7.1. Finite Field Definitions

The two RLC FEC Schemes specified in this document reuse the Finite Fields defined in [RFC5510], section 8.1. More specifically, the elements of the field $GF(2^m)$ are represented by polynomials with binary coefficients (i.e., over $GF(2)$) and degree lower or equal to $m-1$. The addition between two elements is defined as the addition of binary polynomials in $GF(2)$, which is equivalent to a bitwise XOR operation on the binary representation of these elements.

With $GF(2^8)$, multiplication between two elements is the multiplication modulo a given irreducible polynomial of degree 8. The following irreducible polynomial is used for $GF(2^8)$:

$$x^8 + x^4 + x^3 + x^2 + 1$$

With $GF(2)$, multiplication corresponds to a logical AND operation.

3.7.2. Linear Combination of Source Symbols Computation

The two RLC FEC Schemes require the computation of a linear combination of source symbols, using the coding coefficients produced by the `generate_coding_coefficients()` function and stored in the `cc_tab[]` array.

With the RLC over $GF(2^8)$ FEC Scheme, a linear combination of the `ew_size` source symbol present in the encoding window, say `src_0` to `src_ew_size_1`, in order to generate a repair symbol, is computed as follows. For each byte of position `i` in each source and the repair symbol, where `i` belongs to `[0; E-1]`, compute:

$$\text{repair}[i] = \text{cc_tab}[0] * \text{src_0}[i] \text{ XOR } \text{cc_tab}[1] * \text{src_1}[i] \text{ XOR } \dots \text{ XOR } \text{cc_tab}[\text{ew_size} - 1] * \text{src_ew_size_1}[i]$$

where $*$ is the multiplication over $GF(2^{^8})$. In practice various optimizations need to be used in order to make this computation efficient (see in particular [PGM13]).

With the RLC over $GF(2)$ FEC Scheme (binary case), a linear combination is computed as follows. The repair symbol is the XOR sum of all the source symbols corresponding to a coding coefficient $cc_tab[j]$ equal to 1 (i.e., the source symbols corresponding to zero coding coefficients are ignored). The XOR sum of the byte of position i in each source is computed and stored in the corresponding byte of the repair symbol, where i belongs to $[0; E-1]$. In practice, the XOR sums will be computed several bytes at a time (e.g., on 64 bit words, or on arrays of 16 or more bytes when using SIMD CPU extensions).

With both FEC Schemes, the details of how to optimize the computation of these linear combinations are of high practical importance but out of scope of this document.

4. Sliding Window RLC FEC Scheme over $GF(2^{^8})$ for Arbitrary Packet Flows

This fully-specified FEC Scheme defines the Sliding Window Random Linear Codes (RLC) over $GF(2^{^8})$.

4.1. Formats and Codes

4.1.1. FEC Framework Configuration Information

Following the guidelines of [RFC6363], section 5.6, this section provides the FEC Framework Configuration Information (or FFCI). This FFCI needs to be shared (e.g., using SDP) between the FECFRAME sender and receiver instances in order to synchronize them. It includes a FEC Encoding ID, mandatory for any FEC Scheme specification, plus scheme-specific elements.

4.1.1.1. FEC Encoding ID

- o FEC Encoding ID: the value assigned to this fully specified FEC Scheme MUST be XXXX, as assigned by IANA (Section 10).

When SDP is used to communicate the FFCI, this FEC Encoding ID is carried in the 'encoding-id' parameter.

4.1.1.2. FEC Scheme-Specific Information

The FEC Scheme-Specific Information (FSSI) includes elements that are specific to the present FEC Scheme. More precisely:

Encoding symbol size (E): a non-negative integer that indicates the size of each encoding symbol in bytes;

Window Size Ratio (WSR) parameter: a non-negative integer between 0 and 255 (both inclusive) used to initialize window sizes. A value of 0 indicates this parameter is not considered (e.g., a fixed encoding window size may be chosen). A value between 1 and 255 inclusive is required by certain of the parameter derivation techniques described in Appendix C;

This element is required both by the sender (RLC encoder) and the receiver(s) (RLC decoder).

When SDP is used to communicate the FFCI, this FEC Scheme-specific information is carried in the 'fssi' parameter in textual representation as specified in [RFC6364]. For instance:

```
fssi=E:1400,WSR:191
```

In that case the name values "E" and "WSR" are used to convey the E and WSR parameters respectively.

If another mechanism requires the FSSI to be carried as an opaque octet string, the encoding format consists of the following three octets, where the E field is carried in "big-endian" or "network order" format, that is, most significant byte first:

Encoding symbol length (E): 16-bit field;
Window Size Ratio Parameter (WSR): 8-bit field.

These three octets can be communicated as such, or for instance, be subject to an additional Base64 encoding.

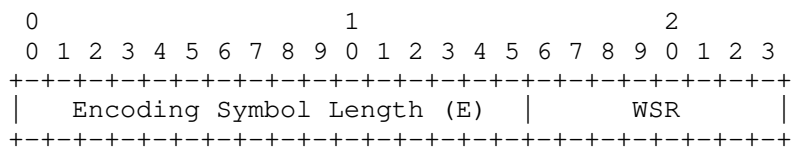


Figure 4: FSSI Encoding Format

4.1.2. Explicit Source FEC Payload ID

A FEC Source Packet MUST contain an Explicit Source FEC Payload ID that is appended to the end of the packet as illustrated in Figure 5.

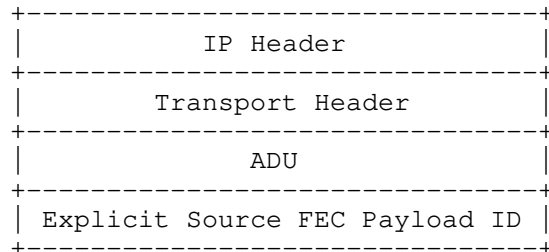


Figure 5: Structure of an FEC Source Packet with the Explicit Source FEC Payload ID

More precisely, the Explicit Source FEC Payload ID is composed of the following field, carried in "big-endian" or "network order" format, that is, most significant byte first (Figure 6):

Encoding Symbol ID (ESI) (32-bit field): this unsigned integer identifies the first source symbol of the ADUI corresponding to this FEC Source Packet. The ESI is incremented for each new source symbol, and after reaching the maximum value ($2^{32}-1$), wrapping to zero occurs.

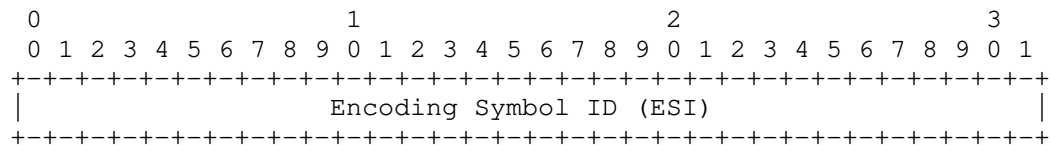


Figure 6: Source FEC Payload ID Encoding Format

4.1.3. Repair FEC Payload ID

A FEC Repair Packet MAY contain one or more repair symbols. When there are several repair symbols, all of them MUST have been generated from the same encoding window, using Repair_Key values that are managed as explained below. A receiver can easily deduce the number of repair symbols within a FEC Repair Packet by comparing the received FEC Repair Packet size (equal to the UDP payload size when UDP is the underlying transport protocol) and the symbol size, E, communicated in the FFCI.

A FEC Repair Packet MUST contain a Repair FEC Payload ID that is prepended to the repair symbol as illustrated in Figure 7.

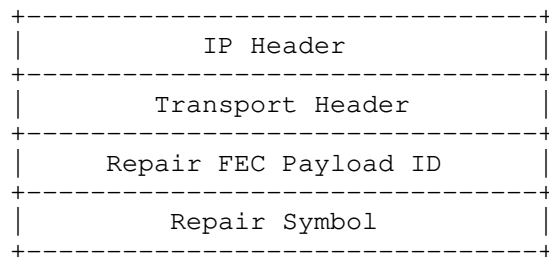


Figure 7: Structure of an FEC Repair Packet with the Repair FEC Payload ID

More precisely, the Repair FEC Payload ID is composed of the following fields where all integer fields are carried in "big-endian" or "network order" format, that is, most significant byte first (Figure 8):

Repair_Key (16-bit field): this unsigned integer is used as a seed by the coefficient generation function (Section 3.6) in order to generate the desired number of coding coefficients. This repair key may be a monotonically increasing integer value that loops back to 0 after reaching 65535 (see Section 6.1). When a FEC Repair Packet contains several repair symbols, this repair key value is that of the first repair symbol. The remaining repair keys can be deduced by incrementing by 1 this value, up to a maximum value of 65535 after which it loops back to 0.

Density Threshold for the coding coefficients, DT (4-bit field): this unsigned integer carries the Density Threshold (DT) used by the coding coefficient generation function Section 3.6. More precisely, it controls the probability of having a non zero coding coefficient, which equals $(DT+1) / 16$. When a FEC Repair Packet contains several repair symbols, the DT value applies to all of them;

Number of Source Symbols in the encoding window, NSS (12-bit field):

this unsigned integer indicates the number of source symbols in the encoding window when this repair symbol was generated. When a FEC Repair Packet contains several repair symbols, this NSS value applies to all of them;

ESI of First Source Symbol in the encoding window, FSS_ESI (32-bit field):

this unsigned integer indicates the ESI of the first source symbol in the encoding window when this repair symbol was generated.

When a FEC Repair Packet contains several repair symbols, this FSS_ESI value applies to all of them;

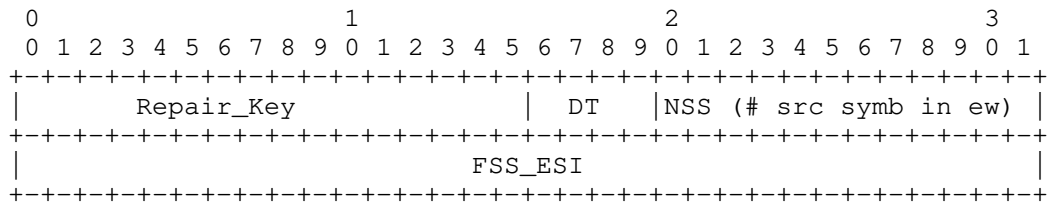


Figure 8: Repair FEC Payload ID Encoding Format

4.2. Procedures

All the procedures of Section 3 apply to this FEC Scheme.

5. Sliding Window RLC FEC Scheme over GF(2) for Arbitrary Packet Flows

This fully-specified FEC Scheme defines the Sliding Window Random Linear Codes (RLC) over GF(2) (binary case).

5.1. Formats and Codes

5.1.1. FEC Framework Configuration Information

5.1.1.1. FEC Encoding ID

- o FEC Encoding ID: the value assigned to this fully specified FEC Scheme MUST be YYYY, as assigned by IANA (Section 10).

When SDP is used to communicate the FFCI, this FEC Encoding ID is carried in the 'encoding-id' parameter.

5.1.1.2. FEC Scheme-Specific Information

All the considerations of Section 4.1.1.2 apply here.

5.1.2. Explicit Source FEC Payload ID

All the considerations of Section 4.1.2 apply here.

5.1.3. Repair FEC Payload ID

All the considerations of Section 4.1.3 apply here, with the only exception that the Repair_Key field is useless if DT = 15 (indeed, in that case all the coefficients are necessarily equal to 1 and the coefficient generation function does not use any PRNG). When DT = 15

the FECFRAME sender MUST set the Repair_Key field to zero on transmission and a receiver MUST ignore it on receipt.

5.2. Procedures

All the procedures of Section 3 apply to this FEC Scheme.

6. FEC Code Specification

6.1. Encoding Side

This section provides a high level description of a Sliding Window RLC encoder.

Whenever a new FEC Repair Packet is needed, the RLC encoder instance first gathers the ew_size source symbols currently in the sliding encoding window. Then it chooses a repair key, which can be a monotonically increasing integer value, incremented for each repair symbol up to a maximum value of 65535 (as it is carried within a 16-bit field) after which it loops back to 0. This repair key is communicated to the coefficient generation function (Section 3.6) in order to generate ew_size coding coefficients. Finally, the FECFRAME sender computes the repair symbol as a linear combination of the ew_size source symbols using the ew_size coding coefficients (Section 3.7). When E is small and when there is an incentive to pack several repair symbols within the same FEC Repair Packet, the appropriate number of repair symbols are computed. In that case the repair key for each of them MUST be incremented by 1, keeping the same ew_size source symbols, since only the first repair key will be carried in the Repair FEC Payload ID. The FEC Repair Packet can then be passed to the transport layer for transmission. The source versus repair FEC packet transmission order is out of scope of this document and several approaches exist that are implementation-specific.

Other solutions are possible to select a repair key value when a new FEC Repair Packet is needed, for instance, by choosing a random integer between 0 and 65535. However, selecting the same repair key as before (which may happen in case of a random process) is only meaningful if the encoding window has changed, otherwise the same FEC Repair Packet will be generated. In any case, choosing the repair key is entirely at the discretion of the sender, since it is communicated to the receiver(s) in each Repair FEC Payload ID. A receiver should not make any assumption on the way the repair key is managed.

6.2. Decoding Side

This section provides a high level description of a Sliding Window RLC decoder.

A FECFRAME receiver needs to maintain a linear system whose variables are the received and lost source symbols. Upon receiving a FEC Repair Packet, a receiver first extracts all the repair symbols it contains (in case several repair symbols are packed together). For each repair symbol, when at least one of the corresponding source symbols it protects has been lost, the receiver adds an equation to the linear system (or no equation if this repair packet does not change the linear system rank). This equation of course re-uses the `ew_size` coding coefficients that are computed by the same coefficient generation function (Section 3.6), using the repair key and encoding window descriptions carried in the Repair FEC Payload ID. Whenever possible (i.e., when a sub-system covering one or more lost source symbols is of full rank), decoding is performed in order to recover lost source symbols. Gaussian elimination is one possible algorithm to solve this linear system. Each time an ADUI can be totally recovered, padding is removed (thanks to the Length field, `L`, of the ADUI) and the ADU is assigned to the corresponding application flow (thanks to the Flow ID field, `F`, of the ADUI). This ADU is finally passed to the corresponding upper application. Received FEC Source Packets, containing an ADU, MAY be passed to the application either immediately or after some time to guaranty an ordered delivery to the application. This document does not mandate any approach as this is an operational and management decision.

With real-time flows, a lost ADU that is decoded after the maximum latency or an ADU received after this delay has no value to the application. This raises the question of deciding whether or not an ADU is late. This decision MAY be taken within the FECFRAME receiver (e.g., using the decoding window, see Section 3.1) or within the application (e.g., using RTP timestamps within the ADU). Deciding which option to follow and whether or not to pass all ADUs, including those assumed late, to the application are operational decisions that depend on the application and are therefore out of scope of this document. Additionally, Appendix D discusses a backward compatible optimization whereby late source symbols MAY still be used within the FECFRAME receiver in order to improve transmission robustness.

7. Implementation Status

Editor's notes: RFC Editor, please remove this section motivated by RFC 6982 before publishing the RFC. Thanks.

An implementation of the Sliding Window RLC FEC Scheme for FECFRAME exists:

- o Organisation: Inria
- o Description: This is an implementation of the Sliding Window RLC FEC Scheme limited to $GF(2^{8})$. It relies on a modified version of our OpenFEC (<http://openfec.org>) FEC code library. It is integrated in our FECFRAME software (see [fecframe-ext]).
- o Maturity: prototype.
- o Coverage: this software complies with the Sliding Window RLC FEC Scheme.
- o Licensing: proprietary.
- o Contact: vincent.roca@inria.fr

8. Security Considerations

The FEC Framework document [RFC6363] provides a fairly comprehensive analysis of security considerations applicable to FEC Schemes. Therefore, the present section follows the security considerations section of [RFC6363] and only discusses specific topics.

8.1. Attacks Against the Data Flow

8.1.1. Access to Confidential Content

The Sliding Window RLC FEC Scheme specified in this document does not change the recommendations of [RFC6363]. To summarize, if confidentiality is a concern, it is RECOMMENDED that one of the solutions mentioned in [RFC6363] is used with special considerations to the way this solution is applied (e.g., is encryption applied before or after FEC protection, within the end-system or in a middlebox), to the operational constraints (e.g., performing FEC decoding in a protected environment may be complicated or even impossible) and to the threat model.

8.1.2. Content Corruption

The Sliding Window RLC FEC Scheme specified in this document does not change the recommendations of [RFC6363]. To summarize, it is RECOMMENDED that one of the solutions mentioned in [RFC6363] is used on both the FEC Source and Repair Packets.

8.2. Attacks Against the FEC Parameters

The FEC Scheme specified in this document defines parameters that can be the basis of attacks. More specifically, the following parameters of the FFCI may be modified by an attacker who targets receivers (Section 4.1.1.2):

- o FEC Encoding ID: changing this parameter leads a receiver to consider a different FEC Scheme. The consequences are severe, the format of the Explicit Source FEC Payload ID and Repair FEC Payload ID of received packets will probably differ, leading to various malfunctions. Even if the original and modified FEC Schemes share the same format, FEC decoding will either fail or lead to corrupted decoded symbols. This will happen if an attacker turns value YYYY (i.e., RLC over $GF(2)$) to value XXXX (RLC over $GF(2^{8})$), an additional consequence being a higher processing overhead at the receiver. In any case, the attack results in a form of Denial of Service (DoS) or corrupted content.
- o Encoding symbol length (E): setting this E parameter to a different value will confuse a receiver. If the size of a received FEC Repair Packet is no longer multiple of the modified E value, a receiver quickly detects a problem and SHOULD reject the packet. If the new E value is a sub-multiple of the original E value (e.g., half the original value), then receivers may not detect the problem immediately. For instance, a receiver may think that a received FEC Repair Packet contains more repair symbols (e.g., twice as many if E is reduced by half), leading to malfunctions whose nature depends on implementation details. Here also, the attack always results in a form of DoS or corrupted content.

It is therefore RECOMMENDED that security measures be taken to guarantee the FFCI integrity, as specified in [RFC6363]. How to achieve this depends on the way the FFCI is communicated from the sender to the receiver, which is not specified in this document.

Similarly, attacks are possible against the Explicit Source FEC Payload ID and Repair FEC Payload ID. More specifically, in case of a FEC Source Packet, the following value can be modified by an attacker who targets receivers:

- o Encoding Symbol ID (ESI): changing the ESI leads a receiver to consider a wrong ADU, resulting in severe consequences, including corrupted content passed to the receiving application;

And in case of a FEC Repair Packet:

- o Repair Key: changing this value leads a receiver to generate a wrong coding coefficient sequence, and therefore any source symbol decoded using the repair symbols contained in this packet will be corrupted;
- o DT: changing this value also leads a receiver to generate a wrong coding coefficient sequence, and therefore any source symbol decoded using the repair symbols contained in this packet will be corrupted. In addition, if the DT value is significantly

- increased, it will generate a higher processing overhead at a receiver. In case of very large encoding windows, this may impact the terminal performance;
- o NSS: changing this value leads a receiver to consider a different set of source symbols, and therefore any source symbol decoded using the repair symbols contained in this packet will be corrupted. In addition, if the NSS value is significantly increased, it will generate a higher processing overhead at a receiver, which may impact the terminal performance;
 - o FSS_ESI: changing this value also leads a receiver to consider a different set of source symbols and therefore any source symbol decoded using the repair symbols contained in this packet will be corrupted.

It is therefore RECOMMENDED that security measures are taken to guarantee the FEC Source and Repair Packets as stated in [RFC6363].

8.3. When Several Source Flows are to be Protected Together

The Sliding Window RLC FEC Scheme specified in this document does not change the recommendations of [RFC6363].

8.4. Baseline Secure FEC Framework Operation

The Sliding Window RLC FEC Scheme specified in this document does not change the recommendations of [RFC6363] concerning the use of the IPsec/ESP security protocol as a mandatory to implement (but not mandatory to use) security scheme. This is well suited to situations where the only insecure domain is the one over which the FEC Framework operates.

8.5. Additional Security Considerations for Numerical Computations

In addition to the above security considerations, inherited from [RFC6363], the present document introduces several formulae, in particular in Appendix C.1. It is RECOMMENDED to check that the computed values stay within reasonable bounds since numerical overflows, caused by an erroneous implementation or an erroneous input value, may lead to hazardous behaviours. However, what "reasonable bounds" means is use-case and implementation dependent and is not detailed in this document.

Appendix C.2 also mentions the possibility of "using the timestamp field of an RTP packet header" when applicable. A malicious attacker may deliberately corrupt this header field in order to trigger hazardous behaviours at a FECFRAME receiver. Protection against this type of content corruption can be addressed with the above recommendations on a baseline secure operation. In addition, it is

also RECOMMENDED to check that the timestamp value be within reasonable bounds.

9. Operations and Management Considerations

The FEC Framework document [RFC6363] provides a fairly comprehensive analysis of operations and management considerations applicable to FEC Schemes. Therefore, the present section only discusses specific topics.

9.1. Operational Recommendations: Finite Field GF(2) Versus GF(2⁸)

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(2⁸) FEC Scheme will be mostly used since it warrants a higher packet loss protection. In case of small encoding windows, the associated processing overhead is not an issue (e.g., we measured decoding speeds between 745 Mbps and 2.8 Gbps on an ARM Cortex-A15 embedded board in [Roca17] depending on the code rate and the channel conditions, using an encoding window of size 18 or 23 symbols; see the above article for the details). Of course the CPU overhead will increase with the encoding window size, because more operations in the GF(2⁸) 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(2⁸), it is usually appropriate that small encoding windows be associated to a density threshold equal to 15, the maximum value, in order to warrant a high loss protection.

On the opposite, with larger encoding windows, it is usually appropriate that the density threshold be reduced. With large encoding windows, an alternative can be to use RLC over GF(2) and a density threshold equal to 7 (i.e., an average density equal to 1/2) or smaller.

Note that using a density threshold equal to 15 with RLC over GF(2) is equivalent to using an XOR code that computes the XOR sum of all

the source symbols in the encoding window. In that case: (1) only a single repair symbol can be produced for any encoding window, and (2) the `repair_key` parameter becomes useless (the coding coefficients generation function does not rely on the PRNG).

10. IANA Considerations

This document registers two values in the "FEC Framework (FECFRAME) FEC Encoding IDs" registry [RFC6363] as follows:

- o YYYY refers to the Sliding Window Random Linear Codes (RLC) over GF(2) FEC Scheme for Arbitrary Packet Flows, as defined in Section 5 of this document.
- o XXXX refers to the Sliding Window Random Linear Codes (RLC) over GF(2⁸) FEC Scheme for Arbitrary Packet Flows, as defined in Section 4 of this document.

11. Acknowledgments

The authors would like to thank the three TSVWG chairs, Wesley Eddy, our shepherd, David Black and Gorrry Fairhurst, as well as Spencer Dawkins, our responsible AD, and all those who provided comments, namely (alphabetical order) Alan DeKok, Jonathan Detchart, Russ Housley, Emmanuel Lochin, Marie-Jose Montpetit, and Greg Skinner. Last but not least, the authors are really grateful to the IESG members, in particular Benjamin Kaduk, Mirja Kuhlewind, Eric Rescorla, Adam Roach, and Roman Danyliw for their highly valuable feedbacks that greatly contributed to improve this specification.

12. References

12.1. Normative References

- [C99] "Programming languages - C: C99, correction 3:2007", International Organization for Standardization, ISO/IEC 9899:1999/Cor 3:2007, November 2007.
- [fecframe-ext]
Roca, V. and A. Begen, "Forward Error Correction (FEC) Framework Extension to Sliding Window Codes", Transport Area Working Group (TSVWG) draft-ietf-tsvwg-fecframe-ext (Work in Progress), January 2019, <<https://tools.ietf.org/html/draft-ietf-tsvwg-fecframe-ext>>.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC6363] Watson, M., Begen, A., and V. Roca, "Forward Error Correction (FEC) Framework", RFC 6363, DOI 10.17487/RFC6363, October 2011, <<https://www.rfc-editor.org/info/rfc6363>>.
- [RFC6364] Begen, A., "Session Description Protocol Elements for the Forward Error Correction (FEC) Framework", RFC 6364, DOI 10.17487/RFC6364, October 2011, <<https://www.rfc-editor.org/info/rfc6364>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [tinymt32] Saito, M., Matsumoto, M., Roca, V., and E. Baccelli, "TinyMT32 Pseudo Random Number Generator (PRNG)", Transport Area Working Group (TSVWG) draft-roca-tsvwg-tinymt32 (Work in Progress), February 2019, <<https://tools.ietf.org/html/draft-roca-tsvwg-tinymt32>>.

12.2. Informative References

- [PGM13] Plank, J., Greenan, K., and E. Miller, "A Complete Treatment of Software Implementations of Finite Field Arithmetic for Erasure Coding Applications", University of Tennessee Technical Report UT-CS-13-717, <http://web.eecs.utk.edu/~plank/plank/papers/UT-CS-13-717.html>, October 2013, <<http://web.eecs.utk.edu/~plank/plank/papers/UT-CS-13-717.html>>.
- [RFC5170] Roca, V., Neumann, C., and D. Furodet, "Low Density Parity Check (LDPC) Staircase and Triangle Forward Error Correction (FEC) Schemes", RFC 5170, DOI 10.17487/RFC5170, June 2008, <<https://www.rfc-editor.org/info/rfc5170>>.
- [RFC5510] Lacan, J., Roca, V., Peltotalo, J., and S. Peltotalo, "Reed-Solomon Forward Error Correction (FEC) Schemes", RFC 5510, DOI 10.17487/RFC5510, April 2009, <<https://www.rfc-editor.org/info/rfc5510>>.

- [RFC6681] Watson, M., Stockhammer, T., and M. Luby, "Raptor Forward Error Correction (FEC) Schemes for FECFRAME", RFC 6681, DOI 10.17487/RFC6681, August 2012, <<https://www.rfc-editor.org/info/rfc6681>>.
- [RFC6726] Paila, T., Walsh, R., Luby, M., Roca, V., and R. Lehtonen, "FLUTE - File Delivery over Unidirectional Transport", RFC 6726, DOI 10.17487/RFC6726, November 2012, <<https://www.rfc-editor.org/info/rfc6726>>.
- [RFC6816] Roca, V., Cunche, M., and J. Lacan, "Simple Low-Density Parity Check (LDPC) Staircase Forward Error Correction (FEC) Scheme for FECFRAME", RFC 6816, DOI 10.17487/RFC6816, December 2012, <<https://www.rfc-editor.org/info/rfc6816>>.
- [RFC6865] Roca, V., Cunche, M., Lacan, J., Bouabdallah, A., and K. Matsuzono, "Simple Reed-Solomon Forward Error Correction (FEC) Scheme for FECFRAME", RFC 6865, DOI 10.17487/RFC6865, February 2013, <<https://www.rfc-editor.org/info/rfc6865>>.
- [RFC8406] Adamson, B., Adjih, C., Bilbao, J., Firoiu, V., Fitzek, F., Ghanem, S., Lochin, E., Masucci, A., Montpetit, M-J., Pedersen, M., Peralta, G., Roca, V., Ed., Saxena, P., and S. Sivakumar, "Taxonomy of Coding Techniques for Efficient Network Communications", RFC 8406, DOI 10.17487/RFC8406, June 2018, <<https://www.rfc-editor.org/info/rfc8406>>.
- [Roca16] Roca, V., Teibi, B., Burdinat, C., Tran, T., and C. Thienot, "Block or Convolutional AL-FEC Codes? A Performance Comparison for Robust Low-Latency Communications", HAL open-archive document, hal-01395937 <https://hal.inria.fr/hal-01395937/en/>, November 2016, <<https://hal.inria.fr/hal-01395937/en/>>.
- [Roca17] Roca, V., Teibi, B., Burdinat, C., Tran, T., and C. Thienot, "Less Latency and Better Protection with AL-FEC Sliding Window Codes: a Robust Multimedia CBR Broadcast Case Study", 13th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob17), October 2017 <https://hal.inria.fr/hal-01571609v1/en/>, October 2017, <<https://hal.inria.fr/hal-01571609v1/en/>>.

Appendix A. TinyMT32 Validation Criteria (Normative)

PRNG determinism, for a given seed, is a requirement. Consequently, in order to validate an implementation of the TinyMT32 PRNG, the following criteria MUST be met.

The first criterion focusses on the `tinymt32_rand256()`, where the 32-bit integer of the core TinyMT32 PRNG is scaled down to an 8-bit integer. Using a seed value of 1, the first 50 values returned by: `tinymt32_rand256()` as 8-bit unsigned integers MUST be equal to values provided in Figure 9, to be read line by line.

37	225	177	176	21
246	54	139	168	237
211	187	62	190	104
135	210	99	176	11
207	35	40	113	179
214	254	101	212	211
226	41	234	232	203
29	194	211	112	107
217	104	197	135	23
89	210	252	109	166

Figure 9: First 50 decimal values (to be read per line) returned by `tinymt32_rand256()` as 8-bit unsigned integers, with a seed value of 1.

The second criterion focusses on the `tinymt32_rand16()`, where the 32-bit integer of the core TinyMT32 PRNG is scaled down to a 4-bit integer. Using a seed value of 1, the first 50 values returned by: `tinymt32_rand16()` as 4-bit unsigned integers MUST be equal to values provided in Figure 10, to be read line by line.

5	1	1	0	5
6	6	11	8	13
3	11	14	14	8
7	2	3	0	11
15	3	8	1	3
6	14	5	4	3
2	9	10	8	11
13	2	3	0	11
9	8	5	7	7
9	2	12	13	6

Figure 10: First 50 decimal values (to be read per line) returned by `tinymt32_rand16()` as 4-bit unsigned integers, with a seed value of 1.

Appendix B. Assessing the PRNG Adequacy (Informational)

This annex discusses the adequacy of the TinyMT32 PRNG and the `tinymt32_rand16()` and `tinymt32_rand256()` functions, to the RLC FEC Schemes. The goal is to assess the adequacy of these two functions in producing coding coefficients that are sufficiently different from one another, across various repair symbols with repair key values in sequence (we can expect this approach to be commonly used by implementers, see Section 6.1). This section is purely informational and does not claim to be a solid evaluation.

The two RLC FEC Schemes use the PRNG to produce pseudo-random coding coefficients (Section 3.6), each time a new repair symbol is needed. A different repair key is used for each repair symbol, usually by incrementing the repair key value (Section 6.1). For each repair symbol, a limited number of pseudo-random numbers is needed, depending on the DT and encoding window size (Section 3.6), using either `tinymt32_rand16()` or `tinymt32_rand256()`. Therefore we are more interested in the randomness of small sequences of random numbers mapped to 4-bit or 8-bit integers, than in the randomness of a very large sequence of random numbers which is not representative of the usage of the PRNG.

Evaluation of `tinymt32_rand16()`: We first generate a huge number (1,000,000,000) of small sequences (20 pseudo-random numbers per sequence), increasing the seed value for each sequence, and perform statistics on the number of occurrences of each of the 16 possible values across all sequences. In this first test we consider 32-bit seed values in order to assess the PRNG quality after output truncation to 4 bits.

value	occurrences	percentage (%) (total of 20000000000)
0	1250036799	6.2502
1	1249995831	6.2500
2	1250038674	6.2502
3	1250000881	6.2500
4	1250023929	6.2501
5	1249986320	6.2499
6	1249995587	6.2500
7	1250020363	6.2501
8	1249995276	6.2500
9	1249982856	6.2499
10	1249984111	6.2499
11	1250009551	6.2500
12	1249955768	6.2498
13	1249994654	6.2500
14	1250000569	6.2500
15	1249978831	6.2499

Figure 11: `tinymt32_rand16()`: occurrence statistics across a huge number (1,000,000,000) of small sequences (20 pseudo-random numbers per sequence), with 0 as the first PRNG seed.

The results (Figure 11) show that all possible values are almost equally represented, or said differently, that the `tinymt32_rand16()` output converges to a uniform distribution where each of the 16 possible values would appear exactly $1 / 16 * 100 = 6.25\%$ of times.

Since the RLC FEC Schemes use of this PRNG will be limited to 16-bit seed values, we carried out the same test for the first 2^{16} seed values only. The distribution (not shown) is of course less uniform, with value occurrences ranging between 6.2121% (i.e., 81,423 occurrences out of a total of $65536 * 20 = 1,310,720$) and 6.2948% (i.e., 82,507 occurrences). However, we do not believe it significantly impacts the RLC FEC Scheme behavior.

Other types of biases may exist that may be visible with smaller tests, for instance to evaluate the convergence speed to a uniform distribution. We therefore perform 200 tests, each of them consisting in producing 200 sequences, keeping only the first value of each sequence. We use non overlapping repair keys for each sequence, starting with value 0 and increasing it after each use.

value	min occurrences	max occurrences	average occurrences
0	4	21	6.3675
1	4	22	6.0200
2	4	20	6.3125
3	5	23	6.1775
4	5	24	6.1000
5	4	21	6.5925
6	5	30	6.3075
7	6	22	6.2225
8	5	26	6.1750
9	3	21	5.9425
10	5	24	6.3175
11	4	22	6.4300
12	5	21	6.1600
13	5	22	6.3100
14	4	26	6.3950
15	4	21	6.1700

Figure 12: `tinymt32_rand16()`: occurrence statistics across 200 tests, each of them consisting in 200 sequences of 1 pseudo-random number each, with non overlapping PRNG seeds in sequence starting from 0.

Figure 12 shows across all 200 tests, for each of the 16 possible pseudo-random number values, the minimum (resp. maximum) number of times it appeared in a test, as well as the average number of occurrences across the 200 tests. Although the distribution is not perfect, there is no major bias. On the opposite, in the same conditions, the Park-Miller linear congruential PRNG of [RFC5170] with a result scaled down to 4-bit values, using seeds in sequence starting from 1, returns systematically 0 as the first value during some time, then after a certain repair key value threshold, it systematically returns 1, etc.

Evaluation of `tinymt32_rand256()`: The same approach is used here. Results (not shown) are similar: occurrences vary between 7,810,3368 (i.e., 0.3905%) and 7,814,7952 (i.e., 0.3907%). Here also we see a convergence to the theoretical uniform distribution where each of the 256 possible values would appear exactly $1 / 256 * 100 = 0.390625\%$ of times.

Appendix C. Possible Parameter Derivation (Informational)

Section 3.1 defines several parameters to control the encoder or decoder. This annex proposes techniques to derive these parameters according to the target use-case. This annex is informational, in the sense that using a different derivation technique will not prevent the encoder and decoder to interoperate: a decoder can still recover an erased source symbol without any error. However, in case

of a real-time flow, an inappropriate parameter derivation may lead to the decoding of erased source packets after their validity period, making them useless to the target application. This annex proposes an approach to reduce this risk, among other things.

The FEC Schemes defined in this document can be used in various manners, depending on the target use-case:

- o the source ADU flow they protect may or may not have real-time constraints;
- o the source ADU flow may be a Constant Bitrate (CBR) or Variable BitRate (VBR) flow;
- o with a VBR source ADU flow, the flow's minimum and maximum bitrates may or may not be known;
- o and the communication path between encoder and decoder may be a CBR communication path (e.g., as with certain LTE-based broadcast channels) or not (general case, e.g., with Internet).

The parameter derivation technique should be suited to the use-case, as described in the following sections.

C.1. Case of a CBR Real-Time Flow

In the following, we consider a real-time flow with `max_lat` latency budget. The encoding symbol size, `E`, is constant. The code rate, `cr`, is also constant, its value depending on the expected communication loss model (this choice is out of scope of this document).

In a first configuration, the source ADU flow bitrate at the input of the FECFRAME sender is fixed and equal to `br_in` (in bits/s), and this value is known by the FECFRAME sender. It follows that the transmission bitrate at the output of the FECFRAME sender will be higher, depending on the added repair flow overhead. In order to comply with the maximum FEC-related latency budget, we have:

$$\text{dw_max_size} = (\text{max_lat} * \text{br_in}) / (8 * E)$$

assuming that the encoding and decoding times are negligible with respect to the target `max_lat`. This is a reasonable assumption in many situations (e.g., see Section 9.1 in case of small window sizes). Otherwise the `max_lat` parameter should be adjusted in order to avoid the problem. In any case, interoperability will never be compromised by choosing a too large value.

In a second configuration, the FECFRAME sender generates a fixed bitrate flow, equal to the CBR communication path bitrate equal to `br_out` (in bits/s), and this value is known by the FECFRAME sender,

as in [Roca17]. The maximum source flow bitrate needs to be such that, with the added repair flow overhead, the total transmission bitrate remains inferior or equal to `br_out`. We have:

$$\text{dw_max_size} = (\text{max_lat} * \text{br_out} * \text{cr}) / (8 * E)$$

assuming here also that the encoding and decoding times are negligible with respect to the target `max_lat`.

For decoding to be possible within the latency budget, it is required that the encoding window maximum size be smaller than or at most equal to the decoding window maximum size. The `ew_max_size` is the main parameter at a FECFRAME sender, but its exact value has no impact on the the FEC-related latency budget. The `ew_max_size` parameter is computed as follows:

$$\text{ew_max_size} = \text{dw_max_size} * \text{WSR} / 255$$

In line with [Roca17], `WSR = 191` is considered as a reasonable value (the resulting encoding to decoding window size ratio is then close to 0.75), but other values between 1 and 255 inclusive are possible, depending on the use-case.

The `dw_max_size` is computed by a FECFRAME sender but not explicitly communicated to a FECFRAME receiver. However, a FECFRAME receiver can easily evaluate the `ew_max_size` by observing the maximum Number of Source Symbols (NSS) value contained in the Repair FEC Payload ID of received FEC Repair Packets (Section 4.1.3). A receiver can then easily compute `dw_max_size`:

$$\text{dw_max_size} = \text{max_NSS_observed} * 255 / \text{WSR}$$

A receiver can then chose an appropriate linear system maximum size:

$$\text{ls_max_size} \geq \text{dw_max_size}$$

It is good practice to use a larger value for `ls_max_size` as explained in Appendix D, which does not impact maximum latency nor interoperability.

In any case, for a given use-case (i.e., for target encoding and decoding devices and desired protection levels in front of communication impairments) and for the computed `ew_max_size`, `dw_max_size` and `ls_max_size` values, it is RECOMMENDED to check that the maximum encoding time and maximum memory requirements at a FECFRAME sender, and maximum decoding time and maximum memory requirements at a FECFRAME receiver, stay within reasonable bounds. When assuming that the encoding and decoding times are negligible

with respect to the target `max_lat`, this should be verified as well, otherwise the `max_lat` SHOULD be adjusted accordingly.

The particular case of session start needs to be managed appropriately since the `ew_size`, starting at zero, increases each time a new source ADU is received by the FECFRAME sender, until it reaches the `ew_max_size` value. Therefore a FECFRAME receiver SHOULD continuously observe the received FEC Repair Packets, since the NSS value carried in the Repair FEC Payload ID will increase too, and adjust its `ls_max_size` accordingly if need be. With a CBR flow, session start is expected to be the only moment when the encoding window size will increase. Similarly, with a CBR real-time flow, the session end is expected to be the only moment when the encoding window size will progressively decrease. No adjustment of the `ls_max_size` is required at the FECFRAME receiver in that case.

C.2. Other Types of Real-Time Flow

In the following, we consider a real-time source ADU flow with a `max_lat` latency budget and a variable bitrate (VBR) measured at the entry of the FECFRAME sender. A first approach consists in considering the smallest instantaneous bitrate of the source ADU flow, when this parameter is known, and to reuse the derivation of Appendix C.1. Considering the smallest bitrate means that the encoding and decoding window maximum size estimations are pessimistic: these windows have the smallest size required to enable on-time decoding at a FECFRAME receiver. If the instantaneous bitrate is higher than this smallest bitrate, this approach leads to an encoding window that is unnecessarily small, which reduces robustness in front of long erasure bursts.

Another approach consists in using ADU timing information (e.g., using the timestamp field of an RTP packet header, or registering the time upon receiving a new ADU). From the global FEC-related latency budget, the FECFRAME sender can derive a practical maximum latency budget for encoding operations, `max_lat_for_encoding`. For the FEC Schemes specified in this document, this latency budget SHOULD be computed with:

$$\text{max_lat_for_encoding} = \text{max_lat} * \text{WSR} / 255$$

It follows that any source symbols associated to an ADU that has timed-out with respect to `max_lat_for_encoding` SHOULD be removed from the encoding window. With this approach there is no pre-determined `ew_size` value: this value fluctuates over the time according to the instantaneous source ADU flow bitrate. For practical reasons, a FECFRAME sender may still require that `ew_size` does not increase beyond a maximum value (Appendix C.3).

With both approaches, and no matter the choice of the FECFRAME sender, a FECFRAME receiver can still easily evaluate the `ew_max_size` by observing the maximum Number of Source Symbols (NSS) value contained in the Repair FEC Payload ID of received FEC Repair Packets. A receiver can then compute `dw_max_size` and derive an appropriate `ls_max_size` as explained in Appendix C.1.

When the observed NSS fluctuates significantly, a FECFRAME receiver may want to adapt its `ls_max_size` accordingly. In particular when the NSS is significantly reduced, a FECFRAME receiver may want to reduce the `ls_max_size` too in order to limit computation complexity. A balance must be found between using an `ls_max_size` "too large" (which increases computation complexity and memory requirements) and the opposite (which reduces recovery performance).

C.3. Case of a Non Real-Time Flow

Finally there are configurations where a source ADU flow has no real-time constraints. FECFRAME and the FEC Schemes defined in this document can still be used. The choice of appropriate parameter values can be directed by practical considerations. For instance, it can derive from an estimation of the maximum memory amount that could be dedicated to the linear system at a FECFRAME receiver, or the maximum computation complexity at a FECFRAME receiver, both of them depending on the `ls_max_size` parameter. The same considerations also apply to the FECFRAME sender, where the maximum memory amount and computation complexity depend on the `ew_max_size` parameter.

Here also, the NSS value contained in FEC Repair Packets is used by a FECFRAME receiver to determine the current coding window size and `ew_max_size` by observing its maximum value over the time.

Appendix D. Decoding Beyond Maximum Latency Optimization (Informational)

This annex introduces non normative considerations. It is provided as suggestions, without any impact on interoperability. For more information see [Roca16].

With a real-time source ADU flow, it is possible to improve the decoding performance of sliding window codes without impacting maximum latency, at the cost of extra memory and CPU overhead. The optimization consists, for a FECFRAME receiver, to extend the linear system beyond the decoding window maximum size, by keeping a certain number of old source symbols whereas their associated ADUs timed-out:

$$ls_max_size > dw_max_size$$

Usually the following choice is a good trade-off between decoding performance and extra CPU overhead:

```
ls_max_size = 2 * dw_max_size
```

When the `dw_max_size` is very small, it may be preferable to keep a minimum `ls_max_size` value (e.g., `LS_MIN_SIZE_DEFAULT = 40` symbols). Going below this threshold will not save a significant amount of memory nor CPU cycles. Therefore:

```
ls_max_size = max(2 * dw_max_size, LS_MIN_SIZE_DEFAULT)
```

Finally, it is worth noting that a receiver that benefits from an FEC protection significantly higher than what is required to recover from packet losses, can choose to reduce the `ls_max_size`. In that case lost ADUs will be recovered without relying on this optimization.

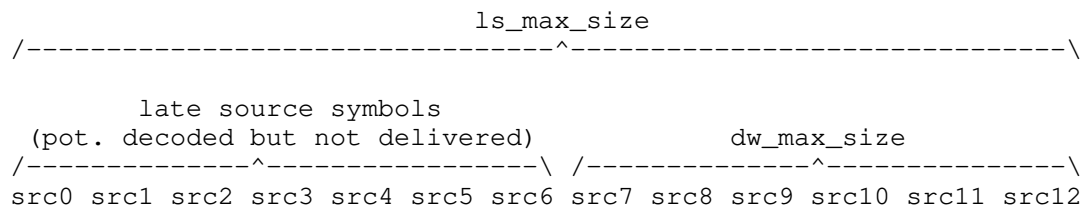


Figure 13: Relationship between parameters to decode beyond maximum latency.

It means that source symbols, and therefore ADUs, may be decoded even if the added latency exceeds the maximum value permitted by the application (the "late source symbols" of Figure 13). It follows that the corresponding ADUs will not be useful to the application. However, decoding these "late symbols" significantly improves the global robustness in bad reception conditions and is therefore recommended for receivers experiencing bad communication conditions [Roca16]. In any case whether or not to use this optimization and what exact value to use for the `ls_max_size` parameter are local decisions made by each receiver independently, without any impact on the other receivers nor on the source.

Authors' Addresses

Vincent Roca
INRIA
Univ. Grenoble Alpes
France

EMail: vincent.roca@inria.fr

Belkacem Teibi
INRIA
Univ. Grenoble Alpes
France

EMail: belkacem.teibi@gmail.com

Internet Engineering Task Force
INTERNET-DRAFT
Intended Status: Informational
Expires: November 7, 2019

X. Wei
Y. Li
Huawei Technologies
S. Boutros
VMware
L. Geng
China Mobile
May 6, 2019

Tunnel Congestion Feedback
draft-ietf-tsvwg-tunnel-congestion-feedback-07

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/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>

Copyright and License Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. 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 tunnelEcnEct0NectBytetTotalCount	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 ingresses 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.

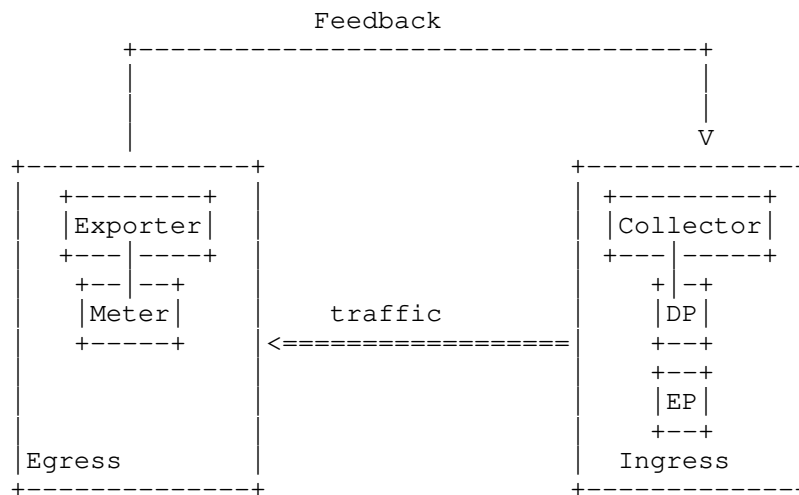


Figure 1: Feedback Model.

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:

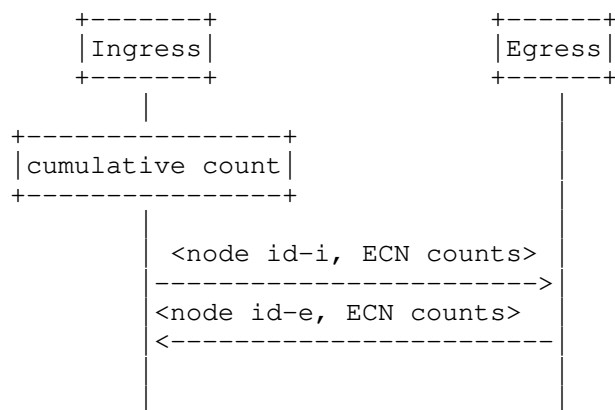


Figure 2: Procedure of Packet Loss Measurement

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

Statuses: current

Units: bytes

5.1.2 tunnelEcnEct0NectBytetTotalCount

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

Statuses: 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

Statuses: 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

Statuses: 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

Statuses: current

Units: bytes

5.1.9 tunnelEcnCEMarkedRatio

Description: The ratio of CE-marked Packet at the Observation Point.

Abstract Data Type: float32

ElementId: TBD8

Statuses: 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.

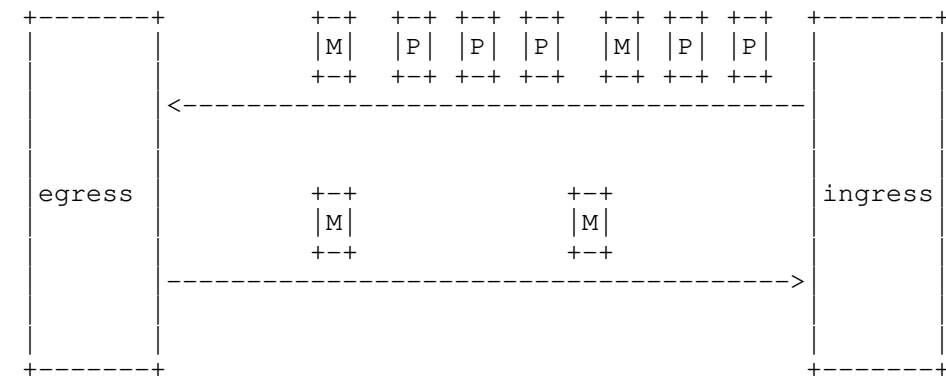
Figure 3 to Figure 6 below show the example procedure between ingress and egress.

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
tunnelEcnCeNectByteTotalCount	Field Length=8
tunnelEcnCeEctByteTotalCount	Field Length=8
tunnelEcnCEMarkedRatio	Field Length=4

Figure 3: Template Record Sent From Egress to Ingress

Set ID=2	Length=28
Template ID=257	Field Count =3
tunnelEcnCeCeByteTotalCount	Field Length=8
tunnelEcnEctNectByteTotalCount	Field Length=8
tunnelEcnEctEctByteTotalCount	Field Length=8

Figure 4: Template Record Sent From Ingress to Egress



+-+
 |M| : Message Packet
 +-+

+-+
 |P| : User Packet
 +-+

Figure 5 Traffic flow Between Ingress and Egress

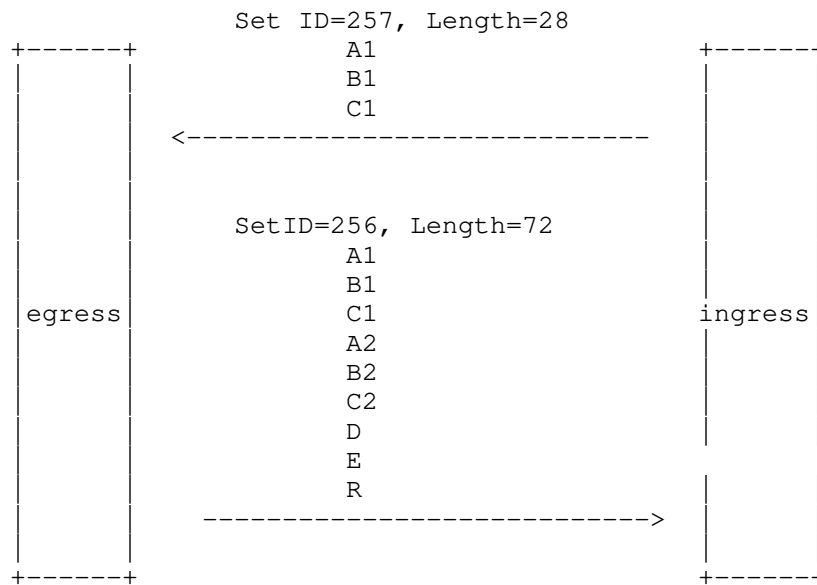


Figure 6: Message Between Ingress and Egress

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:

$total_ingress = (A1 + B1 + C1)$

$total_egress = (A2 + B2 + C2 + D + E)$

$volume_loss = (total_ingress - total_egress)$

7. Security Considerations

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

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.
- [RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, May 2004.
- [RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, March 2006.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, September 2007.
- [RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, November 2010.
- [CONEX] Matt Mathis, Bob Briscoe. "Congestion Exposure (ConEx) Concepts, Abstract Mechanism and Requirements", RFC7713, December 2015

9.2 Informative References

[RFC8084] G. Fairhurst. "Network Transport Circuit Breakers", draft-ietf-tsvwg-circuit-breaker-01, April 02, 2015

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 Lei Zhu, Lingli Deng, 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
EMail: weixinpeng@huawei.com

Yizhou Li
Huawei Technologies
101 Software Avenue,
Nanjing 210012
China

Phone: +86-25-56624584
EMail: liyizhou@huawei.com

Sami Boutros
VMware, Inc.
EMail: boutross@vmware.com

Liang Geng
China Mobile
EMail: gengliang@chinamobile.com

TSVWG
Internet Draft
Intended status: Standards Track
Intended updates: 768
Expires: September 2022

J. Touch
Independent Consultant
March 26, 2022

Transport Options for UDP
draft-ietf-tsvwg-udp-options-18.txt

Abstract

Transport protocols are extended through the use of transport header options. This document extends UDP by indicating the location, syntax, and semantics for UDP transport layer options.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at <https://www.ietf.org/shadow.html>

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 26, 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction.....	3
2. Conventions used in this document.....	3
3. Terminology.....	3
4. Background.....	4
5. The UDP Option Area.....	5
6. The UDP Surplus Area Structure.....	8
7. The Option Checksum (OCS).....	8
8. UDP Options.....	10
9. Safe UDP Options.....	13
9.1. End of Options List (EOL).....	13
9.2. No Operation (NOP).....	14
9.3. Alternate Payload Checksum (APC).....	14
9.4. Fragmentation (FRAG).....	16
9.5. Maximum Datagram Size (MDS).....	19
9.6. Maximum Reassembled Datagram Size (MRDS).....	20
9.7. Echo request (REQ) and echo response (RES).....	21
9.8. Timestamps (TIME).....	21
9.9. Authentication (AUTH).....	22
9.10. Experimental (EXP).....	23
10. UNSAFE Options.....	24
10.1. UNSAFE Encryption (UENC).....	25
10.2. UNSAFE Experimental (UEXP).....	25
11. Rules for designing new options.....	25
12. Option inclusion and processing.....	26
13. UDP API Extensions.....	28
14. UDP Options are for Transport, Not Transit.....	29
15. UDP options vs. UDP-Lite.....	29
16. Interactions with Legacy Devices.....	30
17. Options in a Stateless, Unreliable Transport Protocol.....	30
18. UDP Option State Caching.....	31
19. Updates to RFC 768.....	31
20. Interactions with other RFCs (and drafts).....	32
21. Multicast Considerations.....	33
22. Security Considerations.....	33
23. IANA Considerations.....	34
24. References.....	35
24.1. Normative References.....	35

24.2. Informative References.....	35
25. Acknowledgments.....	38
Appendix A. Implementation Information.....	39

1. Introduction

Transport protocols use options as a way to extend their capabilities. TCP [RFC793], SCTP [RFC4960], and DCCP [RFC4340] include space for these options but UDP [RFC768] currently does not. This document defines an extension to UDP that provides space for transport options including their generic syntax and semantics for their use in UDP's stateless, unreliable message protocol.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

In this document, the characters ">>" preceding an indented line(s) indicates a statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the portions of this RFC covered by these key words.

3. Terminology

The following terminology is used in this document:

- o IP datagram [RFC791][RFC8200] - an IP packet, composed of the IP header and an IP payload area
- o User datagram - a UDP packet, composed of a UDP header and UDP payload; as discussed herein, that payload need not extend to the end of the IP datagram
- o UDP packet - the more contemporary term used herein to refer to a user datagram [RFC768]
- o Surplus area - the area of an IP payload that follows a UDP packet; this area is used for UDP options in this document

- o UDP fragment - one or more components of a UDP packet and its UDP options that enables transmission as IP payloads larger than permitted by IP datagram maximum sizes; note that each UDP fragment is itself transmitted as a UDP packet with its own options
- o (UDP) User data - the user data field of a UDP packet [RFC768]
- o UDP Length - the length field of a UDP header [RFC768]
- o Must-support options - UDP options that all implementations are required to support. Their use in individual UDP packets is optional.

4. Background

Many protocols include a default, invariant header and an area for header options that varies from packet to packet. These options enable the protocol to be extended for use in particular environments or in ways unforeseen by the original designers. Examples include TCP's Maximum Segment Size, Window Scale, Timestamp, and Authentication Options [RFC793][RFC5925][RFC7323].

Header options are used both in stateful (connection-oriented, e.g., TCP [RFC793], SCTP [RFC4960], DCCP [RFC4340]) and stateless (connectionless, e.g., IPv4 [RFC791], IPv6 [RFC8200]) protocols. In stateful protocols they can help extend the way in which state is managed. In stateless protocols their effect is often limited to individual packets, but they can have an aggregate effect on a sequence of packets as well.

UDP is one of the most popular protocols that lacks space for header options [RFC768]. The UDP header was intended to be a minimal addition to IP, providing only ports and a checksum for error detection. This document extends UDP to provide a trailer area for such options, located after the UDP user data.

UDP options are possible because UDP includes its own length field, separate from that of the IP header. Other transport protocols infer transport payload length from the IP datagram length (TCP, DCCP, SCTP). There are a number of reasons why Internet historians suggest that UDP includes this field, e.g., to support multiple UDP packets within the same IP datagram or to indicate the length of the UDP user data as distinct from zero padding required for systems that require writes that are not byte-aligned. These suggestions are not consistent with earlier versions of UDP or with concurrent design of multi-segment multiplexing protocols, however, so the real reason

remains unknown. Regardless, this field presents an opportunity to differentiate the UDP user data from the implied transport payload length, which this document leverages to support a trailer options field.

There are other ways to include additional header fields or options in protocols that otherwise are not extensible. In particular, in-band encoding can be used to differentiate transport payload from additional fields, such as was proposed in [Hi15]. This approach can cause complications for interactions with legacy devices, and is thus not considered further in this document.

IPv6 Teredo [RFC6081] uses values of the UDP Length that are larger than the IP payload as an additional type of signal, as noted in Section 20. UDP options uses a value smaller than the IP payload to enable backwards compatibility with existing UDP implementations, i.e., to deliver the UDP Length of UDP user data to the application and silently ignore the additional surplus area data. Using a value larger than the IP payload could either be considered malformed (and ought to be silently dropped by UDP processing) or could cause buffer overruns, and so is not considered silently and safely backward compatible.

5. The UDP Option Area

The UDP transport header includes demultiplexing and service identification (port numbers), an error detection checksum, and a field that indicates the UDP datagram length (including UDP header). The UDP Length field is typically redundant with the size of the maximum space available as a transport protocol payload, as determined by the IP header (see detail in Section 16). The UDP Option area is created when the UDP Length indicates a smaller transport payload than implied by the IP header.

For IPv4, IP Total Length field indicates the total IP datagram length (including IP header) and the size of the IP options is indicated in the IP header (in 4-byte words) as the "Internet Header Length" (IHL), as shown in Figure 1 [RFC791]. As a result, the typical (and largest valid) value for UDP Length is:

$$\text{UDP_Length} = \text{IPv4_Total_Length} - \text{IPv4_IHL} * 4$$

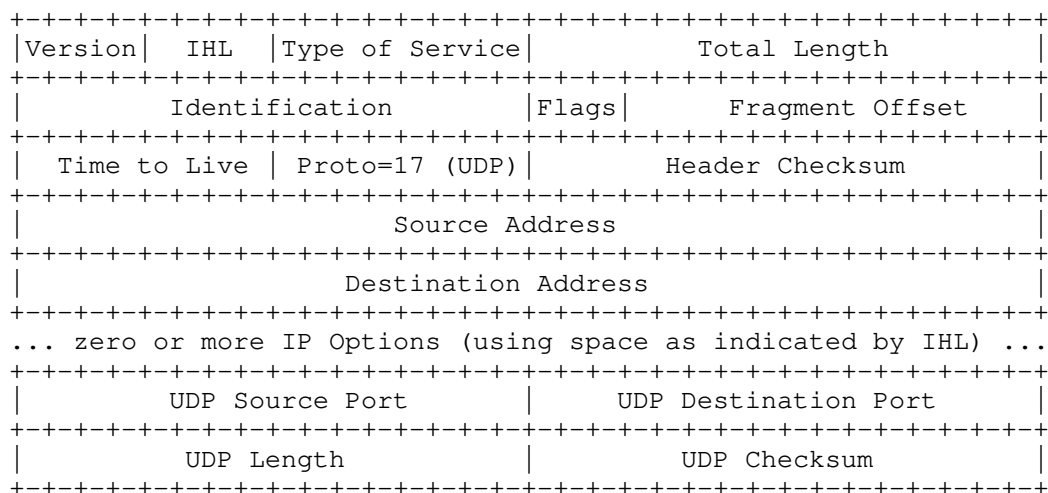


Figure 1 IPv4 datagram with UDP header

For IPv6, the IP Payload Length field indicates the transport payload after the base IPv6 header, which includes the IPv6 extension headers and space available for the transport protocol, as shown in Figure 2 [RFC8200]. Note that the Next HDR field in IPv6 might not indicate UDP (i.e., 17), e.g., when intervening IP extension headers are present. For IPv6, the lengths of any additional IP extensions are indicated within each extension [RFC8200], so the typical (and largest valid) value for UDP Length is:

$$\text{UDP_Length} = \text{IPv6_Payload_Length} - \text{sum}(\text{extension header lengths})$$

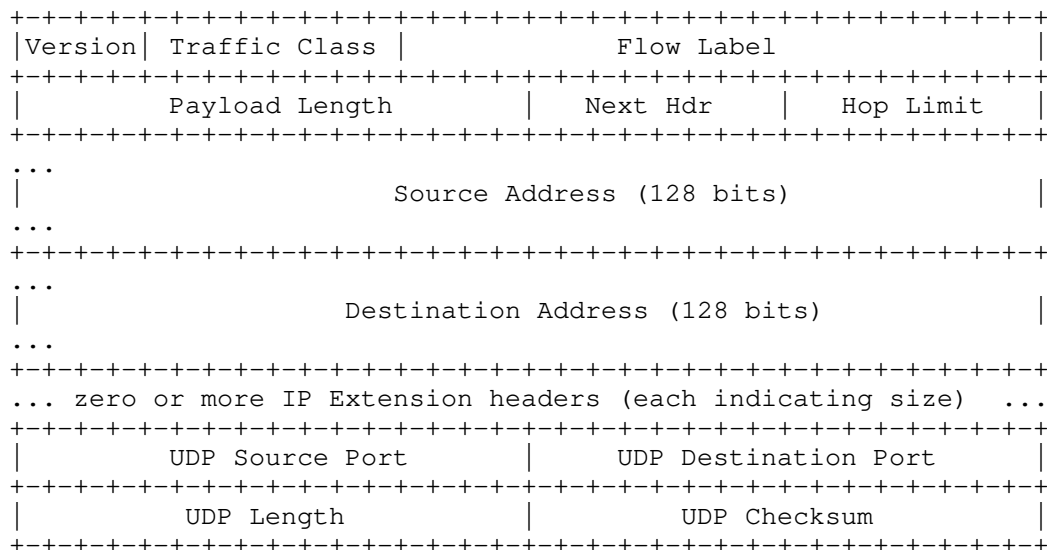


Figure 2 IPv6 datagram with UDP header

In both cases, the space available for the UDP packet is indicated by IP, either directly in the base header (for IPv4) or by adding information in the extensions (for IPv6). In either case, this document will refer to this available space as the "IP transport payload".

As a result of this redundancy, there is an opportunity to use the UDP Length field as a way to break up the IP transport payload into two areas - that intended as UDP user data and an additional "surplus area" (as shown in Figure 3).

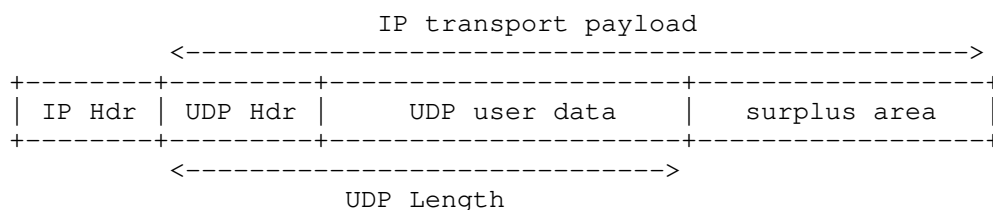


Figure 3 IP transport payload vs. UDP Length

In most cases, the IP transport payload and UDP Length point to the same location, indicating that there is no surplus area. This is not

a requirement of UDP [RFC768] (discussed further in Section 16). This document uses the surplus area for UDP options.

The surplus area can commence at any valid byte offset, i.e., it need not be 16-bit or 32-bit aligned. In effect, this document redefines the UDP "Length" field as a "trailer options offset".

6. The UDP Surplus Area Structure

UDP options use the entire surplus area, i.e., the contents of the IP payload after the last byte of the UDP payload. They commence with a 2-byte Option Checksum (OCS) field aligned to the first 2-byte boundary (relative to the start of the IP datagram) of that area, using zeroes for alignment. The UDP option area can be used with any UDP payload length (including zero), as long as there remains enough space for the aligned OCS and the options used.

>> UDP options MAY begin at any UDP length offset.

>> Option area bytes used for alignment before the OCS MUST be zero.

The OCS contains an optional ones-complement sum that detects errors in the surplus area, which is not otherwise covered by the UDP checksum, as detailed in Section 7.

The remainder of the surplus area consists of options defined using a TLV (type, length, and optional value) syntax similar to that of TCP [RFC793], as detailed in Section 8. These options continue until the end of the surplus area or can end earlier using the EOL (end of list) option, followed by zeroes.

7. The Option Checksum (OCS)

The Option Checksum (OCS) option is conventional Internet checksum [RFC791] that detects errors in the surplus area. The OCS option contains a 16-bit checksum that is aligned to the first 2-byte boundary, preceded by zeroes for padding (if needed), as shown in Figure 4.

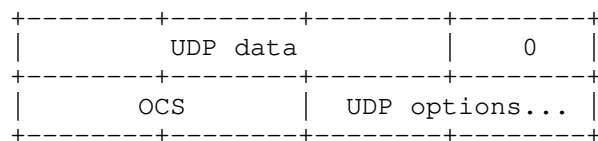


Figure 4 UDP OCS format, here using one zero for alignment

The OCS consists of a 16-bit Internet checksum [RFC1071], computed over the surplus area and including the length of the surplus area as an unsigned 16-bit value. The OCS protects the surplus area from errors in a similar way that the UDP checksum protects the UDP user data (when not zero).

The primary purpose of the OCS is to detect non-standard (i.e., non-option) uses of that area and accidental errors. It is not intended to detect attacks, as discussed further in Section 22.

The design enables traversal of errant middleboxes that incorrectly compute the UDP checksum over the entire IP payload [Fal8], rather than only the UDP header and UDP payload (as indicated by the UDP header length). Because the OCS is computed over the surplus area and its length and then inverted, OCS effectively negates the effect that incorrectly including the surplus has on the UDP checksum. As a result, when OCS is non-zero, the UDP checksum is the same in either case.

>> OCS MUST be non-zero when the UDP checksum is non-zero.

>> When the UDP checksum is zero, the OCS MAY be unused, and is then indicated by a zero OCS value.

Like the UDP checksum, the OCS is optional under certain circumstances and contains zero when not used. UDP checksums can be zero for IPv4 [RFC791] and for IPv6 [RFC8200] when UDP payload already covered by another checksum, as might occur for tunnels [RFC6935]. The same exceptions apply to the OCS when used to detect bit errors; an additional exception occurs for its use in the UDP datagram prior to fragmentation or after reassembly (see Section 9.4).

The OCS covers the surplus area as formatted for transmission and is processed immediately upon reception.

>> If the OCS fails, all options MUST be ignored and the surplus area silently discarded.

>> UDP user data that is validated by a correct UDP checksum MUST be delivered to the application layer, even if the OCS fails, unless the endpoints have negotiated otherwise for this UDP packet's socket pair.

When not used (i.e., containing zero), the OCS is assumed to be "correct" for the purpose of accepting UDP datagrams at a receiver (see Section 12).

8. UDP Options

UDP options are typically a minimum of two bytes in length as shown in Figure 5, excepting only the one byte options "No Operation" (NOP) and "End of Options List" (EOL) described below.

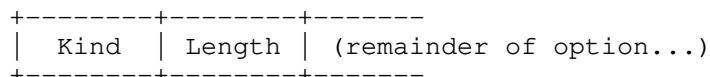


Figure 5 UDP option default format

The Kind field is always one byte. The Length field is one byte for all lengths below 255 (including the Kind and Length bytes). A Length of 255 indicates use of the UDP option extended format shown in Figure 6. The Extended Length field is a 16-bit field in network standard byte order.

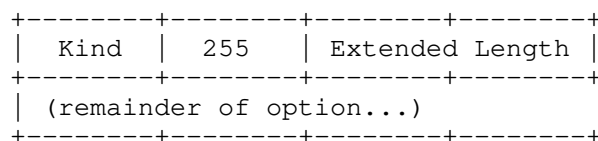


Figure 6 UDP option extended format

>> The UDP length MUST be at least as large as the UDP header (8) and no larger than the IP transport payload. Datagrams with length values outside this range MUST be silently dropped as invalid and logged where rate-limiting permits.

>> Option Lengths (or Extended Lengths, where applicable) smaller than the minimum for the corresponding Kind MUST be treated as an error. Such errors call into question the remainder of the surplus area and thus MUST result in all UDP options being silently discarded.

>> Any UDP option other than EOL and NOP MAY use either the default or extended option formats.

>> Any UDP option whose length is larger than 254 MUST use the UDP option extended format shown in Figure 6.

>> For compactness, UDP options SHOULD use the smallest option format possible.

>> UDP options MUST be interpreted in the order in which they occur in the surplus area.

The following UDP options are currently defined:

Kind	Length	Meaning

0*	-	End of Options List (EOL)
1*	-	No operation (NOP)
2*	6	Alternate payload checksum (APC)
3*	10/12	Fragmentation (FRAG)
4*	4	Maximum datagram size (MDS)
5*	4	Maximum reassembled datagram size (MRDS)
6*	6	Request (REQ)
7*	6	Response (RES)
8	10	Timestamps (TIME)
9	(varies)	Authentication (AUTH)
10-126	(varies)	UNASSIGNED (assignable by IANA)
127	(varies)	RFC 3692-style experiments (EXP)
128-191		RESERVED
192	(varies)	Encryption (UENC)
193-253		UNASSIGNED-UNSAFE (assignable by IANA)
254	(varies)	RFC 3692-style experiments (UEXP)
255		RESERVED

Options indicated by Kind values in the range 0..127 are known as SAFE options because they do not alter the UDP data payload and thus do not interfere with use of that data by legacy endpoints. Options indicated by Kind values in the range 192..254 are known as UNSAFE options because they do alter the UDP data payload and thus would interfere with legacy endpoints. UNSAFE option nicknames are expected to begin with "U", which should be avoided for safe option nicknames (see Section 23). Kind values 128-191 and 255 are RESERVED and not otherwise defined at this time.

>> RESERVED Kind values MUST NOT be assumed to be either SAFE nor UNSAFE until defined.

Although the FRAG option modifies the original UDP payload contents (i.e., is UNSAFE with respect to the original UDP payload), it is used only in subsequent fragments with zero UDP payloads, thus is SAFE in actual use, as discussed further in Section 9.4.

These options are defined in the following subsections. Options 0 and 1 use the same values as for TCP.

>> An endpoint supporting UDP options MUST support those marked with a "*" above: EOL, NOP, APC, FRAG, MDS, MRDS, REQ, and RES. This includes both recognizing and being able to generate these options if configured to do so. These are called "must-support" options.

>> An endpoint supporting UDP options MUST treat unsupported options in the UNSAFE range as terminating all option processing.

>> All other SAFE options (without a "*") MAY be implemented, and their use SHOULD be determined either out-of-band or negotiated, notably if needed to detect when options are silently ignored by legacy receivers.

>> Receivers supporting UDP options MUST silently ignore unknown SAFE options (i.e., in the same way a legacy receiver would). That includes options whose length does not indicate the specified value(s), as long as the length is not inherently invalid (i.e., smaller than 2 for the default and 4 for the extended formats).

>> UNSAFE options are used only in with the FRAG option, in a manner that prevents them from being silently ignored but passing the UDP payload to the user when not supported. This ensures their safe use in environments that might include legacy receivers (See Section 10).

>> Receivers supporting UDP options MUST silently drop all UDP options in a datagram containing an UNSAFE option when any UNSAFE option it contains is unknown. See Section 10 for further discussion of UNSAFE options.

>> Except for NOP, EXP, and UEXP, each option SHOULD NOT occur more than once in a single UDP datagram. If an option other than these occurs more than once, a receiver MUST interpret only the first instance of that option and MUST ignore all others.

>> EXP and UEXP MAY occur more than once, but SHOULD NOT occur more than once using the same ExID (see Sections 9.10 and 10.2).

>> Only the OCS and the AUTH and UENC options depend on the contents of the surplus area. AUTH and UENC are never used together, as UENC would serve both purposes. AUTH and UENC are always computed as if their hash and the OCS are zero; the OCS is always computed as if its contents are zero and after the AUTH or UENC hash has been computed. Future options MUST NOT be defined as having a value dependent on the contents of the surplus area. Otherwise, interactions between those values, the OCS, and the AUTH and UENC options could be unpredictable.

Receivers cannot generally treat unexpected option lengths as invalid, as this would unnecessarily limit future revision of options (e.g., defining a new APC that is defined by having a different length). The exception is only for lengths that imply a physical impossibility, e.g., smaller than two for conventional options and four for extended length options. Impossible lengths should indicate a malformed surplus area and all options silently discarded. Lengths other than those expected should result in safe options being ignored and skipped over, as with any other unknown safe option.

>> Option lengths MUST NOT exceed the IP length of the overall IP datagram. If this occurs, the options MUST be treated as malformed and all options dropped, and the event MAY be logged for diagnostics (logging SHOULD be rate limited).

>> "Must-support" options other than NOP and EOL MUST come before other options.

The requirement that must-support options come before others is intended to allow for endpoints to implement DOS protection, as discussed further in Section 22.

9. Safe UDP Options

Safe UDP options can be silently ignored by legacy receivers without affecting the meaning of the UDP user data. They stand in contrast to Unsafe options, which modify UDP user data in ways that render it unusable by legacy receivers (Section 10). The following subsections describe safe options defined in this document.

9.1. End of Options List (EOL)

The End of Options List (EOL, Kind=0) option indicates that there are no more options. It is used to indicate the end of the list of options without needing to use NOP options (see the following section) as padding to fill all available option space.

```
+-----+
| Kind=0 |
+-----+
```

Figure 7 UDP EOL option format

>> When the UDP options do not consume the entire surplus area, the last non-NOP option MUST be EOL.

>> NOPs SHOULD NOT be used as padding before the EOL option. As a one byte option, it need not be otherwise aligned.

>> All bytes in the surplus area after EOL MUST be set to zero on transmit.

>> Bytes after EOL in the surplus area MAY be checked as being zero on receipt but MUST be treated as zero regardless of their content and are not passed to the user (e.g., as part of the surplus area).

Requiring the post-option surplus area to be zero prevents side-channel uses of this area, requiring instead that all use of the surplus area be UDP options supported by both endpoints. It is useful to allow this area to be used for zero padding to increase the UDP datagram length without affecting the UDP user data length, e.g., for UDP DPLPMTUD (Section 4.1 of [Fa22]).

9.2. No Operation (NOP)

The No Operation (NOP, Kind=1) option is a one-byte placeholder, intended to be used as padding, e.g., to align multi-byte options along 16-bit, 32-bit, or 64-bit boundaries.

```
+-----+
| Kind=1 |
+-----+
```

Figure 8 UDP NOP option format

>> UDP packets SHOULD NOT use more than seven consecutive NOPs, i.e., to support alignment up to 8-byte boundaries. UDP packets SHOULD NOT use NOPs at the end of the options area as a substitute for EOL followed by zero-fill. NOPs are intended to assist with alignment, not as other padding or fill.

This issue is discussed further in Section 22.

9.3. Alternate Payload Checksum (APC)

The Alternate Payload Checksum (APC, Kind=2) option provides a stronger alternative to the checksum in the UDP header, using a 32-bit CRC of the conventional UDP user data payload only (excluding the IP pseudoheader, UDP header, and surplus area). It is an "alternate" to the UDP checksum that covers the user data - not to the OCS (the latter covers the surplus area only). Unlike the UDP checksum, APC does not include the IP pseudoheader or UDP header, thus it does not need to be updated by NATs when IP addresses or UDP

ports are rewritten. Its purpose is to detect user data errors that the UDP checksum, when used, might not detect.

A CRC32c has been chosen because of its ubiquity and use in other Internet protocols, including iSCSI and SCTP. The option contains the CRC32c in network standard byte order, as described in [RFC3385].

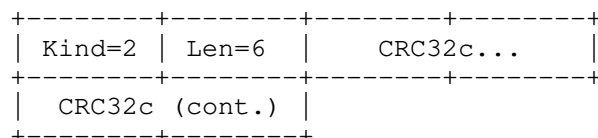


Figure 9 UDP APC option format

When present, the APC always contains a valid CRC checksum. There are no reserved values, including the value of zero. If the CRC is zero, this must indicate a valid checksum (i.e., it does not indicate that the APC is not used; instead, the option would simply not be included if that were the desired effect).

APC does not protect the UDP pseudoheader; only the current UDP checksum provides that protection (when used). APC cannot provide that protection because it would need to be updated whenever the UDP pseudoheader changed, e.g., during NAT address and port translation; because this is not the case, APC does not cover the pseudoheader.

>> UDP packets with incorrect APC checksums MUST be passed to the application by default, e.g., with a flag indicating APC failure.

Like all safe UDP options, APC needs to be silently ignored when failing by default, unless the receiver has been configured to do otherwise. Although all UDP option-aware endpoints support APC (being in the required set), this silently-ignored behavior ensures that option-aware receivers operate the same as legacy receivers unless overridden.

>> UDP packets with unrecognized APC lengths MUST be receive the same treatment as UDP packets with incorrect APC checksums.

Ensuring that unrecognized APC lengths are treated as incorrect checksums enables future variants of APC to be treated as APC-like.

9.4. Fragmentation (FRAG)

The Fragmentation (FRAG, Kind=3) option supports UDP fragmentation and reassembly, which can be used to transfer UDP messages larger than limited by the IP receive MTU (EMTU_R [RFC1122]). FRAG includes a copy of the same UDP transport ports in each fragment, enabling them to traverse Network Address (and port) Translation (NAT) devices, in contrast to the behavior of IP fragments. FRAG is typically used with the UDP MDS and MRDS options to enable more efficient use of large messages, both at the UDP and IP layers. FRAG is designed similar to the IPv6 Fragmentation Header [RFC8200], except that the UDP variant uses a 16-bit Offset measured in bytes, rather than IPv6's 13-bit Fragment Offset measured in 8-byte units. This UDP variant avoids creating reserved fields.

>> When FRAG is present, it SHOULD come as early as possible in the UDP options list.

>> When FRAG is present, the UDP user data MUST be empty. If the user data is not empty, all UDP options MUST be silently ignored and the user data received sent to the user.

Legacy receivers interpret FRAG messages as zero-length user data UDP packets (i.e., UDP Length field is 8, the length of just the UDP header), which would not affect the receiver unless the presence of the UDP packet itself were a signal (see Section 5 of [RFC8085]). In this manner, the FRAG option also helps hide UNSAFE options so they can be used more safely in the presence of legacy receivers.

The FRAG option has two formats; non-terminal fragments use the shorter variant (Figure 10) and terminal fragments use the longer (Figure 11). The latter includes stand-alone fragments, i.e., when data is contained in the FRAG option but reassembly is not required.

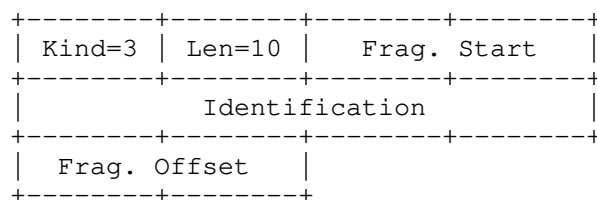


Figure 10 UDP non-terminal FRAG option format

In the non-terminal FRAG option format, Frag. Start indicates the location of the beginning of the fragment data, measured from the beginning of the UDP header of the fragment. The fragment data

follows the remainder of the UDP options and continues to the end of the IP datagram (i.e., the end of the surplus area). Those options are applied to this UDP fragment. Non-terminal fragments never have options after the fragment.

The Frag. Offset field indicates the location of this fragment relative to the original UDP datagram (prior to fragmentation), measured from the start of the original UDP datagram's UDP header.

The FRAG option does not need a "more fragments" bit because it provides the same indication by using the longer, 12-byte variant, as shown in Figure 11.

>> The FRAG option MAY be used on a single fragment, in which case the Frag. Offset would be zero and the option would have the 12-byte format.

>> Endpoints supporting UDP options MUST be capable of fragmenting and reassembling at least 2 fragments, for a total of at least 3,000 bytes (see MRDS in Section 9.6).

Use of the single fragment variant can be helpful in supporting use of UNSAFE options without undesirable impact to receivers that do not support either UDP options or the specific UNSAFE options.

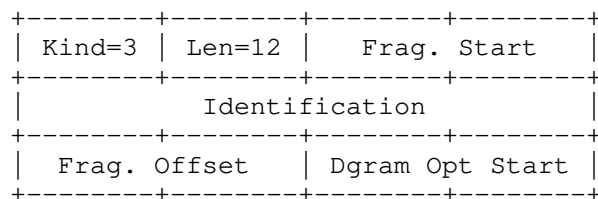


Figure 11 UDP terminal FRAG option format

The terminal FRAG option format adds a Datagram Option Start pointer, measured from the start of the original UDP datagram header, indicating the end of the reassembled data and the start of the surplus area after the original UDP datagram. In this variant, UDP options that apply to the reassembled datagram may occur after the terminal fragment data. UDP options that occur before the FRAG data are processed on the fragment; UDP options after the FRAG data are processed after reassembly, such that the reassembled data represents the original UDP user data. This allows either pre-reassembly or post-reassembly UDP option effects, such as using UENC on each fragment while also using TIME on the reassembled datagram for round-trip latency measurements.

>> During fragmentation, the UDP header checksum of each fragment remains constant and does not depend on the fragment data (which appears in the surplus area), because all fragments have a zero-length user data field.

The Fragment Offset is 16 bits and indicates the location of the UDP payload fragment in bytes from the beginning of the original unfragmented payload. The option Len field indicates whether there are more fragments (Len=10) or no more fragments (Len=12).

>> The Identification field is a 32-bit value that MUST be unique over the expected fragment reassembly timeout.

>> The Identification field SHOULD be generated in a manner similar to that of the IPv6 Fragment ID [RFC8200].

>> UDP fragments MUST NOT overlap.

Similar to IPv6 reassembly [RFC8200], if any of the fragments being reassembled overlap with any other fragments being reassembled for the same UDP packet, reassembly of that UDP packet must be abandoned and all the fragments that have been received for that UDP packet must be discarded, and no ICMP error messages should be sent.

It should be noted that fragments may be duplicated in the network. Instead of treating these exact duplicate fragments as overlapping fragments, an implementation may choose to detect this case and drop exact duplicate fragments while keeping the other fragments belonging to the same UDP packet.

UDP fragmentation relies on a fragment expiration timer, which can be preset or could use a value computed using the UDP Timestamp option.

>> The default UDP reassembly SHOULD be no more than 2 minutes.

>> UDP reassembly space SHOULD be limited to reduce the impact of DOS attacks on resource use.

>> UDP reassembly space limits SHOULD NOT be computed as a shared resource across multiple sockets, to avoid cross-socketpair DOS attacks.

>> Individual UDP fragments MUST NOT be forwarded to the user. The reassembled datagram is received only after complete reassembly, checksum validation, and continued processing of the remaining UDP options.

Any per-datagram UDP options, if used, follow the FRAG option in the final fragment and would be included in the reassembled UDP packet. Processing of those options would commence after reassembly. This is especially important for UNSAFE options, which are interpreted only after FRAG.

In general, UDP packets are fragmented as follows:

1. Create a UDP packet with data and UDP options, which we will call "D". Note that the UDP options treat the data area as UDP user data and thus must follow that data.

Process these UDP options before the rest of the fragmentation steps below. Note that the OCS value of the original packet SHOULD be zero if each fragment will have a non-zero OCS value (as will be the case if the UDP checksum is non-zero).

2. Identify the desired fragment size, which we will call "S". This value should take into account the path MTU (if known) and allow space for per-fragment options.
3. Fragment "D" into chunks of size no larger than "S"-10 each, with one final chunk no larger than "S"-12. Note that all the non-FRAG options in step #1 need not be limited to the terminal fragment, i.e., the Dgram Opt. Start pointer can indicate the start of the original surplus area anywhere in the reassembled data.
4. For each chunk of "D" in step #3, create a zero-data UDP packet followed by the word-aligned OCS, the FRAG option, and any additional UDP options, followed by the FRAG data chunk.

The last chunk includes the non-FRAG options noted in step #1 after the end of the FRAG data. These UDP options apply to the reassembled user data as a whole when received.

5. Process the pre-reassembly UDP options of each fragment.

Receivers reverse the above sequence. They process all received options in each fragment. When the FRAG option is encountered, the FRAG data is used in reassembly. After all fragments are received, the entire UDP packet is processed with any trailing UDP options applying to the reassembled user data.

9.5. Maximum Datagram Size (MDS)

The Maximum Datagram Size (MDS, Kind=4) option is a 16-bit hint of the largest unfragmented UDP packet that an endpoint believes can be

received. As with the TCP Maximum Segment Size (MSS) option [RFC793], the size indicated is the IP layer MTU decreased by the fixed IP and UDP headers only [RFC6691]. The space needed for IP and UDP options need to be adjusted by the sender when using the value indicated. The value transmitted is based on EMTU_R, the largest IP datagram that can be received (i.e., reassembled at the receiver) [RFC1122]. However, as with TCP, this value is only a hint at what the receiver believes; it does not indicate a known path MTU and thus MUST NOT be used to limit transmissions.

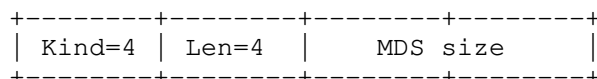


Figure 12 UDP MDS option format

The UDP MDS option MAY be used as a hint for path MTU discovery [RFC1191][RFC8201], but this may be difficult because of known issues with ICMP blocking [RFC2923] as well as UDP lacking automatic retransmission. It is more likely to be useful when coupled with IP source fragmentation or UDP fragmentation to limit the largest reassembled UDP message as indicated by MRDS (see Section 9.6), e.g., when EMTU_R is larger than the required minimums (576 for IPv4 [RFC791] and 1500 for IPv6 [RFC8200]). It can also be used with DPLPMTUD [RFC8899] to provide a hint to maximum DPLPMTU, though it MUST NOT prohibit transmission of larger UDP packets (or fragments) used as DPLPMTU probes.

9.6. Maximum Reassembled Datagram Size (MRDS)

The Maximum Reassembled Segment Size (MRDS, Kind=5) option is a 16-bit indicator of the largest reassembled UDP segment that can be received. MRDS is the UDP equivalent of IP's EMTU_R but the two are not related [RFC1122]. Using the FRAG option (Section 9.4), UDP packets can be transmitted as transport fragments, each in their own (presumably not fragmented) IP datagram and be reassembled at the UDP layer.

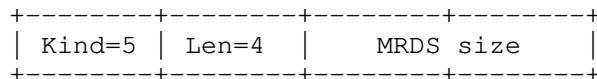


Figure 13 UDP MRDS option format

>> Endpoints supporting UDP options MUST support a local MRDS of at least 3,000 bytes.

9.7. Echo request (REQ) and echo response (RES)

The echo request (REQ, Kind=6) and echo response (RES, Kind=7) options provide a means for UDP options to be used to provide UDP packet-level acknowledgements. One such use is described as part of the UDP options variant of packetization layer path MTU discovery (PLPMTUD) [Fa22]. The options both have the format indicated in Figure 14, in which the token has no internal structure or meaning.

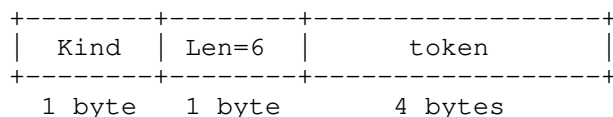


Figure 14 UDP REQ and RES options format

Each of these option kinds appears at most once in each UDP packet, as with other options. Note also that the FRAG option is not used when sending DPLPMTUD probes to determine a PLPMTU [Fa22].

9.8. Timestamps (TIME)

The Timestamp (TIME, Kind=8) option exchanges two four-byte unsigned timestamp fields. It serves a similar purpose to TCP's TS option [RFC7323], enabling UDP to estimate the round trip time (RTT) between hosts. For UDP, this RTT can be useful for establishing UDP fragment reassembly timeouts or transport-layer rate-limiting [RFC8085].

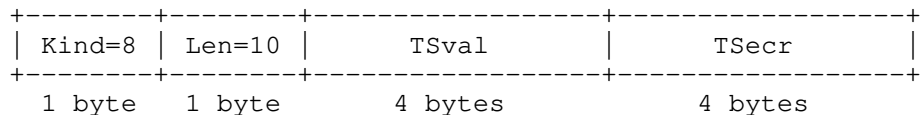


Figure 15 UDP TIME option format

TS Value (TSval) and TS Echo Reply (TSecr) are used in a similar manner to the TCP TS option [RFC7323]. On transmitted UDP packets using the option, TS Value is always set based on the local "time" value. Received TSval and TSecr values are provided to the application, which can pass the TSval value to be used as TSecr on UDP messages sent in response (i.e., to echo the received TSval). A received TSecr of zero indicates that the TSval was not echoed by the transmitter, i.e., from a previously received UDP packet.

>> TIME MAY use an RTT estimate based on nonzero Timestamp values as a hint for fragmentation reassembly, rate limiting, or other mechanisms that benefit from such an estimate.

>> an application MAY use TIME to compute this RTT estimate for further use by the user.

UDP timestamps are modeled after TCP timestamps and have similar expectations. In particular, they are expected to be:

- o Values are monotonic and non-decreasing except for anticipated number-space rollover events
- o Values should "increase" (allowing for rollover) according to a typical 'tick' time
- o A request is defined as TSval being non-zero and a reply is defined as TSecr being non-zero.
- o A receiver should always respond to a request with the highest TSval received (allowing for rollover), which is not necessarily the most recently received.

Rollover can be handled as a special case or more completely using sequence number extension [RFC9187], however zero values need to be avoided explicitly.

>> TIME values MUST NOT use zeros as valid time values, because they are used as indicators of requests and responses.

9.9. Authentication (AUTH)

The Authentication (AUTH, Kind=9) option is intended to allow UDP to provide a similar type of authentication as the TCP Authentication Option (TCP-AO) [RFC5925]. AUTH covers the UDP user data. AUTH supports NAT traversal in a similar manner as TCP-AO [RFC6978]. Figure 16 shows the UDP AUTH format, whose contents are identical to that of the TCP-AO option.

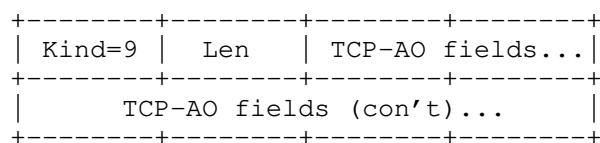


Figure 16 UDP AUTH option format

Like TCP-AO, AUTH is not negotiated in-band. Its use assumes both endpoints have populated Master Key Tuples (MKTs), used to exclude non-protected traffic.

TCP-AO generates unique traffic keys from a hash of TCP connection parameters. UDP lacks a three-way handshake to coordinate connection-specific values, such as TCP's Initial Sequence Numbers (ISNs) [RFC793], thus AUTH's Key Derivation Function (KDF) uses zeroes as the value for both ISNs. This means that the AUTH reuses keys when socket pairs are reused, unlike TCP-AO.

>> UDP packets with incorrect AUTH HMACs MUST be passed to the application by default, e.g., with a flag indicating AUTH failure.

Like all non-UNSAFE UDP options, AUTH needs to be silently ignored when failing. This silently-ignored behavior ensures that option-aware receivers operate the same as legacy receivers unless overridden.

In addition to the UDP user data (which is always included), AUTH can be configured to either include or exclude the surplus area, in a similar way as can TCP-AO can optionally exclude TCP options. When UDP options are covered, the OCS value and AUTH (and later, UENC) hash areas are zeroed before computing the AUTH hash. It is important to consider that options not yet defined might yield unpredictable results if not confirmed as supported, e.g., if they were to contain other hashes or checksums that depend on the surplus area contents. This is why such dependencies are not permitted except as defined for the OCS and the AUTH (and later, UENC) option.

Similar to TCP-AO-NAT, AUTH (and later, UENC) can be configured to support NAT traversal, excluding (by zeroing out) one or both of the UDP ports and corresponding IP addresses [RFC6978].

9.10. Experimental (EXP)

The Experimental option (EXP, Kind=127) is reserved for experiments [RFC3692]. Only one such value is reserved because experiments are expected to use an Experimental ID (ExIDs) to differentiate concurrent use for different purposes, using UDP ExIDs registered with IANA according to the approach developed for TCP experimental options [RFC6994].

Kind=127	Len	UDP ExID
(option contents, as defined)...		

Figure 17 UDP EXP option format

>> The length of the experimental option MUST be at least 4 to account for the Kind, Length, and the minimum 16-bit UDP ExID identifier (similar to TCP ExIDs [RFC6994]).

The UDP EXP option also includes an extended length format, where the option LEN is 255 followed by two bytes of extended length.

Kind=127	255	Extended Length
UDP ExID.	(option contents...)	

Figure 18 UDP EXP option format

Assigned UDP experimental IDs (ExIDs) assigned from a single registry managed by IANA (see Section 23). Assigned ExIDs can be used in either the EXP or UEXP options (see Section 10.2 for the latter).

10. UNSAFE Options

UNSAFE options are not safe to ignore and can be used unidirectionally or without soft-state confirmation of UDP option capability. They are always used only when the user data occurs inside a reassembled set of one or more UDP fragments, such that if UDP fragmentation is not supported, the enclosed UDP user data would be silently dropped anyway.

>> Applications using UNSAFE options SHOULD NOT also use zero-length UDP packets as signals, because they will arrive when UNSAFE options fail. Those that choose to allow such packets MUST account for such events.

>> UNSAFE options MUST be used only as part of UDP fragments, used either per-fragment or after reassembly.

>> Receivers supporting UDP options MUST silently drop the UDP user data of the reassembled datagram if any fragment or the entire

datagram includes an UNSAFE option whose UKind is not supported. Note that this still results in the receipt of a zero-length UDP datagram.

10.1. UNSAFE Encryption (UENC)

UNSAFE encryption (UENC, Kind=192) has the same format as AUTH (Section 9.9), except that it encrypts (modifies) the user data. It provides a similar encryption capability as TCP-AO-ENC, in a similar manner [Tol8]. Its fields, coverage, and processing are the same as for AUTH, except that UENC encrypts only the user data, although it can (optionally) depend on the surplus area (with certain fields zeroed, as per AUTH, e.g., providing authentication over the surplus area). Like AUTH, UENC can be configured to be compatible with NAT traversal.

10.2. UNSAFE Experimental (UEXP)

The UNSAFE Experimental option (UEXP, Kind=254) is reserved for experiments [RFC3692]. As with EXP, only one such UEXP value is reserved because experiments are expected to use an Experimental ID (ExIDs) to differentiate concurrent use for different purposes, using UDP ExIDs registered with IANA according to the approach developed for TCP experimental options [RFC6994].

Assigned ExIDs can be used with either the UEXP or EXP options.

11. Rules for designing new options

The UDP option Kind space allows for the definition of new options, however the currently defined options do not allow for arbitrary new options. The following is a summary of rules for new options and their rationales:

>> New options MUST NOT modify other option content.

>> New options MUST NOT depend on the content of other options.

>> UNSAFE options can both depend on and vary user data content because they are contained only inside UDP fragments and thus are processed only by UDP option capable receivers.

>> New options MUST NOT declare their order relative to other options, whether new or old.

>> At the sender, new options MUST NOT modify UDP packet content anywhere except within their option field, excepting only those

contained within the UNSAFE option; areas that need to remain unmodified include the IP header, IP options, the UDP user data, and the surplus area (i.e., other options).

>> Options MUST NOT be modified in transit. This includes those already defined as well as new options.

>> New options MUST NOT require or intend optionally for modification of any UDP options, including their new areas, in transit.

Note that only certain of the initially defined options violate these rules:

- o >> Only FRAG and UNSAFE options are permitted to modify the UDP body.

The following recommendation helps enable efficient zero-copy processing:

- o >> FRAG SHOULD be the first option, when present.

12. Option inclusion and processing

The following rules apply to option inclusion by senders and processing by receivers.

>> Senders MAY add any option, as configured by the API.

>> All "must-support" options MUST be processed by receivers, if present (presuming UDP options are supported at that receiver).

>> Non-"must-support" options MAY be ignored by receivers, if present, e.g., based on API settings.

>> All options MUST be processed by receivers in the order encountered in the options area.

>> All options except UNSAFE options MUST result in the UDP user data being passed to the application layer, regardless of whether all options are processed, supported, or succeed.

The basic premise is that, for options-aware endpoints, the sender decides what options to add and the receiver decides what options to handle. Simply adding an option does not force work upon a receiver, with the exception of the "must-support" options.

Upon receipt, the receiver checks various properties of the UDP packet and its options to decide whether to accept or drop the UDP packet and whether to accept or ignore some its options as follows (in order):

```
if the UDP checksum fails then
    silently drop the entire UDP packet (per RFC1122)
if the UDP checksum passes then
    if OCS != 0 and fails or is zero when UDP CS != 0 then
        deliver the UDP user data but ignore other options
        (this is required to emulate legacy behavior)
    if OCS is nonzero and passes or is zero then
        deliver the UDP user data after parsing
        and processing the rest of the options,
        regardless of whether each is supported or succeeds
        (again, this is required to emulate legacy behavior)
```

The design of the UNSAFE options as used only inside the FRAG area ensures that the resulting UDP data will be silently dropped in both legacy and options-aware receivers. Again, note that this still results in the delivery of a zero-length UDP packet.

Options-aware receivers can drop UDP packets with option processing errors via either an override of the default UDP processing or at the application layer.

I.e., all options are treated the same, in that the transmitter can add it as desired and the receiver has the option to require it or not. Only if it is required (e.g., by API configuration) should the receiver require it being present and correct.

I.e., for all options:

- o if the option is not required by the receiver, then UDP packets missing the option are accepted.
- o if the option is required (e.g., by override of the default behavior at the receiver) and missing or incorrectly formed, silently drop the UDP packet.
- o if the UDP packet is accepted (either because the option is not required or because it was required and correct), then pass the option with the UDP packet via the API.

Any options whose length exceeds that of the UDP packet (i.e., intending to use data that would have been beyond the surplus area) should be silently ignored (again to model legacy behavior).

13. UDP API Extensions

UDP currently specifies an application programmer interface (API), summarized as follows (with Unix-style command as an example) [RFC768]:

- o Method to create new receive ports
 - o E.g., `bind(handle, recvaddr(optional), recvport)`
- o Receive, which returns data octets, source port, and source address
 - o E.g., `recvfrom(handle, srcaddr, srcport, data)`
- o Send, which specifies data, source and destination addresses, and source and destination ports
 - o E.g., `sendto(handle, destaddr, destport, data)`

This API is extended to support options as follows:

- o Extend the method to create receive ports to include per-packet and per-fragment receive options that are required as indicated by the application. Datagrams not containing these required options MUST be silently dropped and MAY be logged. This includes a minimum datagram length, such that the options list ends in EOL and additional space is zero-filled as needed.
- o WG QUESTION: DO WE ALSO WANT A MIN FRAG SIZE? OR MAX?
- o Extend the receive function to indicate the per-packet options and their parameters as received with the corresponding received datagram. Note that per-fragment options are handled within the processing of each fragment.
- o WG QUESTION: SHOULD WE ACCUMULATE THOSE OPTIONS? OR DISCARD THEM?
- o Extend the send function to indicate the options to be added to the corresponding sent datagram. This includes indicating which options apply to individual fragments vs. which apply to the UDP packet prior to fragmentation, if fragmentation is enabled.

Examples of API instances for Linux and FreeBSD are provided in Appendix A, to encourage uniform cross-platform implementations.

14. UDP Options are for Transport, Not Transit

UDP options are indicated in the surplus area of the IP payload that is not used by UDP. That area is really part of the IP payload, not the UDP payload, and as such, it might be tempting to consider whether this is a generally useful approach to extending IP.

Unfortunately, the surplus area exists only for transports that include their own transport layer payload length indicator. TCP and SCTP include header length fields that already provide space for transport options by indicating the total length of the header area, such that the entire remaining area indicated in the network layer (IP) is transport payload. UDP-Lite already uses the UDP Length field to indicate the boundary between data covered by the transport checksum and data not covered, and so there is no remaining area where the length of the UDP-Lite payload as a whole can be indicated [RFC3828].

UDP options are intended for use only by the transport endpoints. They are no more (or less) appropriate to be modified in-transit than any other portion of the transport datagram.

UDP options are transport options. Generally, transport headers, options, and data are not intended to be modified in-transit. UDP options are no exception and here are specified as "MUST NOT" be altered in transit. However, the UDP option mechanism provides no specific protection against in-transit modification of the UDP header, UDP payload, or surplus area, except as provided by the OCS or the options selected (e.g., AUTH, or UENC).

15. UDP options vs. UDP-Lite

UDP-Lite provides partial checksum coverage, so that UDP packets with errors in some locations can be delivered to the user [RFC3828]. It uses a different transport protocol number (136) than UDP (17) to interpret the UDP Length field as the prefix covered by the UDP checksum.

UDP (protocol 17) already defines the UDP Length field as the limit of the UDP checksum, but by default also limits the data provided to the application as that which precedes the UDP Length. A goal of UDP-Lite is to deliver data beyond UDP Length as a default, which is why a separate transport protocol number was required.

UDP options do not use or need a separate transport protocol number because the data beyond the UDP Length offset (surplus data) is not provided to the application by default. That data is interpreted exclusively within the UDP transport layer.

UDP-Lite cannot support UDP options, either as proposed here or in any other form, because the entire payload of the UDP packet is already defined as user data and there is no additional field in which to indicate a surplus area for options. The UDP Length field in UDP-Lite is already used to indicate the boundary between user data covered by the checksum and user data not covered.

16. Interactions with Legacy Devices

It has always been permissible for the UDP Length to be inconsistent with the IP transport payload length [RFC768]. Such inconsistency has been utilized in UDP-Lite using a different transport number. There are no known systems that use this inconsistency for UDP [RFC3828]. It is possible that such use might interact with UDP options, i.e., where legacy systems might generate UDP datagrams that appear to have UDP options. The OCS provides protection against such events and is stronger than a static "magic number".

UDP options have been tested as interoperable with Linux, macOS, and Windows Cygwin, and worked through NAT devices. These systems successfully delivered only the user data indicated by the UDP Length field and silently discarded the surplus area.

One reported embedded device passes the entire IP datagram to the UDP application layer. Although this feature could enable application-layer UDP option processing, it would require that conventional UDP user applications examine only the UDP user data. This feature is also inconsistent with the UDP application interface [RFC768] [RFC1122].

It has been reported that Alcatel-Lucent's "Brick" Intrusion Detection System has a default configuration that interprets inconsistencies between UDP Length and IP Length as an attack to be reported. Note that other firewall systems, e.g., CheckPoint, use a default "relaxed UDP length verification" to avoid falsely interpreting this inconsistency as an attack.

17. Options in a Stateless, Unreliable Transport Protocol

There are two ways to interpret options for a stateless, unreliable protocol -- an option is either local to the message or intended to

affect a stream of messages in a soft-state manner. Either interpretation is valid for defined UDP options.

It is impossible to know in advance whether an endpoint supports a UDP option.

>> All UDP options other than UNSAFE ones MUST be ignored if not supported or upon failure (e.g., APC).

>> All UDP options that fail MUST result in the UDP data still being sent to the application layer by default, to ensure equivalence with legacy devices.

>> UDP options that rely on soft-state exchange MUST allow for message reordering and loss.

The above requirements prevent using any option that cannot be safely ignored unless it is hidden inside the FRAG area (i.e., UNSAFE options). Legacy systems also always need to be able to interpret the transport fragments as individual UDP packets.

18. UDP Option State Caching

Some TCP connection parameters, stored in the TCP Control Block, can be usefully shared either among concurrent connections or between connections in sequence, known as TCP Sharing [RFC9040]. Although UDP is stateless, some of the options proposed herein may have similar benefit in being shared or cached. We call this UCB Sharing, or UDP Control Block Sharing, by analogy. Just as TCB sharing is not a standard because it is consistent with existing TCP specifications, UCB sharing would be consistent with existing UDP specifications, including this one. Both are implementation issues that are outside the scope of their respective specifications, and so UCB sharing is outside the scope of this document.

19. Updates to RFC 768

This document updates RFC 768 as follows:

- o This document defines the meaning of the IP payload area beyond the UDP length but within the IP length as the surplus area used herein for UDP options.
- o This document extends the UDP API to support the use of UDP options.

20. Interactions with other RFCs (and drafts)

This document clarifies the interaction between UDP Length and IP length that is not explicitly constrained in either UDP or the host requirements [RFC768] [RFC1122].

Teredo extensions (TE) define use of a similar difference between these lengths for trailers [RFC6081]. TE defines the UDP length pointing beyond (larger) than the location indicated by the IP length rather than shorter (as used herein):

"..the IPv6 packet length (i.e., the Payload Length value in the IPv6 header plus the IPv6 header size) is less than or equal to the UDP payload length (i.e., the Length value in the UDP header minus the UDP header size)"

As a result, UDP options are not compatible with TE, but that is also why this document does not update TE. Additionally, it is not at all clear how TE operates, as it requires network processing of the UDP length field to understand the total message including TE trailers.

TE updates Teredo NAT traversal [RFC4380]. The NAT traversal document defined "consistency" of UDP length and IP length as:

"An IPv6 packet is deemed valid if it conforms to [RFC2460]: the protocol identifier should indicate an IPv6 packet and the payload length should be consistent with the length of the UDP datagram in which the packet is encapsulated."

IPv6 is clear on the meaning of this consistency, in which the pseudoheader used for UDP checksums is based on the UDP length, not inferred from the IP length, using the same text in the current specification [RFC8200]:

"The Upper-Layer Packet Length in the pseudo-header is the length of the upper-layer header and data (e.g., TCP header plus TCP data). Some upper-layer protocols carry their own length information (e.g., the Length field in the UDP header); for such protocols, that is the length used in the pseudo-header."

This document is consistent the UDP profile for Robust Header Compression (ROHC) [RFC3095], noted here:

"The Length field of the UDP header MUST match the Length field(s) of the preceding subheaders, i.e., there must not

be any padding after the UDP payload that is covered by the IP Length."

ROHC compresses UDP headers only when this match succeeds. It does not prohibit UDP headers where the match fails; in those cases, ROHC default rules (Section 5.10) would cause the UDP header to remain uncompressed. Upon receipt of a compressed UDP header, Section A.1.3 of that document indicates that the UDP length is "INFERRED"; in uncompressed packets, it would simply be explicitly provided.

This issue of handling UDP header compression is more explicitly described in more recent specifications, e.g., Sec. 10.10 of Static Context Header Compression [RFC8724].

21. Multicast Considerations

UDP options are primarily intended for unicast use. Using these options over multicast IP requires careful consideration, e.g., to ensure that the options used are safe for different endpoints to interpret differently (e.g., either to support or silently ignore) or to ensure that all receivers of a multicast group confirm support for the options in use.

22. Security Considerations

There are a number of security issues raised by the introduction of options to UDP. Some are specific to this variant, but others are associated with any packet processing mechanism; all are discussed in this section further.

The use of UDP packets with inconsistent IP and UDP Length fields has the potential to trigger a buffer overflow error if not properly handled, e.g., if space is allocated based on the smaller field and copying is based on the larger. However, there have been no reports of such vulnerability and it would rely on inconsistent use of the two fields for memory allocation and copying.

UDP options are not covered by DTLS (datagram transport-layer security). Despite the name, neither TLS [RFC8446] (transport layer security, for TCP) nor DTLS [RFC6347] (TLS for UDP) protect the transport layer. Both operate as a shim layer solely on the user data of transport packets, protecting only their contents. Just as TLS does not protect the TCP header or its options, DTLS does not protect the UDP header or the new options introduced by this document. Transport security is provided in TCP by the TCP Authentication Option (TCP-AO [RFC5925]) or in UDP by the Authentication (AUTH) option (Section 9.9) and UNSAFE Encryption

(UENC) option (Section 10). Transport headers are also protected as payload when using IP security (IPsec) [RFC4301].

UDP options use the TLV syntax similar to that of TCP. This syntax is known to require serial processing and may pose a DOS risk, e.g., if an attacker adds large numbers of unknown options that must be parsed in their entirety, as is the case for IPv6 [RFC8504].

>> Implementations concerned with the potential for this vulnerability MAY implement only the required UDP options and MAY also limit processing of TLVs, either in number of non-padding options or total length, or both. The number of non-zero TLVs allowed in such cases MUST be at least 8.

Because required options come first and at most once each (with the exception of NOPs, which should never need to come in sequences of more than seven in a row), this limits their DOS impact. Note that TLV formats for options does require serial processing, but any format that allows future options, whether ignored or not, could introduce a similar DOS vulnerability.

UDP security should never rely solely on transport layer processing of options. UNSAFE options are the only type that share fate with the UDP data, because of the way that data is hidden in the surplus area until after those options are processed. All other options default to being silently ignored at the transport layer but may be dropped either if that default is overridden (e.g., by configuration) or discarded at the application layer (e.g., using information about the options processed that are passed along with the UDP packet).

UDP fragmentation introduces its own set of security concerns, which can be handled in a manner similar to IP reassembly or TCP segment reordering [CERT18]. In particular, the number of UDP packets pending reassembly and effort used for reassembly is typically limited. In addition, it may be useful to assume a reasonable minimum fragment size, e.g., that non-terminal fragments should never be smaller than 500 bytes.

23. IANA Considerations

Upon publication, IANA is hereby requested to create a new registry for UDP Option Kind numbers, similar to that for TCP Option Kinds. Initial values of this registry are as listed in Section 8. Additional values in this registry are to be assigned from the UNASSIGNED values in Section 8 by IESG Approval or Standards Action

[RFC8126]. Those assignments are subject to the conditions set forth in this document, particularly (but not limited to) those in Section 11.

Although option nicknames are not used in-band, IANA should require UNSAFE safe option values to commence with the letter "U" and avoid that letter as commencing safe options.

Upon publication, IANA is hereby requested to create a new registry for UDP Experimental Option Experiment Identifiers (UDP ExIDs) for use in a similar manner as TCP ExIDs [RFC6994]. UDP ExIDs can be used in either (or both) the EXP or UEXP options. This registry is initially empty. Values in this registry are to be assigned by IANA using first-come, first-served (FCFS) rules [RFC8126]. Options using these ExIDs are subject to the same conditions as new options, i.e., they too are subject to the conditions set forth in this document, particularly (but not limited to) those in Section 11.

24. References

24.1. Normative References

- [Fa22] Fairhurst, G., T. Jones, "Datagram PLPMTUD for UDP Options," draft-ietf-tsvwg-udp-options-dplpmtud, Feb. 2022.
- [RFC768] Postel, J., "User Datagram Protocol," RFC 768, August 1980.
- [RFC791] Postel, J., "Internet Protocol," RFC 791, Sept. 1981.
- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts -- Communication Layers," RFC 1122, Oct. 1989.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words," RFC 2119, May 2017.

24.2. Informative References

- [Fa18] Fairhurst, G., T. Jones, R. Zullo, "Checksum Compensation Options for UDP Options", draft-fairhurst-udp-options-cco, Oct. 2018.

- [Hil15] Hildebrand, J., B. Trammel, "Substrate Protocol for User Datagrams (SPUD) Prototype," draft-hildebrand-spud-prototype-03, Mar. 2015.
- [RFC793] Postel, J., "Transmission Control Protocol" RFC 793, September 1981.
- [RFC1071] Braden, R., D. Borman, C. Partridge, "Computing the Internet Checksum," RFC 1071, Sept. 1988.
- [RFC1191] Mogul, J., S. Deering, "Path MTU discovery," RFC 1191, November 1990.
- [RFC2923] Lahey, K., "TCP Problems with Path MTU Discovery," RFC 2923, September 2000.
- [RFC3095] Bormann, C. (Ed), et al., "RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed," RFC 3095, July 2001.
- [RFC3385] Sheinwald, D., J. Satran, P. Thaler, V. Cavanna, "Internet Protocol Small Computer System Interface (iSCSI) Cyclic Redundancy Check (CRC)/Checksum Considerations," RFC 3385, Sep. 2002.
- [RFC3692] Narten, T., "Assigning Experimental and Testing Numbers Considered Useful," RFC 3692, Jan. 2004.
- [RFC3828] Larzon, L-A., M. Degermark, S. Pink, L-E. Jonsson (Ed.), G. Fairhurst (Ed.), "The Lightweight User Datagram Protocol (UDP-Lite)," RFC 3828, July 2004.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, Dec. 2005.
- [RFC4340] Kohler, E., M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, March 2006.
- [RFC4380] Huitema, C., "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)," RFC 4380, Feb. 2006.
- [RFC4960] Stewart, R. (Ed.), "Stream Control Transmission Protocol", RFC 4960, September 2007.
- [RFC5925] Touch, J., A. Mankin, R. Bonica, "The TCP Authentication Option," RFC 5925, June 2010.

- [RFC6081] Thaler, D., "Teredo Extensions," RFC 6081, Jan 2011.
- [RFC6347] Rescorla, E., N. Modadugu, "Datagram Transport Layer Security Version 1.2," RFC 6347, Jan. 2012.
- [RFC6691] Borman, D., "TCP Options and Maximum Segment Size (MSS)," RFC 6691, July 2012.
- [RFC6935] Eubanks, M., P. Chimento, M. Westerlund, "IPv6 and UDP Checksums for Tunneled Packets," RFC 6935, April 2013.
- [RFC6978] Touch, J., "A TCP Authentication Option Extension for NAT Traversal", RFC 6978, July 2013.
- [RFC6994] Touch, J., "Shared Use of Experimental TCP Options," RFC 6994, Aug. 2013.
- [RFC7323] Borman, D., R. Braden, V. Jacobson, R. Scheffenegger (Ed.), "TCP Extensions for High Performance," RFC 7323, Sep. 2014.
- [RFC8085] Eggert, L., G. Fairhurst, G. Shepherd, "UDP Usage Guidelines," RFC 8085, Feb. 2017.
- [RFC8126] Cotton, M., B. Leiba, T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs," RFC 8126, June 2017.
- [RFC8200] Deering, S., R. Hinden, "Internet Protocol Version 6 (IPv6) Specification," RFC 8200, Jul. 2017.
- [RFC8201] McCann, J., S. Deering, J. Mogul, R. Hinden (Ed.), "Path MTU Discovery for IP version 6," RFC 8201, Jul. 2017.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018.
- [RFC8504] Chown, T., J. Loughney, T. Winters, "IPv6 Node Requirements," RFC 8504, Jan. 2019.
- [RFC8724] Minaburo, A., L. Toutain, C. Gomez, D. Barthel, JC., "SCHC: Generic Framework for Static Context Header Compression and Fragmentation," RFC 8724, Apr. 2020.
- [RFC8899] Fairhurst, G., T. Jones, M. Tuxen, I. Rungeler, T. Volker, "Packetization Layer Path MTU Discovery for Datagram Transports," RFC 8899, Sep. 2020.

- [RFC9040] Touch, J., M. Welzl, S. Islam, "TCP Control Block Interdependence," RFC 9040, Jul. 2021.
- [RFC9187] Touch, J., "Sequence Number Extension for Windowed Protocols," RFC 9187, Jan. 2022.
- [CERT18] CERT Coordination Center, "TCP implementations vulnerable to Denial of Service," Vulnerability Note VU 962459, Software Engineering Institute, CMU, 2018, <https://www.kb.cert.org/vuls/id/962459>.
- [To18] Touch, J., "A TCP Authentication Option Extension for Payload Encryption," draft-touch-tcp-ao-encrypt, Jul. 2018.

25. Acknowledgments

This work benefitted from feedback from Erik Auerswald, Bob Briscoe, Ken Calvert, Ted Faber, Gorry Fairhurst (including OCS for misbehaving middlebox traversal), C. M. Heard (including combining previous FRAG and LITE options into the new FRAG), Tom Herbert, Mark Smith, and Raffaele Zullo, as well as discussions on the IETF TSVWG and SPUD email lists.

This work was partly supported by USC/ISI's Postel Center.

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Joe Touch
Manhattan Beach, CA 90266 USA

Phone: +1 (310) 560-0334
Email: touch@strayalpha.com

Appendix A. Implementation Information

The following information is provided to encourage interoperable API implementations.

System-level variables (sysctl):

Name	default	meaning
net.ipv4.udp_opt	0	UDP options available
net.ipv4.udp_opt_ocs	1	Default use OCS
net.ipv4.udp_opt_apc	0	Default include APC
net.ipv4.udp_opt_frag	0	Default fragment
net.ipv4.udp_opt_mds	0	Default include MDS
net.ipv4.udp_opt_mrds	0	Default include MRDS
net.ipv4.udp_opt_req	0	Default include REQ
net.ipv4.udp_opt_resp	0	Default include RES
net.ipv4.udp_opt_time	0	Default include TIME
net.ipv4.udp_opt_auth	0	Default include AUTH
net.ipv4.udp_opt_exp	0	Default include EXP
net.ipv4.udp_opt_uenc	0	Default include UENC
net.ipv4.udp_opt_uexp	0	Default include UEXP

Socket options (sockopt), cached for outgoing datagrams:

Name	meaning
UDP_OPT	Enable UDP options (at all)
UDP_OPT_OCS	Use UDP OCS
UDP_OPT_APC	Enable UDP APC option
UDP_OPT_FRAG	Enable UDP fragmentation
UDP_OPT_MDS	Enable UDP MDS option
UDP_OPT_MRDS	Enable UDP MRDS option
UDP_OPT_REQ	Enable UDP REQ option
UDP_OPT_RES	Enable UDP RES option
UDP_OPT_TIME	Enable UDP TIME option
UDP_OPT_AUTH	Enable UDP AUTH option
UDP_OPT_EXP	Enable UDP EXP option
UDP_OPT_UENC	Enable UDP UENC option
UDP_OPT_UEXP	Enable UDP UEXP option

Send/sendto parameters:

Connection parameters (per-socketpair cached state, part UCB):

Name	Initial value

opts_enabled	net.ipv4.udp_opt
ocs_enabled	net.ipv4.udp_opt_ocs

The following option is included for debugging purposes, and MUST NOT be enabled otherwise.

System variables

net.ipv4.udp_opt_junk 0

System-level variables (sysctl):

Name	default	meaning

net.ipv4.udp_opt_junk	0	Default use of junk

Socket options (sockopt):

Name	params	meaning

UDP_JUNK	-	Enable UDP junk option
UDP_JUNK_VAL	fillval	Value to use as junk fill
UDP_JUNK_LEN	length	Length of junk payload in bytes

Connection parameters (per-socketpair cached state, part UCB):

Name	Initial value

junk_enabled	net.ipv4.udp_opt_junk
junk_value	0xABCD
junk_len	4

6man
Internet-Draft
Intended status: Standards Track
Expires: April 15, 2019

J. Leddy
Unaffiliated
R. Bonica
Juniper Networks
I. Lubashev
Akamai Technologies
October 12, 2018

IPv6 Packet Truncation
draft-leddy-6man-truncate-05

Abstract

This document defines IPv6 packet truncation procedures. These procedures make Path MTU Discovery (PMTUD) more reliable. Upper-layer protocols can leverage these procedures in order to take advantage of large MTUs.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 15, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Requirements Language	5
3. Operational Considerations	5
4. IPv6 Destination Options	6
4.1. The Truncation Eligible Option	6
4.2. The Truncated Packet Option	7
5. Reference Topology	8
6. Truncation Procedures	8
7. Additional Truncation Considerations	10
8. Backwards Compatibility	10
9. Checksum Considerations	11
10. Invalid Packet Types	11
11. Network Considerations	12
12. Encapsulating Security Payload Considerations	12
13. Extension Header Considerations	13
14. Security Considerations	13
15. IANA Considerations	14
16. Acknowledgements	14
17. References	14
17.1. Normative References	14
17.2. Informative References	15
Authors' Addresses	15

1. Introduction

An Internet path connects a source node to a destination node. A path can contain links and routers.

Each link is constrained by the number of bytes that it can convey in a single IP packet. This constraint is called the link Maximum Transmission Unit (MTU). IPv6 [RFC8200] requires every link to have an MTU of 1280 bytes or greater. This value is called IPv6 minimum link MTU.

Likewise, each Internet path is constrained by the number of bytes that it can convey in a single IP packet. This constraint is called the Path MTU (PMTU). For any given path, the PMTU is equal to the smallest of its link MTUs.

IPv6 allows fragmentation at the source node only. If an IPv6 source node sends a packet whose length exceeds the PMTU, an intermediate node will discard the packet. In order to prevent this, IPv6 nodes can either:

- o Refrain from sending packets whose length exceeds the IPv6 minimum link MTU.
- o Maintain a running estimate of the PMTU and refrain from sending packets whose length exceeds that estimate.

In order to maintain a running estimate of the PMTU, IPv6 nodes can execute Path MTU Discovery (PMTUD) [RFC8201] procedures. In these procedures, the source node produces an initial PMTU estimate. This initial estimate equals the MTU of the first link along the path to the destination. It can be greater than the actual PMTU.

Having produced an initial PMTU estimate, the source node sends packets to the destination node. If one of these packets is larger than the actual PMTU, an intermediate node will not be able to forward the packet through the next link along the path. Therefore, the intermediate node discards the packet and sends an Internet Control Message Protocol (ICMP) [RFC4443] Packet Too Big (PTB) message to the source node. The ICMP PTB message indicates the MTU of the link through which the packet could not be forwarded. The source node uses this information to refine its PMTU estimate.

PMTUD relies on the network to deliver ICMP PTB messages from the intermediate node to the source node. If the network cannot deliver these messages, a persistent black hole can develop. In this scenario, the source node sends a packet whose length exceeds the PMTU. An intermediate node discards the packet and sends an ICMP PTB message to the source. However, the network cannot deliver the ICMP PTB message to the source. Therefore, the source node does not update its PMTU estimate and it continues to send packets whose length exceeds the PMTU. The intermediate node discards these packets and sends more ICMP PTB messages to the source. These ICMP PTB messages are lost, exactly as previous ICMP PTB messages were lost.

In some operational scenarios (Section 3), networks cannot deliver ICMP PTB messages from an intermediate node to the source node. Therefore, enhanced procedures are required.

This document defines IPv6 packet truncation procedures. When an IPv6 source node originates a packet, it executes the following procedure:

- o Mark the packet as being eligible for truncation.
- o Forward the packet towards its destination.

If an intermediate node cannot forward the packet because of an MTU issue, it executes the following procedure:

- o Detect that the packet is eligible for truncation.
- o Send an ICMP PTB message to the source node, with the MTU field indicating the MTU of the link through which the packet could not be forwarded.
- o Truncate the packet.
- o Mark the packet as being truncated.
- o Update the packet's upper-layer checksum (if possible).
- o Forward the packet towards its destination.

When the destination node receives the packet, it executes the following procedure:

- o Detect that the packet has been truncated.
- o Send an ICMP PTB message to the source node, with the MTU field indicating the length of the truncated packet.
- o Discard the packet.

Both ICMP PTB messages, mentioned above, contain MTU information that the source node can use to refine its PMTU estimate.

The procedures described herein prevent incomplete (i.e., truncated) data from being delivered to upper-layer protocols. While IPv6 packet truncation may facilitate new upper-layer procedures, upper-layer procedures are beyond the scope of this document.

The procedures described herein make PMTUD more reliable by increasing the probability that the source node will receive ICMP PTB feedback from a downstream device. Even when the network cannot deliver ICMP PTB messages from an intermediate router to a source node, it may be able to deliver an ICMP PTB messages from the destination node to the source node.

However, the procedures described herein do not make PMTUD one hundred per cent reliable. In some operational scenarios, the network cannot deliver any ICMP messages to the source node, regardless of their origin.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Operational Considerations

The packet truncation procedures described herein make PMTUD more resilient when:

- o The network can deliver ICMP messages from the destination node to the source node.
- o The network cannot deliver ICMP messages from an intermediate node to the source node.

The following are common operational scenarios in which packet truncation procedures can make PMTUD more resilient:

- o The destination node has a viable route to the source node, but the intermediate node does not.
- o The source node is protected by a firewall that administratively blocks all packets except for those from specified subnetworks. The destination node resides in one of the specified subnetworks, but the intermediate node does not.
- o The source address of the original packet (i.e., the packet that elicited the ICMP message) was an anycast address. Therefore, the destination address of the ICMP message is the same anycast address. In this case, an ICMP message from the destination node is likely to be delivered to the correct anycast instance. By contrast, an ICMP message from an intermediate node is less likely to be delivered to the correct anycast instance.

Packet truncation procedures do not make PMTUD more resilient when the network cannot reliably deliver any ICMP messages to the source node. The following are operational scenarios where the network cannot reliably deliver any ICMP PTB messages to the source node:

- o The source node is protected by a firewall that administratively blocks all ICMP messages.

- o The source node is an anycast instance served by a load-balancer as defined in [RFC7690]. The load-balancer does not implement the mitigations defined in [RFC7690].

4. IPv6 Destination Options

This document defines the following IPv6 Destination options:

4.1. The Truncation Eligible Option

The Truncation Eligible option indicates that the packet is eligible for truncation. It also indicates that the packet has not been truncated.

The Truncation Eligible option contains the following fields:

- o Option Type - Truncation Eligible option. Value TBD by IANA. See Notes below.
- o Opt Data Len - Length of Option Data, measured in bytes. MUST be equal to 0.

IPv6 packets that include the Fragment header MUST NOT include the Truncation Eligible option.

IPv6 packets whose length is less than the IPv6 minimum link MTU SHOULD NOT include the Truncation Eligible option.

The IPv6 Hop-by-hop Options header SHOULD NOT include the Truncation Eligible option.

The IPv6 Destination Options header:

- o MAY include a single instance of the Truncation Eligible option.
- o SHOULD NOT include multiple instances of the Truncation Eligible option.
- o MUST NOT include both the Truncation Eligible option and the Truncated Packet option (Section 4.2).

NOTE 1: According to [RFC8200], the highest-order two bits of the Option Type (i.e., the "act" bits) specify the action taken by a processing node that does not recognize Option Type. The required action is skip over this option and continue processing the header. Therefore, IANA is requested to assign this Option Type with "act" bits "00".

NOTE 2: According to [RFC8200], the third-highest-order bit (i.e., the "chg" bit) of the Option Type specifies whether Option Data can change on route to the packet's destination. Because this option contains no Option Data, IANA can assign this Option Type without regard to the "chg" bit.

4.2. The Truncated Packet Option

The Truncated Packet option indicates that the packet has been truncated and is eligible for further truncation.

The Truncated Packet option contains the following fields:

- o Option Type - Truncated Packet option. Value TBD by IANA. See Notes below.
- o Opt Data Len - Length of Option Data, measured in bytes. MUST be equal to 0.

IPv6 packets that include the Fragment header MUST NOT include the Truncated Packet option.

IPv6 packets whose length is less than the IPv6 minimum link MTU MUST NOT include the Truncated Packet option.

The IPv6 Hop-by-hop Options header SHOULD NOT include the Truncated Packet option.

The IPv6 Destination Options:

- o MAY include a single instance of the Truncated Packet option.
- o SHOULD NOT include multiple instances of the Truncated Packet option.
- o MUST NOT include both the Truncated Packet option and the Truncation Eligible option.

NOTE 1: According to [RFC8200], the highest-order two bits of the Option Type (i.e., the "act" bits) specify the action taken by a processing node that does not recognize Option Type. The required action is to discard the packet and send an ICMP Parameter Problem, Code 2, message to the packet's Source Address, pointing to the unrecognized Option Type. Therefore, IANA is requested to assign this Option Type with "act" bits "10".

NOTE 2: According to [RFC8200], the third-highest-order bit (i.e., the "chg" bit) of the Option Type specifies whether Option Data of

that option can change on route to the packet's destination. Because this option contains no Option Data, IANA can assign this Option Type without regard to the "chg" bit.

5. Reference Topology

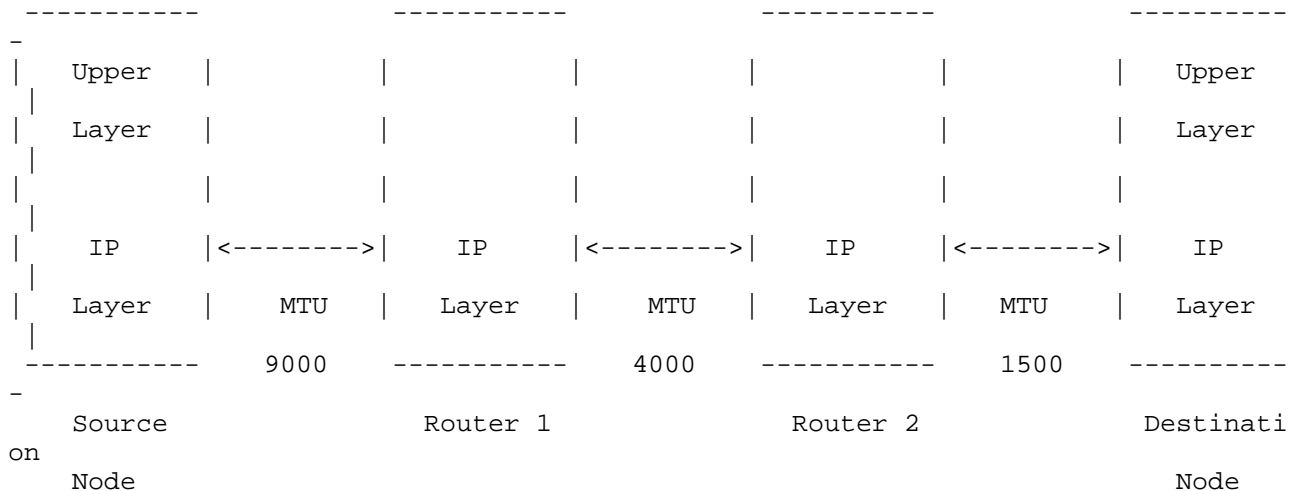


Figure 1: Reference Topology

Figure 1 depicts a network that contains a Source Node, intermediate nodes (i.e., Router 1, Router 2), and a Destination Node. The link that connects the Source Node to Router 1 has an MTU of 9000 bytes. The link that connects Router 1 to Router 2 has an MTU of 4000 bytes, and the link that connects Router 2 to the Destination Node has an MTU of 1500 bytes. The PMTU between the Source Node and the Destination Node is 1500 bytes.

This topology is used in examples throughout the document.

6. Truncation Procedures

In the Reference Topology (Figure 1), the Source Node produces an initial estimate of the PMTU between itself and the Destination Node. This initial estimate equals the MTU of the first link on the path to the Destination Node (e.g., 9000 bytes).

The Source Node refrains from sending packets whose length exceeds the above-mentioned estimate. However, the above-mentioned estimate is significantly larger than the actual PMTU (1500 bytes). Therefore, the Source Node may send packets whose length exceeds the actual PMTU.

At some time in the future, an upper-layer protocol on the Source Node causes the IP layer to emit a packet. The packet contains a Destination Options header and the Destination Options header contains a Truncation Eligible option. The total packet length, including all headers and the payload, is 1350 bytes. Because the total packet length is less than the actual PMTU, this packet can be

delivered to the Destination Node without encountering any MTU issues.

The IP layer on the Source Node forwards the packet to the Router 1, Router 1 forwards the packet to Router 2, and the Router 2 forwards the packet to the Destination Node. The IP layer on the Destination Node examines the Destination Options header and finds the Truncation Eligible option. The Truncation Eligible option requires no action by the Destination Node. Therefore, the Destination Node processes the next header and delivers the packet to an upper-layer protocol.

Subsequently, the same upper-layer protocol on the Source Node causes the IP layer to emit another packet. This packet is identical to the first, except that the total packet length is 2000 bytes. Because the packet length is greater than the actual PMTU, this packet cannot be delivered without encountering an MTU issue.

The IP layer on the source node forwards the packet to Router 1. Router 1 forwards the packet to Router 2, but the Router 2 cannot forward the packet because its length exceeds the MTU of the next link in the path (i.e., 1500 bytes). Because an MTU issue has been encountered, Router 2 examines the Destination Options header, searching for either a Truncation Eligible option or a Truncated Packet option. (Normally, the Router 2 would ignore the Destination Options header).

Because Router 2 finds one of the above-mentioned options, it:

- o Sends an ICMP PTB message to the Source Node. The ICMP PTB message contains an MTU field indicating the MTU of the next link in the path (i.e. 1500 bytes).
- o Truncates the packet, so that its total length equals the MTU of the next link in the path.
- o Updates the IPv6 Payload Length.
- o Overwrites all instances of the Truncation Eligible option with a Truncated Packet option.
- o Updates the upper-layer checksum (if possible)
- o Forwards the packet to the Destination Node.

The IP layer on the Destination Node receives the packet and examines the Destination Options header. Because it finds the Truncated Packet option, it discards the packet and sends an ICMP PTB message

to the Source Node. The MTU field in the ICMP PTB message represents the length of the received packet.

When the Source Node receives the ICMP PTB message, it updates its PMTU estimate, as per [RFC8201].

7. Additional Truncation Considerations

A packet can be truncated multiple times. In the Reference Topology (Figure 1), assume that the Source Node sends a 5000 byte packet to the Destination Node. Using the procedures described in Section 6, Router 1 truncates this packet to 4000 bytes and Router 2 truncates it again, to 1500 bytes.

A truncated packet MUST contain the basic IPv6 header, all extension headers and the first upper-layer header. When an intermediate node cannot forward a packet due to MTU issues, and the total length of the basic IPv6 header, all extension headers, and first upper-layer header exceeds the MTU of the next link in the path, the intermediate node MUST discard the packet and send an ICMP PTB message to the source node. It MUST NOT truncate the packet.

A truncated packet MUST NOT include the Fragment header. When an intermediate node cannot forward a packet due to MTU issues, and the packet contains a Fragment header, the intermediate node MUST discard the packet and send an ICMP PTB message to the source node. It MUST NOT truncate the packet.

A truncated packet must have a total length that is greater than or equal to the IPv6 minimum link MTU.

8. Backwards Compatibility

Section 6 of this document assumes that all nodes recognize the Truncation Eligible and Truncated Packet options. This section explores backwards compatibility issues, where one or more nodes do not recognize the above-mentioned options.

An intermediate node that does not recognize the above-mentioned options behaves exactly as described in [RFC8200]. When it receives a packet that does not cause an MTU issue, it processes the packet. When it receives a packet that causes an MTU issue, it discards the packet and sends an ICMP PTB message to the source node. In neither case does the intermediate node examine the Destination Options header or truncate the packet.

A destination node that does not recognize the Truncation Eligible option also behaves exactly as described in [RFC8200]. When it

receives a packet that contains the Truncation Eligible option, its behavior is determined by the highest-order two bits of the Option Type (i.e., the "act" bits). Because the "act" bits are equal to "00", the destination node skips over the option and continues to process the packet. This is exactly what the destination node would have done if it had recognized the Truncation Eligible option.

A destination node that does not recognize the Truncated Packet option also behaves exactly as described in [RFC8200]. When it receives a packet that contains the Truncated Packet option, its behavior is determined by the highest-order two bits of the Option Type (i.e., the "act" bits). Because the "act" bits are equal to "10", the destination node discards the packet and sends an ICMP Parameter Problem, Code 2, message to the packet's Source Address, pointing to the Truncated Packet option. The destination node does not emit an ICMP PTB message.

The source node takes appropriate action when it receives the ICMP Parameter Problem message.

9. Checksum Considerations

When an intermediate node truncates a packet, it SHOULD update the upper-layer checksum, if possible. This is desirable because it increases the probability that the truncated packet will be delivered to the destination node.

Middleboxes residing downstream of the intermediate node may attempt to validate the upper-layer checksum. If validation fails, they may discard the packet without sending an ICMP message.

10. Invalid Packet Types

The following packet types are invalid:

- o Packets that contain the Fragment header and the Truncation Eligible option.
- o Packets that contain the Fragment header and the Packet Truncated option.
- o Packets that contain the Truncation Eligible option and the Packet Truncated option.
- o Packets that specify an Option Data Length greater than 0 in the Truncation Eligible option.

- o Packets that specify an Option Data Length greater than 0 in the Truncated Packet option.
- o Packets that have a total length less than the IPv6 minimum link MTU and contain the Packet Truncated option.

If an intermediate node cannot forward one of the above-mentioned packets because of an MTU issue, its behavior is as described in [RFC8200]. The intermediate node discards the packet and sends an ICMP PTB message to the source node. It does not truncate or forward the packet.

When the destination node receives one of the above-mentioned packets, it MUST:

- o Discard the packet
- o Send an ICMP Parameter Problem, Code 2, message to the packet's Source Address, pointing to the first invalid option.

The destination node MUST NOT send an ICMP PTB message.

11. Network Considerations

The procedures described herein rely upon the networks ability:

- o To convey packets that contain destination options from the source node to the destination node.
- o To convey ICMP Parameter Problem messages in the reverse direction.

Operational experience [RFC7872] reveals that a significant number of networks drop packets that contain IPv6 destination options. Likewise, many networks drop ICMP Parameter Problem messages.

[I-D.bonica-6man-unrecognized-opt] describes procedures that upper-layer protocols can execute to verify that the above-mentioned requirements are satisfied. Upper-layer protocols can execute these procedures before emitting packets that contain the Truncation Eligible option.

12. Encapsulating Security Payload Considerations

An IPv6 packet can contain both:

- o An Encapsulating Security Payload (ESP) [RFC4303] header.

- o Truncation options (i.e., the Truncation Eligible or Truncated Packet options).

In this case, the packet MUST contain a Destination Options header that precedes the ESP. That Destination Options header contains the truncation options and is not protected by the ESP. The packet MAY also contain another Destination Options header that follows the ESP. That Destination Options header is protected by the ESP and MUST NOT contain the truncation options.

As per RFC 4303, a packet can contain two Destination Options headers one preceding the ESP and one following the ESP.

13. Extension Header Considerations

According to [RFC8200], the following IPv6 extension headers can contain options:

- o The Hop-by-hop Options header.
- o The Destination Options header.

The Hop-by-hop option can be examined by each node along the path to a packet's destination. Destination options are examined by the destination node only. However, [RFC2473] provides a precedent for intermediate nodes examining the Destination options on an exception basis. (See the Tunnel Encapsulation Limit.)

The truncation options described herein are examined by:

- o Intermediate nodes, on an exception basis (i.e, when the packet cannot be forwarded due to MTU issues).
- o The Destination node.

Therefore, the above-mentioned options can be processed most efficiently when they are contained by the Destination Option header. When contained by the Destination Options header, the above-mentioned options are examined by intermediate nodes on an exception basis, only when they are relevant. If contained by the Hop-by-hop Options header, they are always examined by intermediate nodes, even when they are irrelevant.

14. Security Considerations

PMTUD is vulnerable to ICMP PTB forgery attacks. The procedures described herein do nothing to mitigate that vulnerability.

The procedures described herein are susceptible to a new variation on that attack, in which an attacker forges a truncated packet. In this case, the attackers cause the Destination Node to produce an ICMP PTB message on their behalf. To some degree, this vulnerability is mitigated, because the Destination Node will not emit an ICMP PTB message in response to a truncated packet whose length is less than the IPv6 minimum link MTU.

In order to mitigate denial of service attacks, intermediate nodes MUST rate limit the number of packets that they truncate per second.

15. IANA Considerations

IANA is requested to allocate the following codepoints from the Destination Options and Hop-by-hop Options registry (<https://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml#ipv6-parameters-2>).

- o Truncation Eligible ("act-bits" are "00". "chg-bit" can be either 0 or 1.)
- o Truncated Packet ("act-bits" are "10". "chg-but can be either 0 or 1.)

16. Acknowledgements

Special thanks to Mike Heard, Geoff Huston, Joel Jaeggli, Tom Jones, Andy Smith, Jinmei Tatuya, and Reji Thomas who reviewed and commented on this document.

17. References

17.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <<https://www.rfc-editor.org/info/rfc4303>>.
- [RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <<https://www.rfc-editor.org/info/rfc4443>>.

- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <<https://www.rfc-editor.org/info/rfc8200>>.
- [RFC8201] McCann, J., Deering, S., Mogul, J., and R. Hinden, Ed., "Path MTU Discovery for IP version 6", STD 87, RFC 8201, DOI 10.17487/RFC8201, July 2017, <<https://www.rfc-editor.org/info/rfc8201>>.

17.2. Informative References

- [I-D.bonica-6man-unrecognized-opt]
Bonica, R. and J. Leddy, "The IPv6 Probe Option", draft-bonica-6man-unrecognized-opt-03 (work in progress), August 2018.
- [RFC2473] Conta, A. and S. Deering, "Generic Packet Tunneling in IPv6 Specification", RFC 2473, DOI 10.17487/RFC2473, December 1998, <<https://www.rfc-editor.org/info/rfc2473>>.
- [RFC7690] Byerly, M., Hite, M., and J. Jaeggli, "Close Encounters of the ICMP Type 2 Kind (Near Misses with ICMPv6 Packet Too Big (PTB))", RFC 7690, DOI 10.17487/RFC7690, January 2016, <<https://www.rfc-editor.org/info/rfc7690>>.
- [RFC7872] Gont, F., Linkova, J., Chown, T., and W. Liu, "Observations on the Dropping of Packets with IPv6 Extension Headers in the Real World", RFC 7872, DOI 10.17487/RFC7872, June 2016, <<https://www.rfc-editor.org/info/rfc7872>>.

Authors' Addresses

John Leddy
Unaffiliated

Email: john@leddy.net

Ron Bonica
Juniper Networks
2251 Corporate Park Drive
Herndon, Virginia 20171
USA

Email: rbonica@juniper.net

Igor Lubashev
Akamai Technologies
Cambridge, MA
USA

Email: ilubashe@akamai.com

Internet Area Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 6, 2021

V. Olteanu
D. Niculescu
University Politehnica of Bucharest
November 02, 2020

SOCKS Protocol Version 6
draft-olteanu-intarea-socks-6-11

Abstract

The SOCKS protocol is used primarily to proxy TCP connections to arbitrary destinations via the use of a proxy server. Under the latest version of the protocol (version 5), it takes 2 RTTs (or 3, if authentication is used) before data can flow between the client and the server.

This memo proposes SOCKS version 6, which reduces the number of RTTs used, takes full advantage of TCP Fast Open, and adds support for 0-RTT authentication.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 6, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Revision log	4
2. Requirements language	11
3. Mode of operation	11
4. Requests	12
5. Version Mismatch Replies	14
6. Authentication Replies	14
7. Operation Replies	15
7.1. Handling CONNECT	17
7.2. Handling BIND	17
7.3. Handling UDP ASSOCIATE	17
7.3.1. Proxying UDP servers	21
7.3.2. Proxying multicast traffic	21
7.3.3. Reporting ICMP Errors	21
8. SOCKS Options	22
8.1. Stack options	23
8.1.1. IP TOS options	24
8.1.2. Happy Eyeballs options	24
8.1.3. TTL options	25
8.1.4. No Fragmentation options	26
8.1.5. TFO options	26
8.1.6. Multipath options	27
8.1.7. Listen Backlog options	27
8.1.8. UDP Error options	28
8.1.9. Port Parity options	29
8.2. Authentication Method options	30
8.3. Authentication Data options	32
8.4. Session options	32
8.4.1. Session initiation	33
8.4.2. Further SOCKS Requests	34
8.4.3. Tearing down the session	34
8.5. Idempotence options	35
8.5.1. Requesting a token window	35
8.5.2. Spending a token	36
8.5.3. Shifting windows	38
8.5.4. Out-of-order Window Advertisements	38
9. Username/Password Authentication	38
10. TCP Fast Open on the Client-Proxy Leg	39
11. False Starts	39
12. DNS provided by SOCKS	40
13. Security Considerations	40

13.1. Large requests	40
13.2. Replay attacks	41
13.3. Resource exhaustion	41
14. Privacy Considerations	41
15. IANA Considerations	41
16. Acknowledgments	42
17. References	43
17.1. Normative References	43
17.2. Informative References	43
Authors' Addresses	44

1. Introduction

Versions 4 and 5 [RFC1928] of the SOCKS protocol were developed two decades ago and are in widespread use for circuit level gateways or as circumvention tools, and enjoy wide support and usage from various software, such as web browsers, SSH clients, and proxifiers. However, their design needs an update in order to take advantage of the new features of transport protocols, such as TCP Fast Open [RFC7413], or to better assist newer transport protocols, such as MPTCP [RFC6824].

One of the main issues faced by SOCKS version 5 is that, when taking into account the TCP handshake, method negotiation, authentication, connection request and grant, it may take up to 5 RTTs for a data exchange to take place at the application layer. This is especially costly in networks with a large delay at the access layer, such as 3G, 4G, or satellite.

The desire to reduce the number of RTTs manifests itself in the design of newer security protocols. TLS version 1.3 [RFC8446] defines a zero round trip (0-RTT) handshake mode for connections if the client and server had previously communicated.

TCP Fast Open [RFC7413] is a TCP option that allows TCP to send data in the SYN and receive a response in the first ACK, and aims at obtaining a data response in one RTT. The SOCKS protocol needs to concern itself with at least two TFO deployment scenarios: First, when TFO is available end-to-end (at the client, at the proxy, and at the server); second, when TFO is active between the client and the proxy, but not at the server.

This document describes the SOCKS protocol version 6. The key improvements over SOCKS version 5 are:

- o The client sends as much information upfront as possible, and does not wait for the authentication process to conclude before requesting the creation of a socket.

- o The connection request also mimics the semantics of TCP Fast Open [RFC7413]. As part of the connection request, the client can supply the potential payload for the initial SYN that is sent out to the server.
- o The protocol can be extended via options without breaking backward-compatibility.
- o The protocol can leverage the aforementioned options to support 0-RTT authentication schemes.

1.1. Revision log

Typos and minor clarifications are not listed.

draft-11

- o Changed intended status to Standards Track
- o Renamed Vendor-specific option range to Experimental
- o Stack options:
 - * Fixed some instances where an unsupported option was indistinguishable from a case where the proxy couldn't or wouldn't honor it (offenders: Happy Eyeballs, IP Fragmentation, UDP Error, Port Parity)
 - * MPTCP: changed semantics w.r.t. TCP BIND: the absence of such an option SHOULD no longer lead the proxy to refuse MPTCP
 - * Port Parity: relaxed restrictions in case the client supplies a specific port

draft-10

- o Removed untrusted sessions
- o IP DF
- o UDP relay:
 - * Support ICMPv6 Too Big
 - * Shifted some fields in the error messages
 - * RTP support

draft-09

- o Revamped UDP relay
 - * Support for ICMP errors: host/net unreachable, TTL exceeded
 - * Datagrams can be sent over TCP
 - * Timeout for the receipt of the initial datagram
- o TTL stack option (intended use: traceroute)
- o Added the "Privacy Considerations" section
- o SOCKS-provided DNS: the proxy may provide a valid bind address and port

draft-08

- o Removed Address Resolution options
- o Happy Eyeballs options
- o DNS provided by SOCKS

draft-07

- o All fields are now aligned.
- o Eliminated version minors
- o Lots of changes to options
 - * 2-byte option kinds
 - * Flattened option kinds/types/reply codes; also renamed some options
 - * Socket options
 - + Proxies MUST always answer them (Clients can probe for support)
 - + MPTCP Options: expanded functionality ("please do/don't do MPTCP on my behalf")
 - + MPTCP Scheduler options removed

- + Listen Backlog options: code changed to 0x03
 - * Revamped Idempotence options
 - * Auth data options limited to one per method
 - o Authentication Reply: all authentication-related information is now in the options
 - * Authentication replies no longer have a field indicating the chosen auth. method
 - * Method that must proceed (or whereby authentication succeeded) indicated in options
 - * Username/password authentication: proxy now sends reply in option
 - o Removed requirements w.r.t. caching authentication methods by multihomed clients
 - o UDP: 8-byte association IDs
 - o Sessions
 - * The proxy is now free to terminate ongoing connections along with the session.
 - * The session-terminating request is not part of the session that it terminated.
 - o Address Resolution options
- draft-06
- o Session options
 - o Options now have a 2-byte length field.
 - o Stack options
 - * Stack options can no longer contain duplicate information.
 - * TFO: Better payload size semantics
 - * TOS: Added missing code field.
 - * MPTCP Scheduler options:

- + Removed support for round-robin
- + "Default" renamed to "Lowest latency first"
- * Listen Backlog options: now tied to sessions, instead of an authenticated user
- o Idempotence options
 - * Now used in the context of a session (no longer tied to an authenticated user)
 - * Idempotence options have a different codepoint: 0x05. (Was 0x04.)
 - * Clarified that implementations that support Idempotence Options must support all Idempotence Option Types.
 - * Shifted Idempotence Option Types by 1. (Makes implementation easier.)
- o Shrunk vendor-specific option range to 32 (down from 64).
- o Removed reference to dropping initial data. (It could no longer be done as of -05.)
- o Initial data size capped at 16KB.
- o Application data is never encrypted by SOCKS 6. (It can still be encrypted by the TLS layer under SOCKS.)
- o Messages now carry the total length of the options, rather than the number of options. Limited options length to 16KB.
- o Security Considerations
 - * Updated the section to reflect the smaller maximum message size.
 - * Added a subsection on resource exhaustion.

draft-05

- o Limited the "slow" authentication negotiations to one (and Authentication Replies to 2)
- o Revamped the handling of the first bytes in the application data stream

- * False starts are now recommended. (Added the "False Start" section.)
 - * Initial data is only available to clients willing to do "slow" authentication. Moved the "Initial data size" field from Requests to Authentication Method options.
 - * Initial data size capped at 2^{13} . Initial data can no longer be dropped by the proxy.
 - * The TFO option can hint at the desired SYN payload size.
 - o Request: clarified the meaning of the Address and Port fields.
 - o Better reverse TCP proxy support: optional listen backlog for TCP BIND
 - o TFO options can no longer be placed inside Operation Replies.
 - o IP TOS stack option
 - o Suggested a range for vendor-specific options.
 - o Revamped UDP functionality
 - * Now using fixed UDP ports
 - * DTLS support
 - o Stack options: renamed Proxy-Server leg to Proxy-Remote leg
- draft-04
- o Moved Token Expenditure Replies to the Authentication Reply.
 - o Shifted the Initial Data Size field in the Request, in order to make it easier to parse.

draft-03

- o Shifted some fields in the Operation Reply to make it easier to parse.
- o Added connection attempt timeout response code to Operation Replies.
- o Proxies send an additional Authentication Reply after the authentication phase. (Useful for token window advertisements.)

- o Renamed the section "Connection Requests" to "Requests"
- o Clarified the fact that proxies don't need to support any command in particular.
- o Added the section "TCP Fast Open on the Client-Proxy Leg"
- o Options:
 - * Added constants for option kinds
 - * Salt options removed, along with the relevant section from Security Considerations. (TLS 1.3 Makes AEAD mandatory.)
 - * Limited Authentication Data options to one per method.
 - * Relaxed proxy requirements with regard to handling multiple Authentication Data options. (When the client violates the above bullet point.)
 - * Removed interdependence between Authentication Method and Authentication Data options.
 - * Clients SHOULD omit advertising the "No authentication required" option. (Was MAY.)
 - * Idempotence options:
 - + Token Window Advertisements are now part of successful Authentication Replies (so that the proxy-server RTT has no impact on their timeliness).
 - + Proxies can't advertise token windows of size 0.
 - + Tweaked token expenditure response codes.
 - + Support no longer mandatory on the proxy side.
 - * Revamped Socket options
 - + Renamed Socket options to Stack options.
 - + Banned contradictory socket options.
 - + Added socket level for generic IP. Removed the "socket" socket level.
 - + Stack options no longer use option codes from `setsockopt()`.

- + Changed MPTCP Scheduler constants.

draft-02

- o Made support for Idempotence options mandatory for proxies.
- o Clarified what happens when proxies can not or will not issue tokens.
- o Limited token windows to $2^{31} - 1$.
- o Fixed definition of "less than" for tokens.
- o NOOP commands now trigger Operation Replies.
- o Renamed Authentication options to Authentication Data options.
- o Authentication Data options are no longer mandatory.
- o Authentication methods are now advertised via options.
- o Shifted some Request fields.
- o Option range for vendor-specific options.
- o Socket options.
- o Password authentication.
- o Salt options.

draft-01

- o Added this section.
- o Support for idempotent commands.
- o Removed version numbers from operation replies.
- o Request port number for SOCKS over TLS. Deprecate encryption/encapsulation within SOCKS.
- o Added Version Mismatch Replies.
- o Renamed the AUTH command to NOOP.
- o Shifted some fields to make requests and operation replies easier to parse.

2. Requirements language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Mode of operation

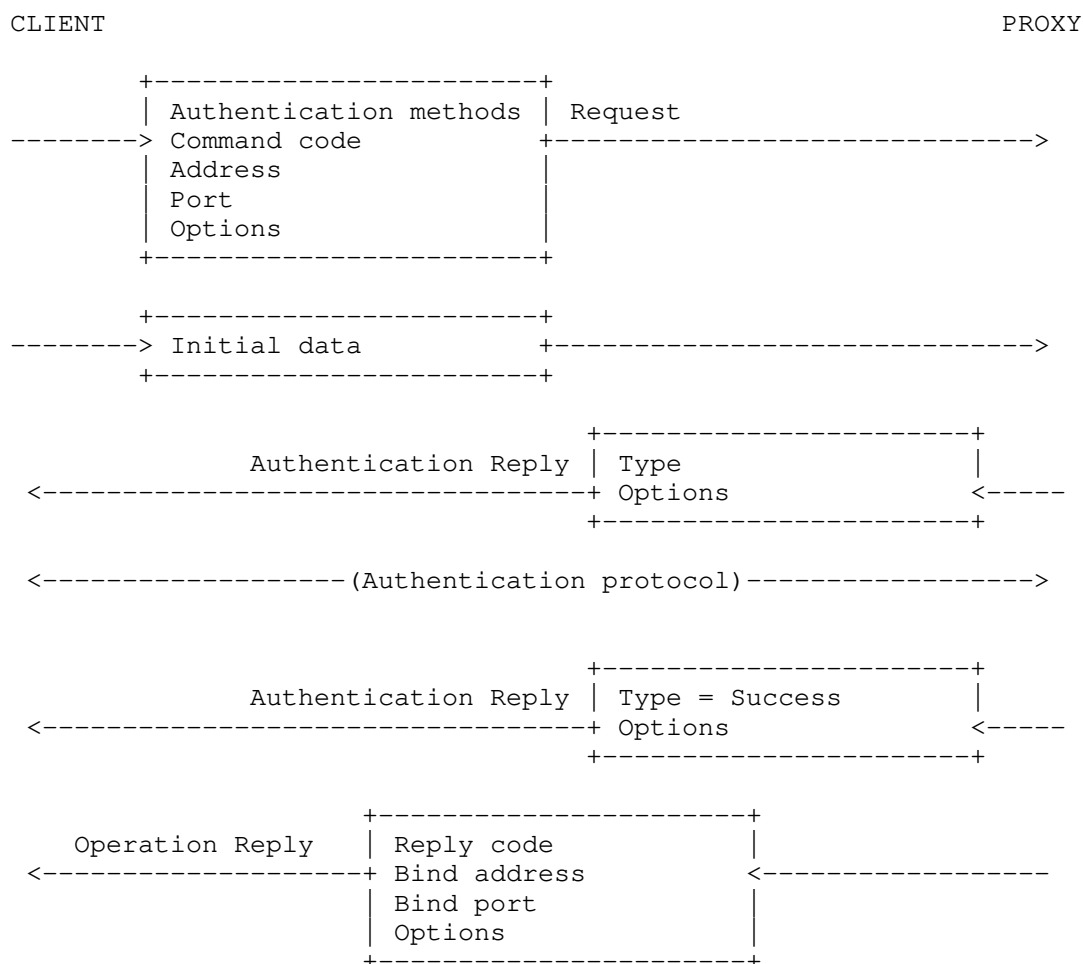


Figure 1: The SOCKS version 6 protocol message exchange

When a TCP-based client wishes to establish a connection to a server, it must open a TCP connection to the appropriate SOCKS port on the

SOCKS proxy. The client then enters a negotiation phase, by sending the request in Figure 1, that contains, in addition to fields present in SOCKS 5 [RFC1928], fields that facilitate low RTT usage and faster authentication negotiation.

Next, the server sends an authentication reply. If the request did not contain the necessary authentication information, the proxy indicates an authentication method that must proceed. This may trigger a longer authentication sequence that could include tokens for ulterior faster authentications. The part labeled "Authentication protocol" is specific to the authentication method employed and is not expected to be employed for every connection between a client and its proxy server. The authentication protocol typically takes up 1 RTT or more.

If the authentication is successful, an operation reply is generated by the proxy. It indicates whether the proxy was successful in creating the requested socket or not.

In the fast case, when authentication is properly set up, the proxy attempts to create the socket immediately after the receipt of the request, thus achieving an operational connection in one RTT (provided TFO functionality is available at the client, proxy, and server).

4. Requests

The client starts by sending a request to the proxy.

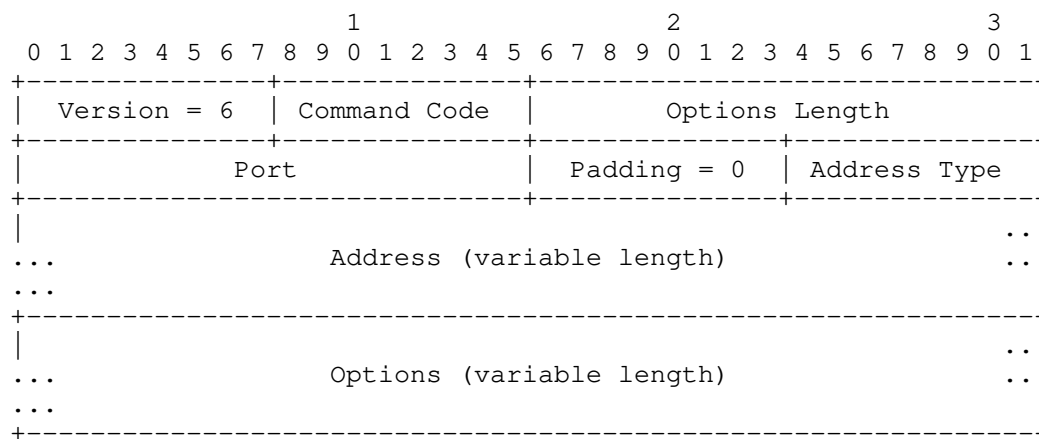


Figure 2: SOCKS 6 Request

- o Version: 6
- o Command Code:
 - * 0x00 NOOP: does nothing.
 - * 0x01 CONNECT: requests the establishment of a TCP connection. TFO MUST NOT be used unless explicitly requested.
 - * 0x02 BIND: requests the establishment of a TCP port binding.
 - * 0x03 UDP ASSOCIATE: requests a UDP port association.
- o Address Type:
 - * 0x01: IPv4
 - * 0x03: Domain Name
 - * 0x04: IPv6
- o Address: this field's format depends on the address type:
 - * IPv4: a 4-byte IPv4 address
 - * Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated, but padded by NUL characters, if needed.
 - * IPv6: a 16-byte IPv6 address
- o Port: the port in network byte order.
- o Padding: set to 0
- o Options Length: the total size of the SOCKS options that appear in the Options field. MUST NOT exceed 16KB.
- o Options: see Section 8.

The Address and Port fields have different meanings based on the Command Code:

- o NOOP: The fields have no meaning. The Address Type field MUST be either 0x01 (IPv4) or 0x04 (IPv6). The Address and Port fields MUST be 0.

- o CONNECT: The fields signify the address and port to which the client wishes to connect.
- o BIND, UDP ASSOCIATE: The fields indicate the desired bind address and port. If the client does not require a certain address, it can set the Address Type field to 0x01 (IPv4) or 0x04 (IPv6), and the Address field to 0. Likewise, if the client does not require a certain port, it can set the Port field to 0.

Clients can advertise their supported authentication methods by including an Authentication Method Advertisement option (see Section 8.2).

5. Version Mismatch Replies

Upon receipt of a request starting with a version number other than 6, the proxy sends the following response:

```

 0 1 2 3 4 5 6 7
+-----+
| Version = 6 |
+-----+
```

Figure 3: SOCKS 6 Version Mismatch Reply

- o Version: 6

A client MUST close the connection after receiving such a reply.

6. Authentication Replies

Upon receipt of a valid request, the proxy sends an Authentication Reply:

```

                                1                2                3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+
| Version = 6 | Type | Options Length |
+-----+-----+-----+
|                                     ...
...                               Options (variable length)         ...
...                                     ...
+-----+-----+-----+
```

Figure 4: SOCKS 6 Authentication Reply

- o Version: 6
- o Type:
 - * 0x00: authentication successful.
 - * 0x01: authentication failed.
- o Options Length: the total size of the SOCKS options that appear in the Options field. MUST NOT exceed 16KB.
- o Options: see Section 8.

If the server signals that the authentication has failed and does not signal that any authentication negotiation can continue (via an Authentication Method Selection option), the client MUST close the connection.

The client and proxy begin a method-specific negotiation. During such negotiations, the proxy MAY supply information that allows the client to authenticate a future request using an Authentication Data option. Application data is not subject to any encryption negotiated during this phase. Descriptions of such negotiations are beyond the scope of this memo.

When the negotiation is complete (either successfully or unsuccessfully), the proxy sends a second Authentication Reply. The second Authentication Reply MUST NOT allow for further negotiations.

7. Operation Replies

After the authentication negotiations are complete, the proxy sends an Operation Reply:

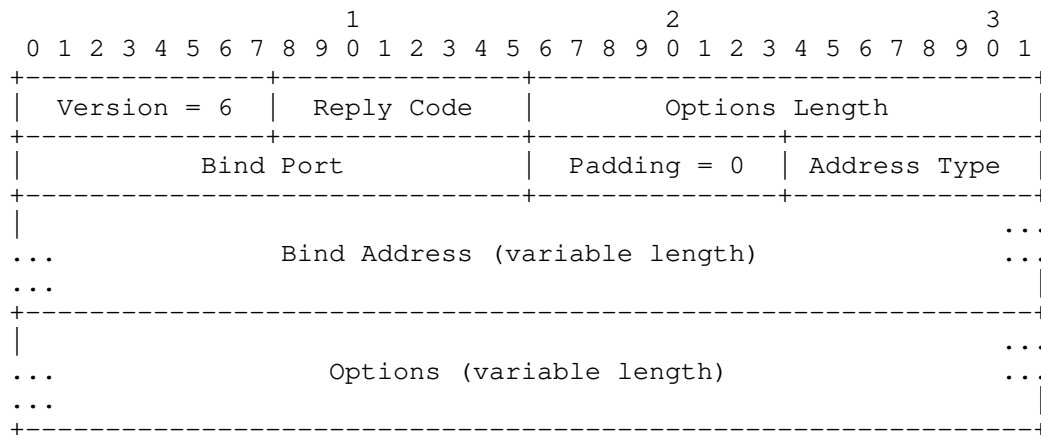


Figure 5: SOCKS 6 Operation Reply

- o Version: 6
- o Reply Code:
 - * 0x00: Success
 - * 0x01: General SOCKS server failure
 - * 0x02: Connection not allowed by ruleset
 - * 0x03: Network unreachable
 - * 0x04: Host unreachable
 - * 0x05: Connection refused
 - * 0x06: TTL expired
 - * 0x07: Command not supported
 - * 0x08: Address type not supported
 - * 0x09: Connection attempt timed out
- o Bind Port: the proxy bound port in network byte order.
- o Padding: set to 0
- o Address Type:

- * 0x01: IPv4
- * 0x03: Domain Name
- * 0x04: IPv6
- o Bind Address: the proxy bound address in the following format:
 - * IPv4: a 4-byte IPv4 address
 - * Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated, but padded by NUL characters, if needed.
 - * IPv6: a 16-byte IPv6 address
- o Options Length: the total size of the SOCKS options that appear in the Options field. MUST NOT exceed 16KB.
- o Options: see Section 8.

Proxy implementations MAY support any subset of the client commands listed in Section 4.

If the proxy returns a reply code other than "Success", the client MUST close the connection.

If the client issued an NOOP command, the client MUST close the connection after receiving the Operation Reply.

7.1. Handling CONNECT

In case the client has issued a CONNECT request, data can now pass.

7.2. Handling BIND

In case the client has issued a BIND request, it must wait for a second Operation reply from the proxy, which signifies that a host has connected to the bound port. The Bind Address and Bind Port fields contain the address and port of the connecting host. Afterwards, application data may pass.

7.3. Handling UDP ASSOCIATE

Proxies offering UDP functionality may be configured with a UDP port used for relaying UDP datagrams to and from the client, and/or a port used for relaying datagrams over DTLS.

Following a successful Operation Reply, the client and the proxy begin exchanging messages with the following header:

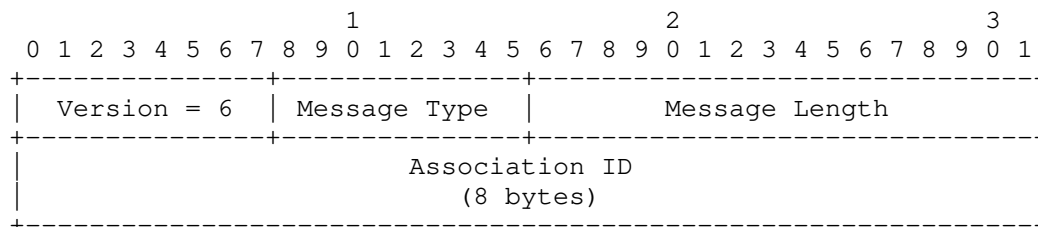


Figure 6: UDP Association Header

o Message Type:

- * 0x01: Association Initialization
- * 0x02: Association Confirmation
- * 0x03: Datagram
- * 0x04: Error

o Message Length: the total length of the message

o Association ID: the identifier of the UDP association

First, the proxy picks an Association ID sends a an Association Initialization message:

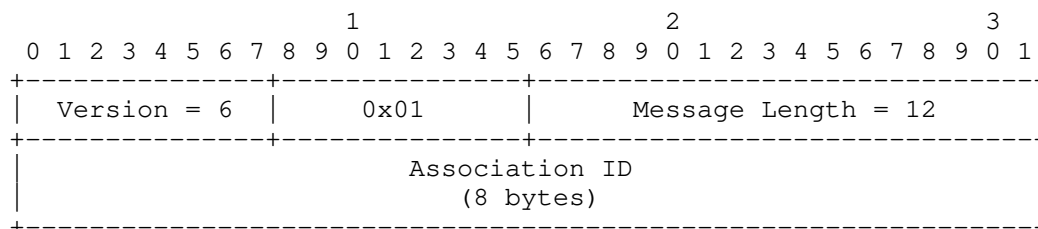


Figure 7: UDP Association Initialization

Proxy implementations SHOULD generate Association IDs randomly or pseudo-randomly.

Clients may start sending datagrams to the proxy either:

- o over the TCP connection,
- o in plaintext, using the proxy's configured UDP port(s), or
- o over an established DTLS session.

A client's datagrams are prefixed by a Datagram Header, indicating the remote host's address and port:

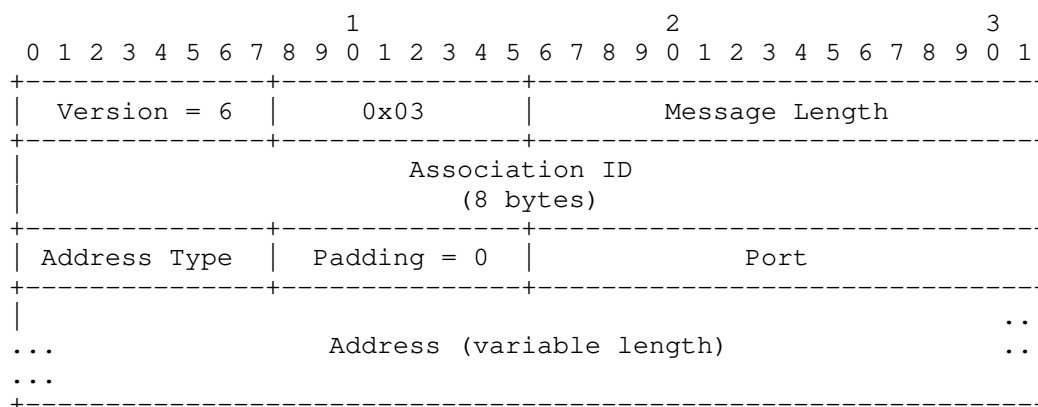


Figure 8: Datagram Header

- o Version: 0x06
- o Association ID: the identifier of the UDP association
- o Address Type:
 - * 0x01: IPv4
 - * 0x03: Domain Name
 - * 0x04: IPv6
- o Address: this field's format depends on the address type:
 - * IPv4: a 4-byte IPv4 address
 - * Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated.
 - * IPv6: a 16-byte IPv6 address

- o Port: the port in network byte order.

Datagrams sent over UDP MAY be padded with arbitrary data (i. e., the Message Length MAY be smaller than the actual UDP/DTLS payload). Client and proxy implementations MUST ignore the padding. If the Message Length is larger than the size of the UDP or DTLS payload, the message MUST be silently ignored.

Following the receipt of the first datagram from the client, the proxy makes a one-way mapping between the Association ID and:

- o the TCP connection, if it was received over TCP, or
- o the 5-tuple of the UDP conversation, if the datagram was received over plain UDP, or
- o the DTLS connection, if the datagram was received over DTLS. The DTLS connection is identified either by its 5-tuple, or some other mechanism, like [I-D.ietf-tls-dtls-connection-id].

The proxy SHOULD close the TCP connection if the initial datagram is not received after a timeout.

Further datagrams carrying the same Association ID, but not matching the established mapping, are silently dropped.

The proxy then sends an UDP Association Confirmation message over the TCP connection with the client:

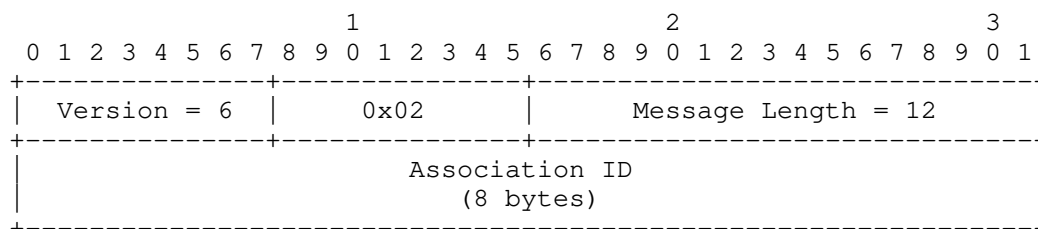


Figure 9: UDP Association Confirmation

Following the confirmation message, UDP packets bound for the proxy's bind address and port are relayed to the client, also prefixed by a Datagram Header.

The UDP association remains active for as long as the TCP connection between the client and the proxy is kept open.

7.3.1. Proxying UDP servers

Under some circumstances (e.g. when hosting a server), the SOCKS client expects the remote host to send UDP datagrams first. As such, the SOCKS client must trigger a UDP Association Confirmation without having the proxy relay any datagrams on its behalf.

To that end, it sends an empty datagram prefixed by a Datagram Header with an IP address and port consisting of zeroes. If it is using UDP, the client **SHOULD** resend the empty datagram if an UDP Association Confirmation is not received after a timeout.

7.3.2. Proxying multicast traffic

The use of multicast addresses is permitted for UDP traffic only.

7.3.3. Reporting ICMP Errors

If a client has opted in (see Section 8.1.8), the proxy **MAY** relay information contained in some ICMP Error packets. The message format is as follows:

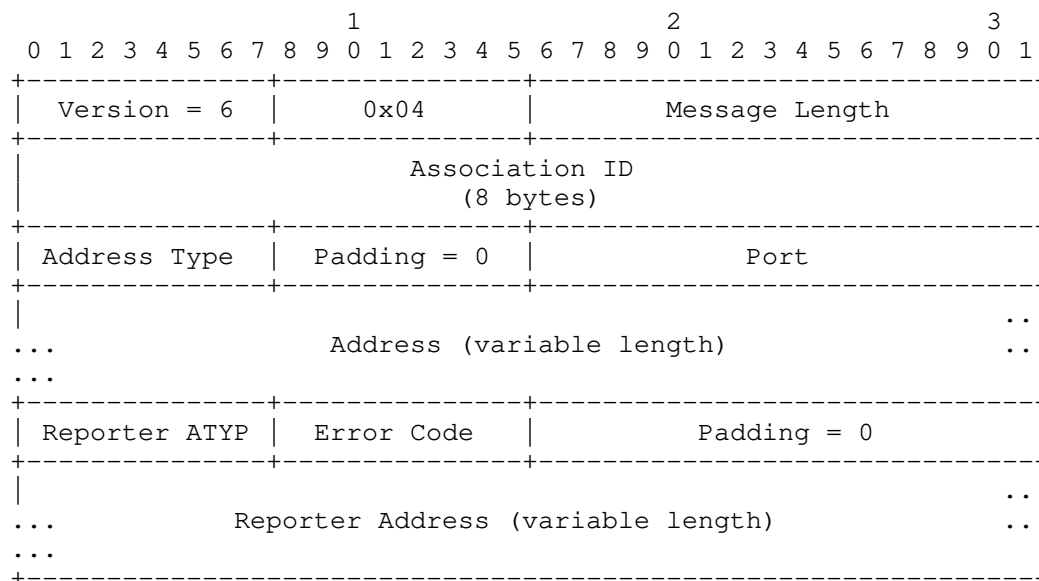


Figure 10: Datagram Error Message

- o Address: The destination address of the IP header contained in the ICMP payload

- o Address Type: Either 0x01 (IPv4) or 0x04 (IPv6)
- o Port: The destination port of the UDP header contained in the ICMP payload
- o Reporter Address: The IP address of the host that issued the ICMP error
- o Reporter Address Type (ATYP): Either 0x01 (IPv4) or 0x04 (IPv6)
- o Error code:
 - * 0x01: Network unreachable
 - * 0x02: Host unreachable
 - * 0x03: TTL expired
 - * 0x04: Datagram too big (IPv6 only)

It is possible for ICMP Error packets to be spurious, and not be related to any UDP packet that was sent out. The proxy is not required to check the validity of ICMP Error packets before reporting them to the client.

Clients **MUST NOT** send Datagram Error messages to the proxy. Proxies **MUST NOT** send Error messages unless the clients have opted in.

8. SOCKS Options

SOCKS options have the following format:

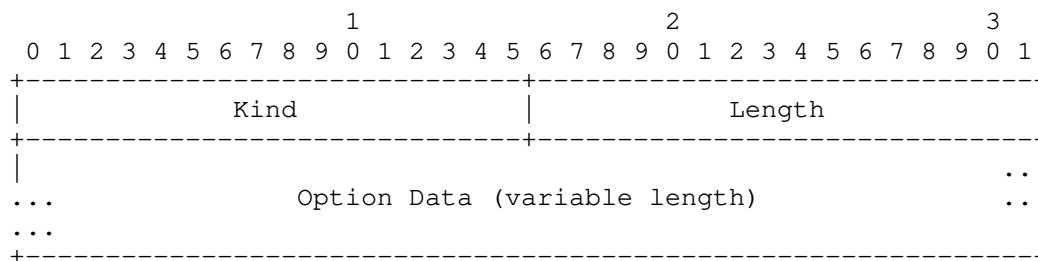


Figure 11: SOCKS 6 Option

- o Kind: Allocated by IANA. (See Section 15.)
- o Length: The total length of the option. **MUST** be a multiple of 4.

- o Option Data: The contents are specific to each option kind.

Unless otherwise noted, client and proxy implementations MAY omit supporting any of the options described in this document. Upon encountering an unsupported option, a SOCKS endpoint MUST silently ignore it.

8.1. Stack options

Stack options can be used by clients to alter the behavior of the protocols on top of which SOCKS is running, as well the protocols used by the proxy to communicate with the remote host (i.e. IP, TCP, UDP). A Stack option can affect either the proxy's protocol on the client-proxy leg or on the proxy-remote leg. Clients can only place Stack options inside SOCKS Requests.

Proxies MAY choose not to honor any Stack options sent by the client.

Proxies include Stack options in their Operation Replies to signal their behavior, and MUST do so for every supported Stack option sent by the client. Said options MAY also be unsolicited, i. e. the proxy MAY send them to signal behavior that was not explicitly requested by the client.

If a particular Stack option is unsupported, the proxy MUST silently ignore it.

In case of UDP ASSOCIATE, the stack options refer to the UDP traffic relayed by the proxy.

Stack options that are part of the same message MUST NOT contradict one another or contain duplicate information.

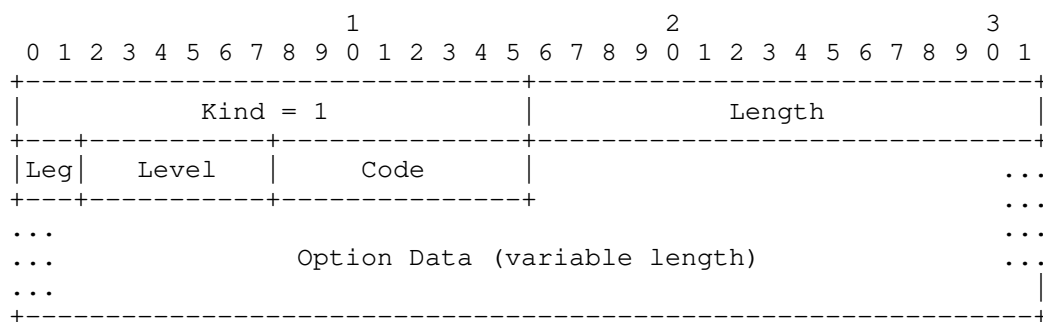


Figure 12: Stack Option

- o Leg:
 - * 1: Client-Proxy Leg
 - * 2: Proxy-Remote Leg
 - * 3: Both Legs
- o Level:
 - * 1: IP: options that apply to either IPv4 or IPv6
 - * 2: IPv4
 - * 3: IPv6
 - * 4: TCP
 - * 5: UDP
- o Code: Option code
- o Option Data: Option-specific data

8.1.1. IP TOS options

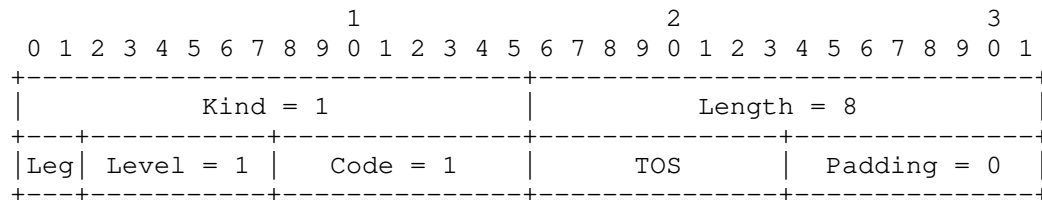


Figure 13: IP TOS Option

- o TOS: The IP TOS code

The client can use IP TOS options to request that the proxy use a certain value for the IP TOS field. Likewise, the proxy can use IP TOS options to advertise the TOS values being used.

8.1.2. Happy Eyeballs options

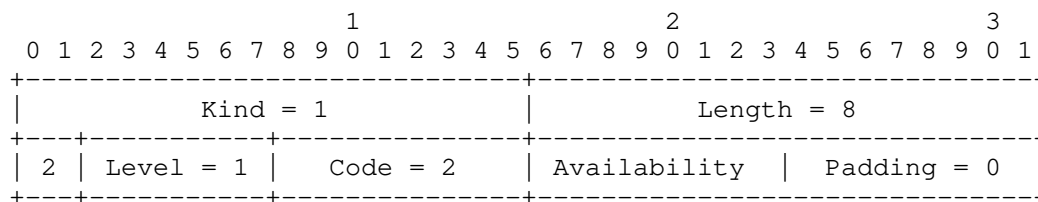


Figure 14: Happy Eyeballs Option

o Availability:

- * 0x01: Happy Eyeballs is not desired (client) or was not performed (proxy)
- * 0x02: Happy Eyeballs is desired (client) or was attempted (proxy)

This memo provides enough features for clients to implement a mechanism analogous to Happy Eyeballs [RFC8305] over SOCKS. However, when the delay between the client and the proxy, or the proxy's vantage point, is high, doing so can become impractical or inefficient.

In such cases, the client can instruct the proxy to employ the Happy Eyeballs technique on its behalf when connecting to a remote host.

The client **MUST** supply a Domain Name as part of its Request. Otherwise, the proxy **MUST** silently ignore the option.

TODO: Figure out which knobs to include.

8.1.3. TTL options

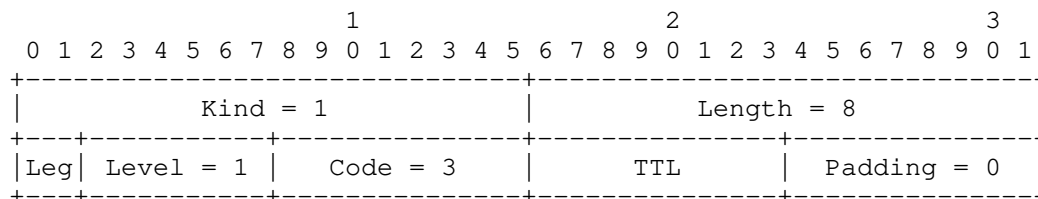


Figure 15: IP TTL Option

o TTL: The IP TTL or Hop Limit

8.1.4. No Fragmentation options

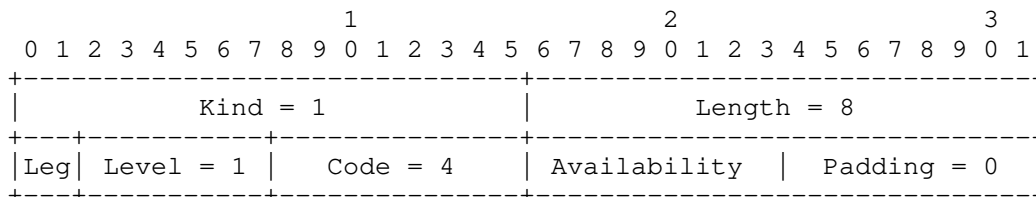


Figure 16: No Fragmentation Option

o Availability:

- * 0x01: IP fragmentation is allowed (client) or the lack thereof is not enforced (proxy)
- * 0x02: IP fragmentation is not desired (client) or avoidance of fragmentation is enforced (proxy)

A No Fragmentation option can be used to instruct the proxy to avoid IP fragmentation. In the case of IPv4, this also entails setting the DF bit on outgoing packets.

8.1.5. TFO options

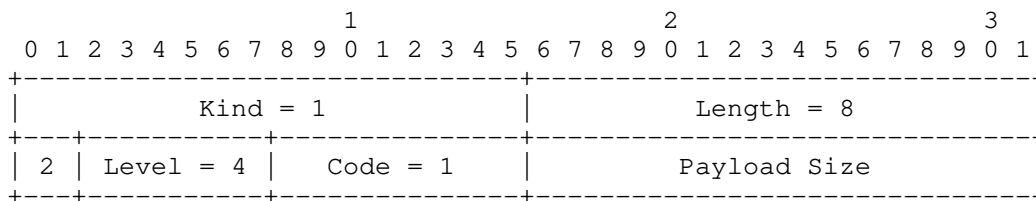


Figure 17: TFO Option

- o Payload Size: The desired payload size of the TFO SYN. Ignored in case of a BIND command.

If a SOCKS Request contains a TFO option, the proxy SHOULD attempt to use TFO in case of a CONNECT command, or accept TFO in case of a BIND command. Otherwise, the proxy MUST NOT attempt to use TFO in case of a CONNECT command, or accept TFO in case of a BIND command.

In case of a CONNECT command, the client can indicate the desired payload size of the SYN. If the field is 0, the proxy can use an

arbitrary payload size. If the field is non-zero, the proxy MUST NOT use a payload size larger than the one indicated. The proxy MAY use a smaller payload size than the one indicated.

8.1.6. Multipath options

In case of a CONNECT or BIND command, the client can inform the proxy whether MPTCP is desired on the proxy-remote leg by sending a Multipath option.

Conversely, the proxy can use a Multipath option to convey the following information:

- o whether or not the connection uses MPTCP or not, when replying to a CONNECT command, or in the second Operation reply to a BIND command, or
- o whether an MPTCP connection will be accepted, when first replying to a BIND command.

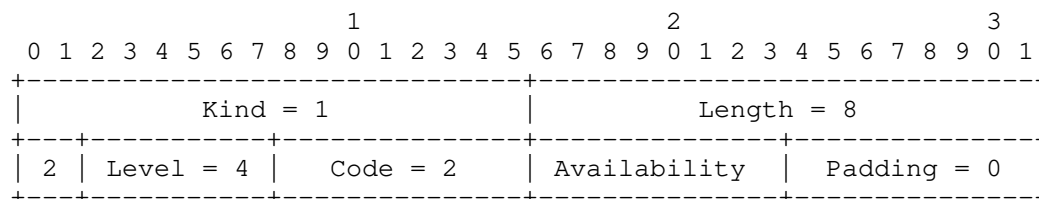


Figure 18: Multipath Option

- o Availability:
 - * 0x01: MPTCP is not desired (client) or available (proxy)
 - * 0x02: MPTCP is desired (client) or available (proxy)

In the absence of such an option, the proxy SHOULD NOT enable MPTCP for CONNECT commands.

8.1.7. Listen Backlog options

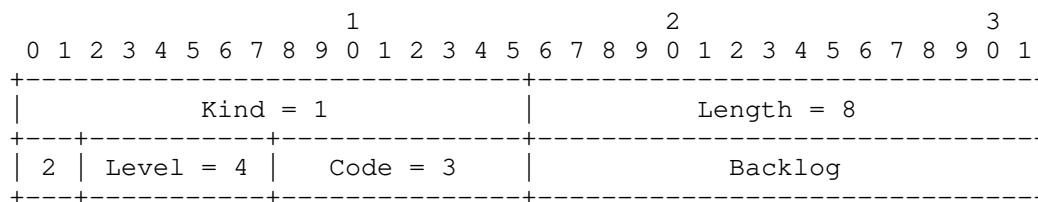


Figure 19: Listen Backlog Option

- o Backlog: The length of the listen backlog.

The default behavior of the BIND does not allow a client to simultaneously handle multiple connections to the same bind address. A client can alter BIND's behavior by adding a TCP Listen Backlog Option to a BIND Request, provided that the Request is part of a Session.

In response, the proxy sends a TCP Listen Backlog Option as part of the Operation Reply, with the Backlog field signaling the actual backlog used. The proxy SHOULD NOT use a backlog longer than requested.

Following the successful negotiation of a backlog, the proxy listens for incoming connections for as long as the initial connection stays open. The initial connection is not used to relay data between the client and a remote host.

To accept connections, the client issues further BIND Requests using the bind address and port supplied by the proxy in the initial Operation Reply. Said BIND requests must belong to the same Session as the original Request.

If no backlog is issued, the proxy signals a backlog length of 0, and BIND's behavior remains unaffected.

8.1.8. UDP Error options

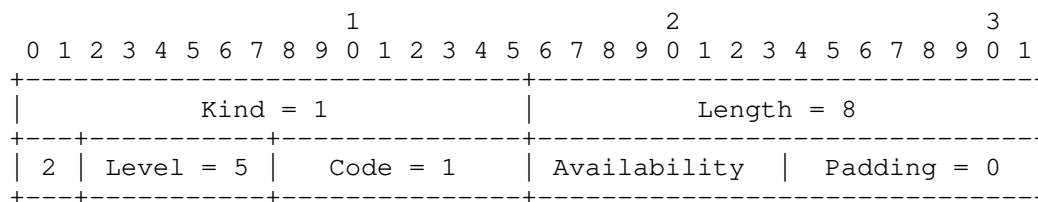


Figure 20: UDP Error Option

o Availability:

- * 0x01: Error reporting is not desired (client) or will not be performed (proxy)
- * 0x02: Error reporting is desired (client) or will be performed (proxy)

Clients can use this option to turn on error reporting for a particular UDP association. See Section 7.3.3.

8.1.9. Port Parity options

The RTP specification [RFC3550] recommends running the protocol on consecutive UDP ports, where the even port is the lower of the two.

SOCKS clients can specify the desired port parity when issuing a UDP ASSOCIATE command, and request that the port's counterpart be reserved.

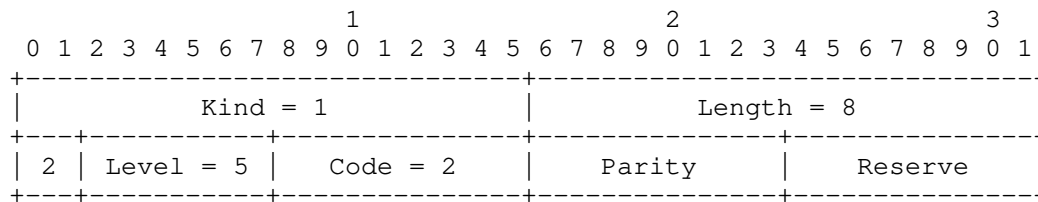


Figure 21: Port Parity Option

o Parity:

- * 0x00: No particular parity
- * 0x01: Even

- * 0x02: Odd
- o Reserve: whether or not to reserve the port's counterpart
- * 0x00: Don't reserve
- * 0x01: Reserve

If the UDP ASSOCIATE request does not have the Port field set to 0 (indicating that an arbitrary port can be chosen), the proxy MUST ignore the suggested parity.

A port's counterpart is determined as follows:

- o for even ports, it is the next higher port and
- o for odd ports, it is the next lower port.

If the proxy can not or will not comply with the requested parity, it also does not reserve the allocated port's counterpart.

Port reservations are in place until either:

- o the original association ends, or
- o an association involving the reserved port is made.

An association involving a reserved port can only be made if a client explicitly requests said port. Further, if the original association is part of a session (see Section 8.4), the reserved port can only be claimed from within the same session.

8.2. Authentication Method options

A client that is willing to go through the authentication phase MUST include an Authentication Method Advertisement option in its Request. In case of a CONNECT Request, the option is also used to specify the amount of initial data supplied before any method-specific authentication negotiations take place.

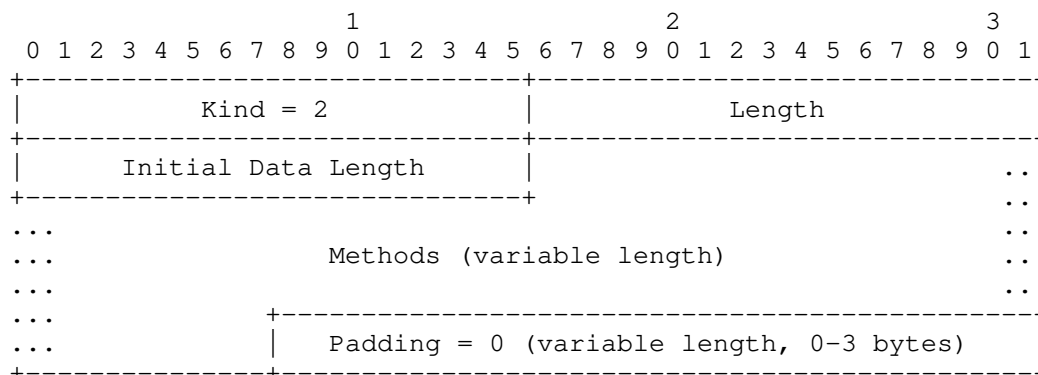


Figure 22: Authentication Method Advertisement Option

- o Initial Data Size: A two-byte number in network byte order. In case of CONNECT, this is the number of bytes of initial data that are supplied by the client immediately following the Request. This number MUST NOT be larger than 2^{14} .
- o Methods: One byte per advertised method. Method numbers are assigned by IANA.
- o Padding: A minimally-sized sequence of zeroes, such that the option length is a multiple of 4. Note that 0 coincides with the value for "No Authentication Required".

Clients MUST support the "No authentication required" method. Clients SHOULD omit advertising the "No authentication required" option.

The proxy indicates which authentication method must proceed by sending an Authentication Method Selection option in the corresponding Authentication Reply:

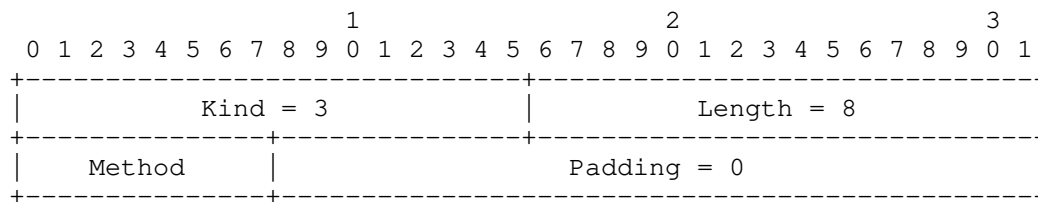


Figure 23: Authentication Method Selection Option

- o Method: The selected method.

If the proxy selects "No Acceptable Methods", the client MUST close the connection.

If authentication is successful via some other means, or not required at all, the proxy silently ignores the Authentication Method Advertisement option.

8.3. Authentication Data options

Authentication Data options carry method-specific authentication data. They can be part of SOCKS Requests and Authentication Replies.

Authentication Data options have the following format:

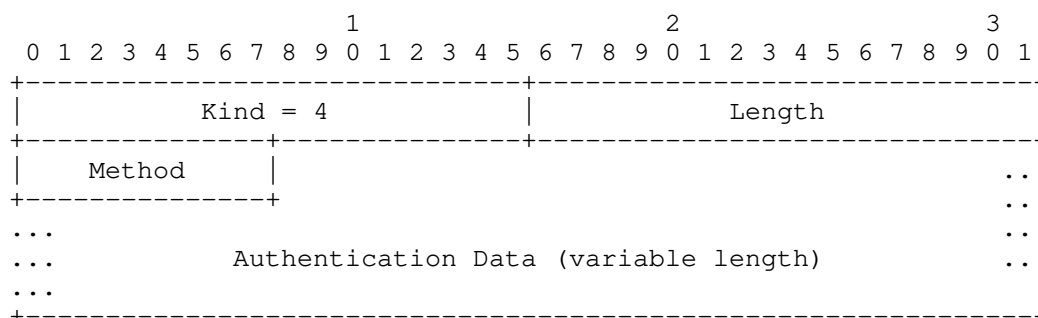


Figure 24: Authentication Data Option

- o Method: The number of the authentication method. These numbers are assigned by IANA.
- o Authentication Data: The contents are specific to each method.

Clients MUST only place one Authentication Data option per authentication method.

8.4. Session options

Clients and proxies can establish SOCKS sessions, which span one or more Requests. All session-related negotiations are done via Session Options, which are placed in Requests and Authentication Replies by the client and, respectively, by the proxy.

Client and proxy implementations MUST either support all Session Option Types, or none.

8.4.1. Session initiation

A client can initiate a session by sending a Session Request Option:

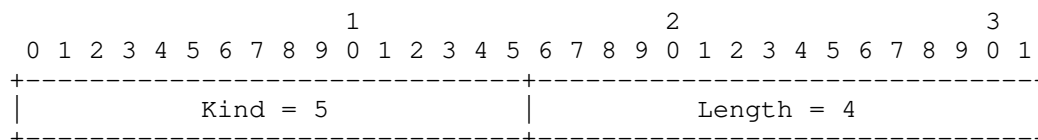


Figure 25: Session Request Option

The proxy then replies with a Session ID Option in the successful Operation Reply:

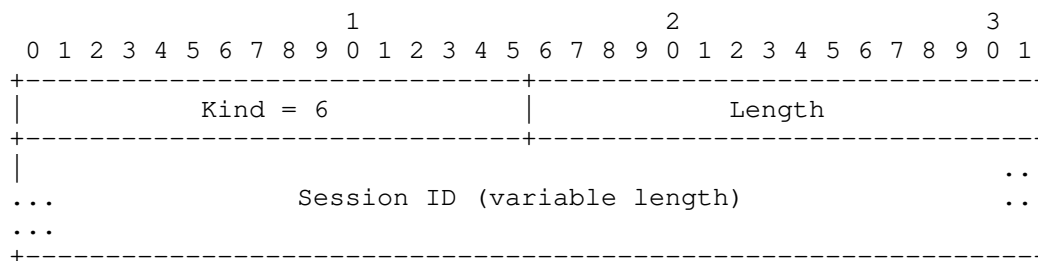


Figure 26: Session ID Option

- o Session ID: An opaque sequence of bytes specific to the session. The size MUST be a multiple of 4. MUST NOT be empty.

The Session ID serves to identify the session and is opaque to the client.

The credentials, or lack thereof, used to initiate the session are tied to the session.

The SOCKS Request that initiated the session is considered part of the session. A client MUST NOT attempt to initiate a session from within a different session.

If the proxy can not or will not honor the Session Request, it does so silently.

8.4.2. Further SOCKS Requests

Any further SOCKS Requests that are part of the session MUST include a Session ID Option (as seen in Figure 26). The proxy MUST silently ignore any authentication attempt in the Request, and MUST NOT require any authentication.

The proxy then replies by placing a Session OK option in the successful Authentication Reply:

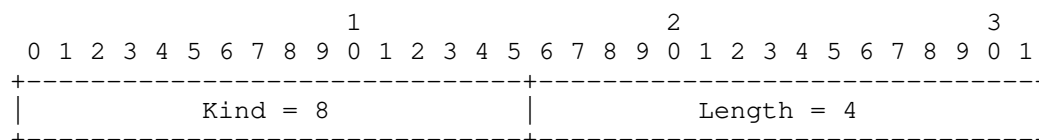


Figure 27: Session OK Option

If the Session ID is invalid, the first Authentication Reply MUST signal that authentication failed and can not continue (by setting the Type field to 0x01). Further, it SHALL contain a Session Invalid option:

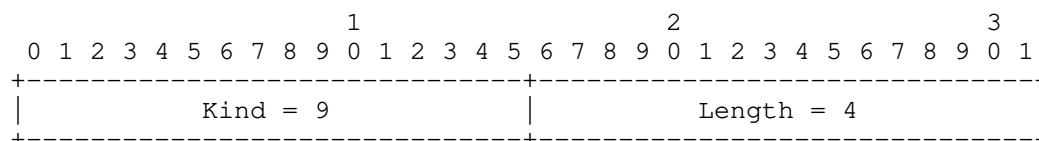


Figure 28: Session Invalid Option

8.4.3. Tearing down the session

Proxies can, at their discretion, tear down a session and free all associated state. Proxy implementations SHOULD feature a timeout mechanism that destroys sessions after a period of inactivity. When a session is terminated, the proxy MAY close all connections associated with said session.

Clients can signal that a session is no longer needed, and can be torn down, by sending a Session Teardown option in addition to the Session ID option:

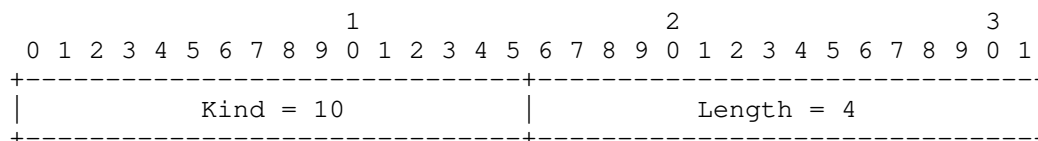


Figure 29: Session Teardown Option

After sending such an option, the client **MUST** assume that the session is no longer valid. The proxy **MUST** treat the session-terminating request as if it were not part of any session.

8.5. Idempotence options

To protect against duplicate SOCKS Requests, clients can request, and then spend, idempotence tokens. A token can only be spent on a single SOCKS request.

Tokens are 4-byte unsigned integers in a modular 4-byte space. Therefore, if x and y are tokens, x is less than y if $0 < (y - x) < 2^{31}$ in unsigned 32-bit arithmetic.

Proxies grant contiguous ranges of tokens called token windows. Token windows are defined by their base (the first token in the range) and size.

All token-related operations are done via Idempotence options.

Idempotence options are only valid in the context of a SOCKS Session. If a SOCKS Request is not part of a Session (either by supplying a valid Session ID or successfully initiating one via a Session Request), the proxy **MUST** silently ignore any Idempotence options.

Token windows are tracked by the proxy on a per-session basis. There can be at most one token window for every session and its tokens can only be spent from within said session.

Client and proxy implementations **MUST** either support all Idempotence Option Types, or none.

8.5.1. Requesting a token window

A client can obtain a window of tokens by sending an Idempotence Request option as part of a SOCKS Request:

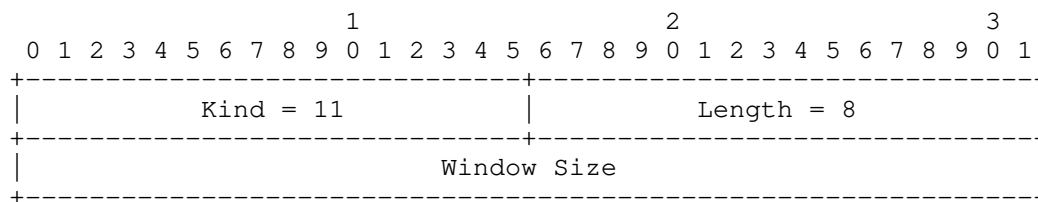


Figure 30: Token Request

- o Window Size: The requested window size.

Once a token window is issued, the proxy MUST include an Idempotence Window option in all subsequent successful Authentication Replies:

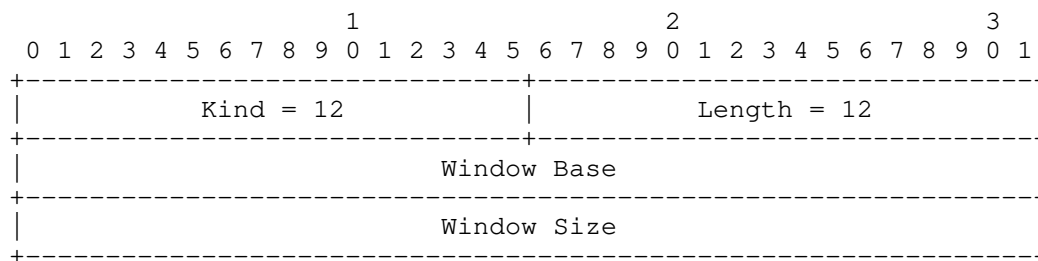


Figure 31: Idempotence Window

- o Window Base: The first token in the window.
- o Window Size: The window size. This value MAY differ from the requested window size. Window sizes MUST be less than 2^{31} . Window sizes MUST NOT be 0.

If no token window is issued, the proxy MUST silently ignore the Token Request. If there is already a token window associated with the session, the proxy MUST NOT issue a new window.

8.5.2. Spending a token

The client can attempt to spend a token by including a Idempotence Expenditure option in its SOCKS request:

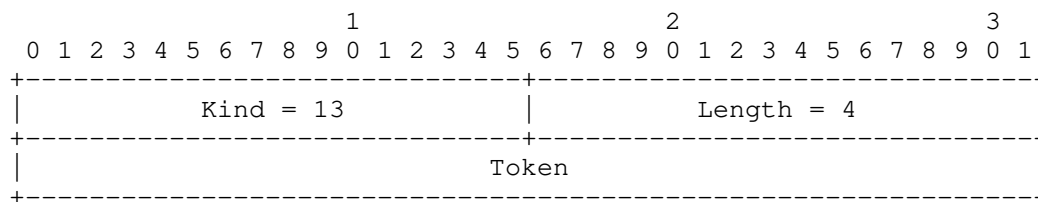


Figure 32: Idempotence Expenditure

- o Kind: 13 (Idempotence Expenditure option)
- o Length: 8
- o Token: The token being spent.

Clients SHOULD prioritize spending the smaller tokens.

The proxy responds by sending either an Idempotence Accepted or Rejected option as part of the Authentication Reply:

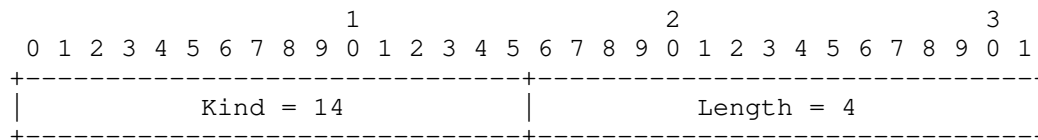


Figure 33: Idempotence Accepted

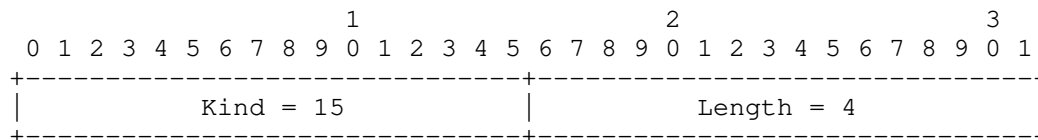


Figure 34: Idempotence Rejected

If eligible, the token is spent before attempting to honor the Request. If the token is not eligible for spending, the Authentication Reply MUST indicate failure.

8.5.3. Shifting windows

Windows can be shifted (i. e. have their base increased, while retaining their size) unilaterally by the proxy.

Proxy implementations SHOULD shift the window: * as soon as the lowest-order token in the window is spent and * when a sufficiently high-order token is spent.

Proxy implementations SHOULD NOT shift the window's base beyond the highest unspent token.

8.5.4. Out-of-order Window Advertisements

Even though the proxy increases the window's base monotonically, there is no mechanism whereby a SOCKS client can receive the Token Window Advertisements in order. As such, clients SHOULD disregard Token Window Advertisements with a Window Base less than the previously known value.

9. Username/Password Authentication

Username/Password authentication is carried out as in [RFC1929].

Clients can also attempt to authenticate by placing the Username/Password request in an Authentication Data Option.

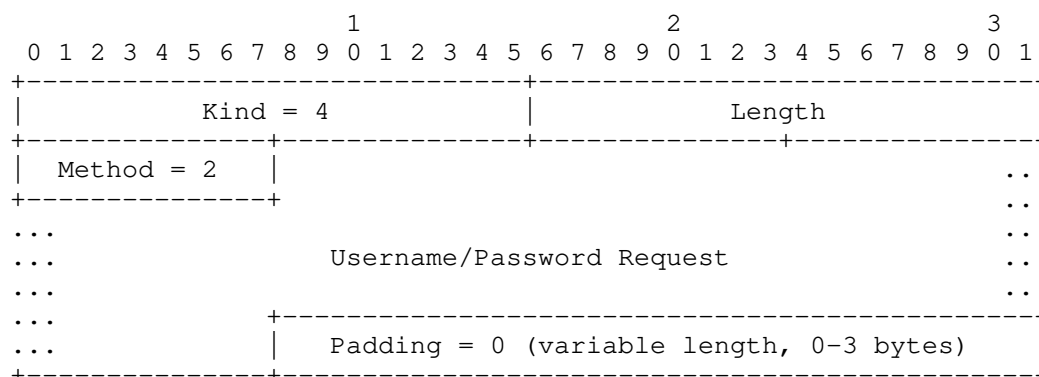


Figure 35: Password authentication via a SOCKS Option

- o Username/Password Request: The Username/Password Request, as described in [RFC1929].

Proxies reply by including a Authentication Data Option in the next Authentication Reply which contains the Username/Password reply:

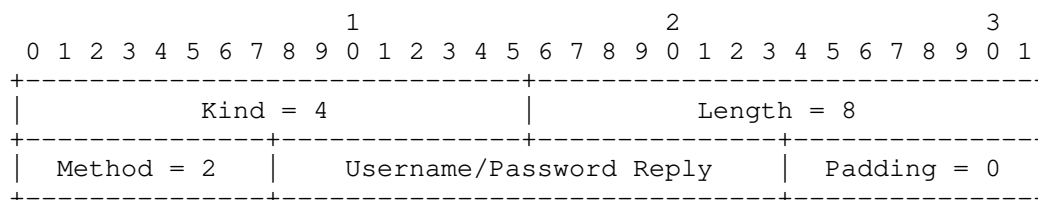


Figure 36: Reply to password authentication via a SOCKS Option

- o Username/Password Reply: The Username/Password Reply, as described in [RFC1929].

10. TCP Fast Open on the Client-Proxy Leg

TFO breaks TCP semantics, causing replays of the data in the SYN's payload under certain rare circumstances [RFC7413]. A replayed SOCKS Request could itself result in a replayed connection on behalf of the client.

As such, client implementations SHOULD NOT use TFO on the client-proxy leg unless:

- o The protocol running on top of SOCKS tolerates the risks of TFO, or
- o The SYN's payload does not contain any application data (so that no data is replayed to the server, even though duplicate connections are still possible), or
- o The client uses Idempotence Options, making replays very unlikely, or
- o SOCKS is running on top of TLS and Early Data is not used.

11. False Starts

In case of CONNECT Requests, the client MAY start sending application data as soon as possible, as long as doing so does not incur the risk of breaking the SOCKS protocol.

Clients must work around the authentication phase by doing any of the following:

- o If the Request does not contain an Authentication Method Advertisement option, the authentication phase is guaranteed not to happen. In this case, application data MAY be sent immediately after the Request.
- o Application data MAY be sent immediately after receiving an Authentication Reply indicating success.
- o When performing a method-specific authentication sequence, application data MAY be sent immediately after the last client message.

12. DNS provided by SOCKS

Clients may require information typically obtained from DNS servers, albeit from the proxy's vantage point.

While the CONNECT command can work with domain names, some clients' workflows require that addresses be resolved as a separate step prior to connecting. Moreover, the SOCKS Datagram Header, as described in Section 7.3, can be reduced in size by providing the resolved destination IP address, rather than the FQDN.

Emerging techniques may also make use of DNS to deliver server-specific information to clients. For example, Encrypted SNI [I-D.ietf-tls-esni] relies on DNS to publish encryption keys.

Proxy implementations MAY provide a default plaintext DNS service. A client looking to make use of it issues a CONNECT Request to IP address 0.0.0.0 or 0:0:0:0:0:0:0:0 on port 53. Following successful authentication, the Operation Reply MAY indicate an unspecified bind address (0.0.0.0 or ::) and port (0). The client and proxy then behave as per [RFC7766].

The service itself can be provided directly by the proxy daemon, or by proxying the client's request to a pre-configured DNS server.

If the proxy does not implement such functionality, it MAY return an error code signaling "Connection refused".

13. Security Considerations

13.1. Large requests

Given the format of the request message, a malicious client could craft a request that is in excess of 16 KB and proxies could be prone to DDoS attacks.

To mitigate such attacks, proxy implementations SHOULD be able to incrementally parse the requests. Proxies MAY close the connection to the client if:

- o the request is not fully received after a certain timeout, or
- o the number of options or their size exceeds an imposed hard cap.

13.2. Replay attacks

In TLS 1.3, early data (which is likely to contain a full SOCKS request) is prone to replay attacks.

While Token Expenditure options can be used to mitigate replay attacks, anything prior to the initial Token Request is still vulnerable. As such, client implementations SHOULD NOT make use of TLS early data unless the Request attempts to spend a token.

13.3. Resource exhaustion

Malicious clients can issue a large number of Session Requests, forcing the proxy to keep large amounts of state.

To mitigate this, the proxy MAY implement policies restricting the number of concurrent sessions on a per-IP or per-user basis, or barring unauthenticated clients from establishing sessions.

14. Privacy Considerations

The timing of Operation Replies can reveal some information about a proxy's recent usage:

- o The DNS resolver used by the proxy may cache the answer to recent queries. As such, subsequent connection attempts to the same hostname are likely to be slightly faster, even if requested by different clients.
- o Likewise, the proxy's OS typically caches TFO cookies. Repeated TFO connection attempts tend to be sped up, regardless of the client.

15. IANA Considerations

This document requests that IANA allocate 2-byte option kinds for SOCKS 6 options. Further, this document requests the following option kinds:

- o Unassigned: 0

- o Stack: 1
- o Authentication Method Advertisement: 2
- o Authentication Method Selection: 3
- o Authentication Data: 4
- o Session Request: 5
- o Session ID: 6
- o Session OK: 8
- o Session Invalid: 9
- o Session Teardown: 10
- o Idempotence Request: 11
- o Idempotence Window: 12
- o Idempotence Expenditure: 13
- o Idempotence Accepted: 14
- o Idempotence Rejected: 15
- o Resolution Request: 16
- o IPv4 Resolution: 17
- o IPv6 Resolution: 18
- o Experimental: 64512-0xFFFF

This document also requests that IANA allocate a TCP and UDP port for SOCKS over TLS and DTLS, respectively.

16. Acknowledgments

The protocol described in this draft builds upon and is a direct continuation of SOCKS 5 [RFC1928].

17. References

17.1. Normative References

- [RFC1929] Leech, M., "Username/Password Authentication for SOCKS V5", RFC 1929, DOI 10.17487/RFC1929, March 1996, <<https://www.rfc-editor.org/info/rfc1929>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7766] Dickinson, J., Dickinson, S., Bellis, R., Mankin, A., and D. Wessels, "DNS Transport over TCP - Implementation Requirements", RFC 7766, DOI 10.17487/RFC7766, March 2016, <<https://www.rfc-editor.org/info/rfc7766>>.
- [RFC8305] Schinazi, D. and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency", RFC 8305, DOI 10.17487/RFC8305, December 2017, <<https://www.rfc-editor.org/info/rfc8305>>.

17.2. Informative References

- [I-D.ietf-tls-dtls-connection-id] Rescorla, E., Tschofenig, H., and T. Fossati, "Connection Identifiers for DTLS 1.2", draft-ietf-tls-dtls-connection-id-07 (work in progress), October 2019.
- [I-D.ietf-tls-esni] Rescorla, E., Oku, K., Sullivan, N., and C. Wood, "TLS Encrypted Client Hello", draft-ietf-tls-esni-08 (work in progress), October 2020.
- [RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 1928, DOI 10.17487/RFC1928, March 1996, <<https://www.rfc-editor.org/info/rfc1928>>.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <<https://www.rfc-editor.org/info/rfc3550>>.

- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <<https://www.rfc-editor.org/info/rfc6824>>.
- [RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <<https://www.rfc-editor.org/info/rfc7413>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

Authors' Addresses

Vladimir Olteanu
University Politehnica of Bucharest
313 Splaiul Independentei, Sector 6
Bucharest
Romania

Email: vladimir.olteanu@cs.pub.ro

Dragos Niculescu
University Politehnica of Bucharest
313 Splaiul Independentei, Sector 6
Bucharest
Romania

Email: dragos.niculescu@cs.pub.ro

Network Working Group
Internet-Draft
Intended status: Informational
Expires: December 27, 2018

T. Pauly
E. Kinnear
Apple Inc.
June 25, 2018

TCP Encapsulation Considerations
draft-pauly-tcp-encapsulation-00

Abstract

Network protocols other than TCP, such as UDP, are often blocked or suboptimally handled by network middleboxes. One strategy that applications can use to continue to send non-TCP traffic on such networks is to encapsulate datagrams or messages within in a TCP stream. However, encapsulating datagrams within TCP streams can lead to performance degradation. This document provides guidelines for how to use TCP for encapsulation, a summary of performance concerns, and some suggested mitigations for these concerns.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 27, 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	2
2. Motivations for Encapsulation	3
2.1. UDP Blocking	3
2.2. UDP NAT Timeouts	3
3. Encapsulation Formats	3
3.1. Multiplexing Flows	4
4. Deployment Considerations	5
5. Performance Considerations	5
5.1. Loss Recovery	6
5.1.1. Concern	6
5.1.2. Mitigation	6
5.2. Bufferbloat	7
5.2.1. Concern	7
5.2.2. Mitigation	8
5.3. Head of Line Blocking	8
5.3.1. Concern	8
5.3.2. Mitigation	9
6. Security Considerations	9
7. IANA Considerations	9
8. Informative References	9
Authors' Addresses	10

1. Introduction

TCP streams are sometimes used as a mechanism for encapsulating datagrams or messages, which is referred to in this document as "TCP encapsulation". Encapsulation may be used to transmit data over networks that block or suboptimally handle non-TCP traffic. The current motivations for using encapsulation generally revolve around the treatment of UDP packets (Section 2).

Implementing a TCP encapsulation strategy consists of mapping datagram messages into a stream protocol, often with a length-value record format (Section 3). While these formats are described here as applying to encapsulating datagrams in a TCP stream, the formats are equally suited to encapsulating datagrams within any stream abstraction. For example, the same format may be used for both raw TCP streams and TLS streams running over TCP.

2. Motivations for Encapsulation

The primary motivations for enabling TCP encapsulation that will be explored in this document relate mainly to the treatment of UDP packets on a given network. UDP can be used for real-time network traffic, as a mechanism for deploying non-TCP transport protocols, and as a tunneling protocol that is compatible with Network Address Translators (NATs).

2.1. UDP Blocking

Some network middleboxes block any IP packets that do not appear to be used for HTTP traffic, either as a security mechanism to block unknown traffic or as a way to restrict access to whitelisted services. Network applications that rely on UDP to transmit data will be blocked by these middleboxes. In this case, the application can attempt to use TCP encapsulation to transmit the same data over a TCP stream.

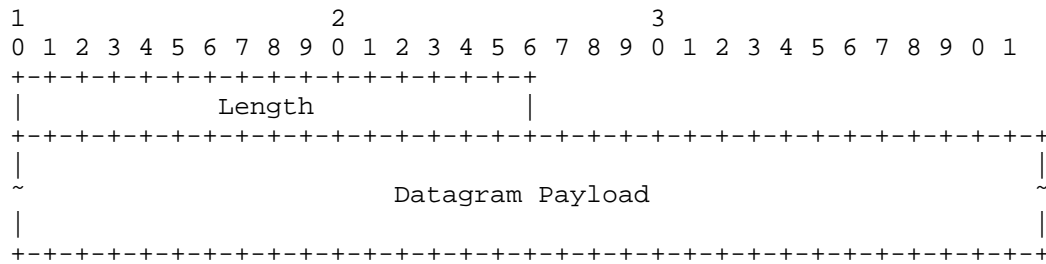
2.2. UDP NAT Timeouts

Other networks may not altogether block non-TCP traffic, but instead make other protocols unsuitable for use. For example, many Network Address Translation (NAT) devices will maintain TCP port mappings for long periods of time, since the end of a TCP stream can be detected by the NAT. Since UDP packet flows do not signal when no more packets will be sent, NATs often use short timeouts for UDP port mappings. Thus, applications can attempt to use TCP encapsulation when long-lived flows are required on networks with NATs.

3. Encapsulation Formats

The simplest approach for encapsulating datagram messages within a TCP stream is to use a length-value record format. That is, a header consisting of a length field, followed by the datagram message itself.

For example, if an encapsulation protocol uses a 16-bit length field (allowing up to 65536 bytes of datagram payload), it will use a format like the following:



The format of the length header field could be longer or shorter depending on the needs of the protocol. 16 bits is most appropriate when encapsulating datagrams that would otherwise be sent directly in IP packets, since the payload length field for an IP header is also 16 bits.

The length field must be specified to either include itself in the length of the entire record, or to only describe the length of the payload field. The protocol used for encapsulating IKE and ESP packets in TCP [RFC8229] does include the length field itself in the length of the record. This may be slightly easier for implementations to parse out records, since they will not need to add the length of the length field when finding record offsets within a stream.

3.1. Multiplexing Flows

Since TCP encapsulation is used to avoid failures caused by NATs or firewalls, some implementations re-use one TCP port or one established TCP stream for multiple kinds of encapsulated traffic. Using a single port or stream allows re-use of NAT bindings and reduces the chance that a firewall will block some flows, but not others.

If multiple kinds of traffic are multiplexed on the same listening TCP port, individual streams opened to that port need to be differentiated. This may require adding a one-time header that is sent on the stream to indicate the type of encapsulated traffic that will follow. For example, TCP encapsulated IKE [RFC8229] uses a stream prefix to differentiate its encapsulation strategy from proprietary Virtual Private Network (VPN) protocols.

Multiplexing multiple kinds of datagrams, or independent flows of datagrams, over a single TCP stream requires adding a per-record type field or marker to the encapsulation record format. For ease of parsing records, this value should be placed after the length field of the record format. For example, various ESP packet flows are identified by the four-byte Security Parameter Index (SPI) that

comprises the first bytes of the datagram payload, while IKE packets in the same TCP encapsulated stream are differentiated by using all zeros for the first four bytes.

4. Deployment Considerations

In general, any new TCP encapsulation protocol should allocate a new TCP port. If TCP is being used to encapsulate traffic that is normally sent over UDP, then the most obvious port choice for the TCP encapsulated version is the equivalent port value in the TCP port namespace.

Simply using TCP instead of UDP may be enough in some cases to mitigate the connectivity problems of using UDP with NATs and other middleboxes. However, it may be useful to also add a layer of encryption to the stream using TLS to obfuscate the contents of the stream. This may be done for security and privacy reasons, or to prevent middleboxes from mishandling encapsulated traffic or ossifying around a particular format for encapsulation.

5. Performance Considerations

Many encapsulation or tunnelling protocols utilize an underlying transport like UDP, which does not provide stateful features such as loss recovery or congestion control. Because encapsulation using TCP involves an additional layer of state that is shared among all traffic inside the tunnel, there are additional performance considerations to address.

Even though this document describes encapsulating datagrams or messages inside a TCP stream, some protocols, such as ESP, themselves often encapsulate additional TCP streams, such as when transmitting data for a VPN protocol [RFC8229]. This introduces several potential sources of suboptimal behavior, as multiple TCP contexts act upon the same traffic.

For the purposes of this discussion, we will refer to the TCP encapsulation context as the "outer" TCP context, while the TCP context applicable to any encapsulated protocol will be referred to as the "inner" TCP context.

The use of an outer TCP context may cause signals from the network to be hidden from the inner TCP contexts. Depending on the signals that the inner TCP contexts use for indicating congestion, events that would otherwise result in a modification of behavior may go unnoticed, or may build up until a large modification of behavior is necessary. Generally, the main areas of concern are signals that

inform loss recovery, Bufferbloat and delay avoidance, and head of line blocking between streams.

5.1. Loss Recovery

5.1.1. Concern

The outer TCP context experiences packet loss on the network directly, while any inner TCP contexts present observe the effects of that loss on the delivery of their packets by the encapsulation layer. Furthermore, inner TCP contexts still observe direct network effects for any network segments that are traversed outside of the encapsulation, as is common with a VPN.

In this way, the outer TCP context masks packet loss from the inner contexts by retransmitting encapsulated segments to recover from those losses. An inner context observes this as a delay while the packets are retransmitted rather than a loss. This can lead to spurious retransmissions if the recovery of the lost packets takes longer than the inner context's retransmission timeout (RTO). Since the outer context is retransmitting the packets to make up for the losses, the spurious retransmissions waste bandwidth that could be used for packets that advance the progress of the flows being encapsulated. A RTO event on an inner TCP context also hinders performance beyond generating spurious retransmissions, as many TCP congestion control algorithms dramatically reduce the sending rate after an RTO is observed.

When recovery from a loss event on the outer TCP context completes, the network or endpoint on the other end of the encapsulation will receive a potentially large burst of packets as the retransmitted packets fill in any gaps and the entire set of pending data can be delivered.

If content from multiple inner flows is shared within a single TCP packet in the outer context, the effects of lost packets from the outer context will be experienced by more than one inner flow at a time. However, this loss is actually shared by all inner flows, since forward progress for the entire encapsulation tunnel is generally blocked until the lost segments can be filled in. This is discussed further in Section 5.3.

5.1.2. Mitigation

Generally, TCP congestion controls and loss recovery algorithms are capable of recovering from loss events very efficiently, and the inner TCP contexts observe brief periods of added delay without much penalty.

A TCP congestion control should be selected and tuned to be able to gracefully handle extremely variable RTT values, which may already be the case for some congestion controls, as RTT variance is often greatly increased in mobile and cellular networks.

Additionally, use of a TCP congestion control that considers delay to be a sign of congestion may help the coordination between inner and outer TCP contexts. LEDBAT [RFC6817] and BBR [I-D.cardwell-iccr-g-bbr-congestion-control] are two examples of delay based congestion control that an inner TCP context could use to properly interpret loss events experienced by the outer TCP context. Care must be taken to ensure that any TCP congestion control in use is also appropriate for an inner context to use on any network segments that are traversed outside of the encapsulation.

Since any losses will be handled by the outer TCP context, it might seem reasonable to modify the the inner TCP contexts' loss recovery algorithms to prevent retransmissions, there are often network segments outside of the encapsulated segments that still rely on the inner contexts' loss recovery algorithms. Instead, spurious retransmissions can be reduced by ensuring that RTO values are tuned such that the outer TCP context will fully time out before any inner TCP contexts.

5.2. Bufferbloat

5.2.1. Concern

"Bufferbloat", or delay introduced by consistently full large buffers along a network path [TSV2011] [BB2011], can increase observed RTTs along a network path, which can harm the performance of latency sensitive applications. Any spurious retransmissions sent on the network take place in queues that would otherwise be filled by useful data. In this case, any retransmission sent by an inner TCP context for a loss or timeout along the network segments also covered by the outer TCP context is considered to be spurious. This can pose a performance problem for implementations that rely on interactive data transfer.

Additionally, because there may be multiple inner TCP contexts being multiplexed over a single outer TCP context, even a minor reduction in sending rate by each of the inner contexts can result in a dramatic decrease in data sent through the outer context. Similarly, an increase in sending rate is also amplified.

5.2.2. Mitigation

Great care should be taken in tuning the inner TCP congestion control to avoid spurious retransmissions as much as possible. However, in order to provide effective loss recovery for the segments of the network outside the tunnel, the set of parameters used for tuning needs to be viable both inside and outside the tunnel. Adjusting the retransmission timeout (RTO) value for the TCP congestion control on the inner TCP context to be greater than that of the out TCP context will often help to reduce the number of spurious retransmissions generated while the outer TCP context attempts to catch up with lost or reordered packets.

In most cases, fast retransmit will be sufficient to recover from losses on network segments after the inner flows leave the tunnel, although loss events that trigger a full RTO on those last-mile segments will carry a higher penalty with such tuning. However, in many deployments, the last-mile segments will often observe lower loss rates than the first-mile segments, leading to a balance that often favors spurious retransmission avoidance on the first-mile over loss recovery speed on the last-mile.

5.3. Head of Line Blocking

5.3.1. Concern

Because TCP provides in-order delivery and reliability, even if there are multiple flows being multiplexed over the encapsulation layer, loss events, spurious retransmissions, or other recovery efforts will cause data for all other flows to back up and not be delivered to the client. In deployments where there are additional network segments to traverse beyond the encapsulation boundary, this may mean that flows are not delivered onto those segments until recovery for the outer TCP context is complete.

With UDP encapsulation, packet reordering and loss did not necessarily prevent data from being delivered, even if it was delivered out of order. Because TCP groups all data being encapsulated into one outer congestion control and loss recovery context, this may cause significant delays for flows not directly impacted by a recovery event.

Reordering on the network will also cause problems in this case, as it will often trigger fast retransmissions on the outer TCP context, blocking all inner contexts from being able to deliver data until the retransmissions are complete. However, a well behaved TCP will reorder the data that arrived out of order and deliver it before the

retransmissions arrive, reducing the detrimental impact of such reordering.

5.3.2. Mitigation

One option to help address the head of line blocking would be to run multiple tunnels, one for throughput sensitive flows and one for latency sensitive flows. This can help to reduce the amount of time that a latency sensitive flow can possibly be blocked on recovery for any other flow. Latency sensitive flows should take extra care to ensure that only the necessary amount of data is in flight at any given time.

Explicit Congestion Notification (ECN) ([RFC3168], [RFC5562]) could also be used to communicate between outer and inner TCP contexts during any recovery scenario. In a strategy similar to that taken by tunnelling of ECN fields in IP-in-IP tunnels [RFC6040], if an implementation supports such behavior, any ECN markings communicated to the outer TCP context by the network could be passed through to any inner TCP contexts transported by a given packet. Alternately, an implementation could elect to pass through such markings to all inner TCP contexts if a greater reduction in sending rate was deemed to be necessary.

6. Security Considerations

Any attacker on the path that observes the encapsulation could potentially discard packets from the outer TCP context and cause significant delays due to head of line blocking. However, an attacker in a position to arbitrarily discard packets could have a similar effect on the inner TCP context directly or on any other encapsulation schemes.

7. IANA Considerations

This document has no request to IANA.

8. Informative References

[BB2011] "Bufferbloat: Dark Buffers in the Internet", n.d..

[I-D.cardwell-iccr-g-bbr-congestion-control]
Cardwell, N., Cheng, Y., Yeganeh, S., and V. Jacobson,
"BBR Congestion Control", draft-cardwell-iccr-g-bbr-
congestion-control-00 (work in progress), July 2017.

- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", RFC 5562, DOI 10.17487/RFC5562, June 2009, <<https://www.rfc-editor.org/info/rfc5562>>.
- [RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <<https://www.rfc-editor.org/info/rfc6040>>.
- [RFC6817] Shalunov, S., Hazel, G., Iyengar, J., and M. Kuehlewind, "Low Extra Delay Background Transport (LED BAT)", RFC 6817, DOI 10.17487/RFC6817, December 2012, <<https://www.rfc-editor.org/info/rfc6817>>.
- [RFC8229] Pauly, T., Touati, S., and R. Mantha, "TCP Encapsulation of IKE and IPsec Packets", RFC 8229, DOI 10.17487/RFC8229, August 2017, <<https://www.rfc-editor.org/info/rfc8229>>.
- [TSV2011] "Bufferbloat: Dark Buffers in the Internet", March 2011.

Authors' Addresses

Tommy Pauly
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: tpauly@apple.com

Eric Kinnear
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America

Email: ekinnear@apple.com