

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: June 07, 2018

G. Fairhurst
T. Jones
University of Aberdeen
M. Tuexen
I. Ruengeler
Muenster University of Applied Sciences
December 6, 2017

Packetization Layer Path MTU Discovery for Datagram Transports
draft-fairhurst-tsvwg-datagram-plpmtud-02

Abstract

This document describes a robust method for Path MTU Discovery (PMTUD) for datagram Packetization layers. The method allows a Packetization layer (or a datagram application that uses it) to probe an network path with progressively larger packets to determine a maximum packet size. The document describes as an extension to RFC 1191 and RFC 8201, which specify ICMP-based Path MTU Discovery for IPv4 and IPv6. This provides functionality for datagram transports that is equivalent to the Packetization layer PMTUD specification for TCP, specified in RFC4821.

When published, this specification updates RFC4821.

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 June 07, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
2. Terminology	4
3. Features required to provide Datagram PLPMTUD	6
3.1. PMTU Probe Packets	8
3.2. Validation of the current effective PMTU	9
3.3. Reduction of the effective PMTU	10
4. Datagram Packetization Layer PMTUD	10
4.1. Probing	10
4.2. Verification and use of PTB messages	11
4.3. Timers	11
4.4. Constants	12
4.5. Variables	12
4.6. State Machine	13
5. Specification of Protocol-Specific Methods	15
5.1. DPLPMTUD for UDP and UDP-Lite	16
5.1.1. UDP Options	16
5.1.2. UDP Options required for PLPMTUD	16
5.1.2.1. Echo Request Option	16
5.1.2.2. Echo Response Option	16
5.1.3. Sending UDP-Option Probe Packets	17
5.1.4. Validating the Path with UDP Options	17
5.1.5. Handling of PTB Messages by UDP	17
5.2. DPLPMTUD for SCTP	17
5.2.1. SCTP/IP4 and SCTP/IPv6	17
5.2.1.1. Sending SCTP Probe Packets	18
5.2.1.2. Validating the Path with SCTP	18
5.2.1.3. PTB Message Handling by SCTP	18
5.2.2. DPLPMTUD for SCTP/UDP	18
5.2.2.1. Sending SCTP/UDP Probe Packets	18
5.2.2.2. Validating the Path with SCTP/UDP	18
5.2.2.3. Handling of PTB Messages by SCTP/UDP	19
5.2.3. DPLPMTUD for SCTP/DTLS	19
5.2.3.1. Sending SCTP/DTLS Probe Packets	19
5.2.3.2. Validating the Path with SCTP/DTLS	19
5.2.3.3. Handling of PTB Messages by SCTP/DTLS	19
5.3. Other IETF Transports	19
5.4. DPLPMTUD by Applications	19
6. Acknowledgements	20
7. IANA Considerations	20
8. Security Considerations	20
9. References	20
9.1. Normative References	20

9.2. Informative References	22
Appendix A. Event-driven state changes	22
Appendix B. Revision Notes	25
Authors' Addresses	26

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).

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]). It adjusts the effective Path MTU (PMTU), based on reception of ICMP Path too Big (PTB) messages to decrease the PMTU when a packet is sent with a size larger than the value supported along a path, and a method that from time-to-time increases the packet size in attempt to discover an increase in the supported PMTU.

However, Classical PMTUD is subject to protocol failures. One failure arises when traffic using a packet size larger than the actual supported PMTU is black-holed (all datagrams sent with this size are silently discarded). This could continue to happen when ICMP PTB messages are not delivered back to the sender for some reason [RFC2923]). For example, ICMP messages are increasingly filtered by middleboxes (including firewalls) [RFC4890], and in some cases are not correctly processed by tunnel endpoints.

Another failure could result if a system not on the network path sends a PTB that attempts to force the sender to change the effective PMTU [RFC8201]. A sender can protect itself from reacting to such messages by utilising the quoted packet within the PTB message payload to verify that the received PTB message was generated in response to a packet that had actually been sent. However, there are situations where a sender is unable to provide this verification (e.g., when the PTB message does not include sufficient information, often the case for IPv4; or where the information corresponds to an encrypted packet). Most routers implement RFC792 [RFC0792], which requires them to return only the first 64 bits of the IP payload of the packet, whereas RFC1812 [RFC1812] requires routers to return the full packet if possible.

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

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

performed by other encapsulation methods working above the transport. PTB verification is more straight forward at the PL or at a higher layer.

In contrast to PMTUD, Packetization Layer Path MTU Discovery (PLPMTUD) [RFC4821] does not rely upon reception and verification of PTB messages. It is therefore more robust than Classical PMTUD. This has become the recommended approach for implementing PMTU discovery with TCP. It uses a general strategy where the PL searches for an appropriate PMTU by sending probe packets along the network path with a progressively larger packet size. If a probe packet is successfully delivered (as determined by the PL), then the effective Path MTU is raised to the size of the successful probe.

PLPMTUD introduces flexibility in the implementation of PMTU discovery. At one extreme, it can be configured to only perform PTB black hole detection and recovery to increase the robustness of Classical PMTUD, or at the other extreme, all PTB processing can be disabled and PLPMTUD can completely replace Classical PMTUD. PLPMTUD can also include additional consistency checks without increasing the risk of increased blackholing.

The UDP-Guidelines [RFC8085] state "an application SHOULD either use the path MTU information provided by the IP layer or implement Path MTU Discovery (PMTUD)", but does not provide a mechanism for discovering the largest size of unfragmented datagram than can be used on a path. PLPMTUD has not currently been specified for UDP, while Section 10.2 of [RFC4821] recommends a PLPMTUD probing method for SCTP that utilises heartbeat messages as probe packets, but does not provide a complete specification. This document provides the details to complete that specification. Similarly, the method defined in this specification could be used with the Datagram Congestion Control Protocol (DCCP) [RFC4340] requires implementations to support Classical PMTUD and states that a DCCP sender "MUST maintain the maximum packet size (MPS) allowed for each active DCCP session". It also defines the current congestion control maximum packet size (CCMPS) supported by a 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.

Section 4 of this document presents a set of algorithms for datagram protocols to discover a maximum size for the effective PMTU across a path. The methods described rely on features of the PL Section 3 and apply to transport protocols over IPv4 and IPv6. It does not require cooperation from the lower layers (except that they are consistent about which packet sizes are acceptable). A method can utilise ICMP PTB messages when received messages are made available to the PL.

Finally, Section 5 specifies the method for a set of transports, and provides information to enables the implementation of PLPMTUD with other datagram transports and applications that use datagram transports.

2. Terminology

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

Other terminology is directly copied from [RFC4821], and the definitions in [RFC1122].

Black-Holed: When the sender is unaware that packets are not delivered to the destination endpoint (e.g., when the sender transmits packets of a particular size with a previously known PMTU, but is unaware of a change to the path that resulted in a smaller PMTU).

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

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

Effective PMTU: The current estimated value for PMTU that is used by a Packetization Layer.

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

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

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

Link MTU: The 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 MTU. 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 link MTU to include the link layer headers.

MPS: The Maximum Packet Size (MPS), the largest size of application data block that can be sent unfragmented across a path. In PLPMTUD this quantity is derived from Effective PMTU by taking into consideration the size of the application and lower protocol layer headers, and can be limited by the application protocol.

Packet: An IP header plus the IP payload.

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

Path: The set of link and routers traversed by a packet between a source node and a destination node.

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

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

Probe packet: A datagram sent with a purposely chosen size (typically larger than the current Effective PMTU or MPS) to detect if messages of this size can be successfully sent along the end-to-end path.

3. Features required to provide Datagram PLPMTUD

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

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

1. **PMTU parameters:** A PLPMTUD sender is REQUIRED to provide information about the maximum size of packet that can be transmitted by the sender on the local link (the Link MTU and MAY utilize similar information about the receiver when this is supplied (note this could be less than EMTU_R). Some applications also have a maximum transport protocol data unit (PDU) size, in which case there is no benefit from probing for a size larger than this (unless a transport allows multiplexing multiple applications PDUs into the same datagram).
2. **Effective PMTU:** A datagram application MUST be able to choose the size of datagrams sent to the network, up to the effective PMTU, or a smaller value (such as the MPS) derived from this. This value is managed by the PMTUD method. The effective PMTU (specified in Section 1 of [RFC1191]) is equivalent to the EMTU_S (specified in [RFC1122]).

3. Probe packets: On request, a PLPMTUD sender is REQUIRED to be able to transmit a packet larger than the current effective PMTU (but always with a total size less than the link MTU). The method can use this as a probe packet. In IPv4, a probe packet is always sent with the Don't Fragment (DF) bit set and without network layer endpoint fragmentation. In IPv6, a probe packet is always sent without source fragmentation (as specified in section 5.4 of [RFC8201]).
4. Processing PTB messages: A PLPMTUD sender MAY optionally utilize PTB messages received from the network layer to help identify when a path does not support the current size of packet probe. Any received PTB message SHOULD/MUST be verified before it is used to update the PMTU discovery information [RFC8201]. This verification confirms that the PTB message was sent in response to a packet originating by the sender, and needs to be performed before the PMTU discovery method reacts to the PTB message. When the router link MTU is indicated in the PTB message this MAY be used by datagram PLPMTUD to reduce the size of a probe, but MUST NOT be used increase the effective PMTU ([RFC8201]).
5. Reception feedback: The destination PL endpoint is REQUIRED to provide a feedback method that indicates when a probe packet has been received by the destination endpoint. The local PL endpoint at the sending node is REQUIRED to pass this feedback to the sender-side PLPMTUD method.
6. Probing and congestion control: The isolated loss of a probe packet SHOULD NOT be treated as an indication of congestion and its loss does not directly trigger a congestion control reaction [RFC4821].
7. Probe loss recovery: If the data block carried by a probe message needs to be sent reliably, the PL (or layers above) MUST arrange retransmission/repair of any resulting loss. This method MUST be robust in the case where probe packets are lost due to other reasons (including link transmission error, congestion). The PLPMTUD method treats isolated loss of a probe packet (with or without an PTB message) as a potential indication of a PMTU limit on the path. The PL MAY retransmit any data included in a lost probe packet without adjusting its congestion window [RFC4821].
8. Cached effective PMTU: The sender MUST cache the effective PMTU value used by an instance of the PL between probes and needs also

to consider the disruption that could be incurred by an unsuccessful probe - both upon the flow that incurs a probe loss, and other flows that experience the effect of additional probe traffic.

9. Shared effective PMTU state: The PMTU value could also be stored with the corresponding entry in the destination cache and used by other PL instances. The specification of PLPMTUD [RFC4821] states: "If PLPMTUD updates the MTU for a particular path, all Packetization Layer sessions that share the path representation (as described in Section 5.2 of [RFC4821]) SHOULD be notified to make use of the new MTU and make the required congestion control adjustments". Such methods need to be robust to the wide variety of underlying network forwarding behaviours. 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 design principles are stated:

- o Suitable MPS: The PLPMTUD method SHOULD avoid forcing an application to use an arbitrary small MPS (effective PMTU) for transmission while the method is searching for the currently supported PMTU. Datagram PLs do not necessarily support fragmentation of PDUs larger than the PMTU. A reduced MPS can adversely impact the performance of a datagram application.
- o Path validation: The PLPMTUD method MUST be robust to path changes that could have occurred since the path characteristics were last confirmed.
- o Datagram reordering: A method MUST be robust to the possibility that a flow encounters reordering, or has the traffic (including probe packets) is divided over more than one network path.
- o When to probe: The PLPMTUD method SHOULD determine whether the path capacity has increased since it last measured the path. This determines when the path should again be probed.

3.1. PMTU Probe Packets

PMTU discovery relies upon the sender being able to generate probe messages with a specific size. TCP is able to generate probe packets by choosing to appropriately segment data being sent [RFC4821].

In contrast, a datagram PL that needs to construct a probe packet has to either request an application to send a data block that is larger than that generated by an application, or to utilise padding functions to extend a datagram beyond the size of the application data block. Protocols that permit exchange of control messages (without an application data block) could alternatively prefer to generate a probe packet by extending a control message with padding data.

When the method fails to validate the PMTU for the path, it may be required to send a probe packet with a size less than the size of the data block generated by an application. In this case, the PL could provide a way to fragment a datagram at the PL, or could instead utilise a control packet with padding.

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

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

Probing using application data: A probe packet that contains a data block supplied by an application that matches the size required for the probe. This method requests the application to issue a data block of the desired probe size. If the application/transport needs protection from the loss of an unsuccessful probe packet, the application/transport needs then to perform transport-layer retransmission/repair of the data block (e.g., by retransmission after loss is detected or by duplicating the data block in a datagram without the padding).

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

Probing using padding data: A probe packet that contains only control information together with any padding needed to inflate the packet to the size required for the probe. 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.

A datagram PLPMTUD MAY choose to use only one of these methods to simplify the implementation.

3.2. Validation of the current effective PMTU

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

Transport protocols can include end-to-end methods that detect and report reception of specific datagrams that they send (e.g., DCCP and

SCTP provide keep-alive/heartbeat features). When supported, this mechanism SHOULD also be used by PLPMTUD to acknowledge reception of a probe packet.

A PL that does not acknowledge data reception (e.g., UDP and UDP-Lite) is unable to detect when the packets it sends are discarded because their size is greater than the actual PMTUD. These PLs need to either rely on an application protocol to detect this, or make use of an additional transport method such as UDP-Options [I-D.ietf-tsvwg-udp-options]. In addition, they might need to send reachability probes (e.g., periodically solicit a response from the destination) to determine whether the current effective PMTU is still supported by the network path.

Section 4 specifies this function for a set of IETF-specified protocols.

3.3. Reduction of the effective PMTU

When the current effective PMTU is no longer supported by the network path, the transport needs to detect this and reduce the effective PMTU.

- o A PL that sends a datagram larger than the actual PMTU that includes no application data block, or one that does not attempt to provide any retransmission, can send a new probe packet with an updated probe size.
- o A PL that wishes to resend the application data block, could then need to re-fragment the data block to a smaller packet size that is expected to traverse the end-to-end path. This could utilise network-layer or PL fragmentation when these are available. A fragmented datagram MUST NOT be used as a probe packet (see [RFC8201]).

A method can additionally utilise PTB messages to detect when the actual PMTU supported by a network path is less than the current size of datagrams (or probe messages) that are being sent.

4. Datagram Packetization Layer PMTUD

This section specifies Datagram PLPMTUD.

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

4.1. Probing

The PLPMTUD method utilises a timer to trigger the generation of probe packets. The `probe_timer` is started each time a probe packet is sent to the destination and is cancelled when receipt of the probe packet is acknowledged.

The `PROBE_COUNT` is initialised to zero when a probe packet is first sent with a particular size. Each time the `probe_timer` expires, the `PROBE_COUNT` is incremented, and a probe packet of the same size is retransmitted. The maximum number of retransmissions per probing size is configured (`MAX_PROBES`). If the value of the `PROBE_COUNT` reaches `MAX_PROBES`, probing will be stopped and the last successfully probed PMTU is set as the effective PMTU.

Once probing is completed, the sender continues to use the effective PMTU until either a PTB message is received or the `PMTU_RAISE_TIMER` expires. If the PL is unable to verify reachability to the destination endpoint after probing has completed, the method uses a `REACHABILITY_TIMER` to periodically repeat a probe packet for the current effective PMTU size, while the `PMTU_RAISE_TIMER` is running. If the resulting probe packet is not acknowledged (i.e. the `PROBE_TIMER` expires), the method re-starts probing for the PMTU.

4.2. Verification and use of PTB messages

XXX A decision on SHOULD/MUST needs to be made XXX

A node that receives a PTB message from a router or middlebox, SHOULD /MUST verify the PTB message. The node checks the protocol information in the quoted payload to verify that the message originated from the sending node. The node also checks that the reported MTU size is less than the size used by packet probes. PTB messages are discarded if they fail to pass these checks, or where there is insufficient ICMP payload to perform these checks. The checks are intended to provide protection from packets that originate from a node that is not on the network path or a node that attempts to report a larger MTU than the current probe size.

PTB messages that have been verified can be utilised by the DPLPMTUD algorithm. A method that utilises these PTB messages can improve performance compared to one that relies solely on probing.

4.3. Timers

This method utilises three timers:

`PROBE_TIMER`: Configured to expire after a period longer than the maximum time to receive an acknowledgment to a probe packet. This value MUST be larger than 1 second, and SHOULD be larger than 15 seconds. Guidance on selection of the timer value are provide in

section 3.1.1 of the UDP Usage Guidelines [RFC8085].

PMTU_RAISE_TIMER: Configured to the period a sender ought to continue use the current effective PMTU, after which it re-commences probing for a higher PMTU. This timer has a period of 600 secs, as recommended by PLPMTUD [RFC4821].

REACHABILITY_TIMER: Configured to the period a sender ought to wait before confirming the current effective PMTU is still supported. This is less than the PMTU_RAISE_TIMER.

An application that needs to employ keep-alive messages to deliver useful service over UDP SHOULD NOT transmit them more frequently than once every 15 seconds and SHOULD use longer intervals when possible. DPLPMTUD ought to suspend reachability probes when no application data has been sent since the previous probe packet. Guidance on selection of the timer value are provide in section 3.1.1 of the UDP Usage Guidelines[RFC8085].

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

4.4. Constants

The following constants are defined:

MAX_PROBES: The maximum value of the PROBE_ERROR_COUNTER. The default value of MAX_PROBES is 10.

MIN_PMTU: The smallest allowed probe packet size. This value is 1280 bytes, as specified in [RFC2460]. For IPv4, the minimum value is 68 bytes. (An IPv4 routed is required to be able to forward a datagram of 68 octets without further fragmentation. This is the combined size of an IPv4 header and the minimum fragment size of 8 octets.)

BASE_PMTU: The BASE_PMTU is a considered a size that ought to work in most cases. The size is equal to or larger than the minimum permitted and smaller than the maximum allowed. In the case of IPv6, this value is 1280 bytes [RFC2460]. When using IPv4, a size of 1200 is RECOMMENDED.

MAX_PMTU: The MAX_PMTU is the largest size of PMTU that is probed. This has to be less than or equal to the minimum of the local MTU of the outgoing interface and the destination effective MTU for receiving. An application or PL may reduce this when it knows there is no need to send packets above a specific size.

4.5. Variables

This method utilises a set of variables:

effective PMTU: The effective PMTU is the maximum size of datagram that the method has currently determined can be supported along the entire path.

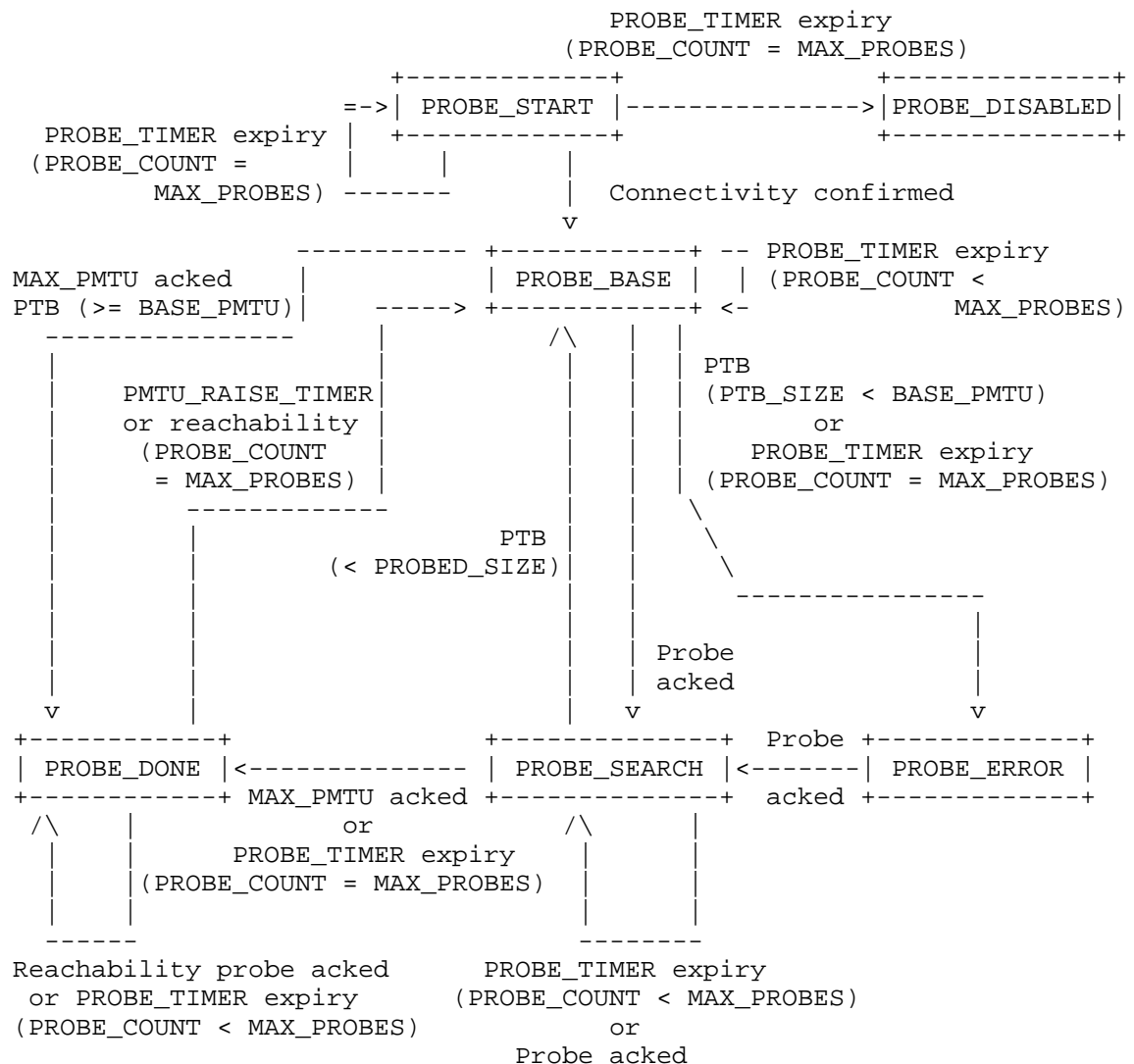
PROBED_SIZE: The PROBED_SIZE is the size of the current probe packet. This is a tentative value for the effective PMTU, which is awaiting confirmation by an acknowledgment.

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

PTB_SIZE: The PTB_Size is value returned by a verified PTB message indicating the local MTU size of a router along the path.

4.6. State Machine

A state machine for Datagram PLPMTUD is depicted in Figure 1. If multihoming is supported, a state machine is needed for each active path.



The following states are defined to reflect the probing process:

PROBE_START: The PROBE_START state is the initial state before probing has started. PLPMTUD is not performed in this state. The state transitions to PROBE_BASE, when a path has been confirmed, i.e. when a sent packet has been acknowledged on this path. The effective PMTU is set to the BASE_PMTU size. Probing ought to start immediately after connection setup to prevent the loss of user data.

PROBE_BASE: The PROBE_BASE state is the starting point for probing with datagram PLPMTUD. It is used to confirm whether the BASE_PMTU size is supported by the network path. On entry, the PROBED_SIZE is set to the BASE_PMTU size and the PROBE_COUNT is set to zero. A probe packet is sent, and the PROBE_TIMER is started. The state is left when the PROBE_COUNT reaches MAX_PROBES; a PTB message is verified, or a probe packet is acknowledged.

PROBE_SEARCH: The PROBE_SEARCH state is the main probing state. This state is entered either when probing for the BASE_PMTU was successful or when there is a successful reachability test in the PROBE_ERROR state. On entry, the effective PMTU is set to the last acknowledged PROBED_SIZE.

On the first probe packet for each probed size, the PROBE_COUNT is set to zero. Each time a probe packet is acknowledged, the effective PMTU is set to the PROBED_SIZE, and then the PROBED_SIZE is increased. When a probe packet is not acknowledged within the period of the PROBE_TIMER, the PROBE_COUNT is incremented and the probe packet is retransmitted. The state is exited when the PROBE_COUNT reaches MAX_PROBES; a PTB message is verified; or a probe of size PMTU_MAX is acknowledged.

PROBE_ERROR: The PROBE_ERROR state represents the case where the network path is not known to support an effective PMTU of at least the BASE_PMTU size. It is entered when either a probe of size BASE_PMTU has not been acknowledged or a verified PTB message indicates a smaller link MTU than the BASE_PMTU. On entry, the PROBE_COUNT is set to zero and the PROBED_SIZE is set to the MIN_PMTU size, and the effective PMTU is reset to MIN_PMTU size. In this state, a probe packet is sent, and the PROBE_TIMER is started. The state transitions to the PROBE_SEARCH state when a probe packet is acknowledged.

PROBE_DONE: The PROBE_DONE state indicates a successful end to a probing phase. Datagram PLPMTUD remains in this state until either the PMTU_RAISE_TIMER expires or a PTB message is verified.

When PLPMTUD uses an unacknowledged PL and is in the PROBE_DONE state, a REACHABILITY_TIMER periodically resets the PROBE_COUNT and schedules a probe packet with the size of the effective PMTU. If the probe packet fails to be acknowledged after MAX_PROBES attempts, the method enters the PROBE_BASE state. When used with an acknowledged PL (e.g., SCTP), DPLPMTUD SHOULD NOT continue to probe in this state.

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

Appendix Appendix A contains an informative description of key events.

5. Specification of Protocol-Specific Methods

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

5.1. DPLPMTUD for UDP and 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, these transports do not provide the transport layer features needed to implement datagram PLPMTUD, and any support for Datagram PLPMTUD would therefore need to rely on higher-layer protocol features [RFC8085].

5.1.1. UDP Options

UDP-Options [I-D.ietf-tsvwg-udp-options] supply the additional functionality required to implement datagram PLPMTUD. This enables padding to be added to UDP datagrams and can be used to provide feedback acknowledgement of received probe packets.

5.1.2. UDP Options required for PLPMTUD

This subsection proposes two new UDP-Options that add support for requesting a datagram response be sent and to mark this datagram as a response to a request.

XXX << Future versions of the spec may define a parameter in an Option to indicate the EMTU_R to the peer.>>

5.1.2.1. Echo Request Option

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

The Echo Request carries a four byte token set by the sender. This token can be set to a value that is likely to be known only to the sender (and becomes known to nodes along the end-to-end path). The sender can then check the value returned in the response to provide additional protection from off-path insertion of data [RFC8085].

```

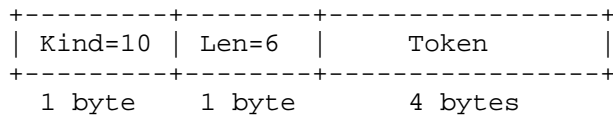
+-----+-----+-----+
| Kind=9 | Len=6 | Token          |
+-----+-----+-----+
1 byte   1 byte   4 bytes

```

5.1.2.2. Echo Response Option

The Echo Response Option is generated by the PL in response to reception of a previously received Echo Request. The Token field associates the response with the Token value carried in the most recently-received Echo Request. The rate of generation of UDP

packets carrying an Echo Response Option MAY be rate-limited.



5.1.3. Sending UDP-Option Probe Packets

This method specifies a probe packet that does not carry an application data block. The probe packet consists of a UDP datagram header followed by a UDP Option containing the ECHOREQ option, which is followed by NOP Options to pad the remainder of the datagram payload to the probe size. NOP padding is used to control the length of the probe packet.

A UDP Option carrying the ECHORES option is used to provide feedback when a probe packet is received at the destination endpoint.

5.1.4. Validating the Path with UDP Options

Since UDP is an unacknowledged PL, a sender that does not have higher-layer information confirming correct delivery of datagrams SHOULD implement the REACHABILITY_TIMER to periodically send probe packets while in the PROBE_DONE state.

5.1.5. Handling of PTB Messages by UDP

Normal ICMP verification MUST be performed as specified in Section 5.2 of [RFC8085]. This requires that the PL verifies each received PTB messages to verify these are received in response to transmitted traffic and that the reported LInk MTU is less than the current probe size. A verified PTB message MAY be used as input to the PLPMTUD algorithm.

5.2. DPLPMTUD for SCTP

Section 10.2 of [RFC4821] specifies a recommended PLPMTUD probing method for SCTP. It recommends the use of the PAD chunk, defined in [RFC4820] to be attached to a minimum length HEARTBEAT chunk to build a probe packet. This enables probing without affecting the transfer of user messages and without interfering with congestion control. This is preferred to using DATA chunks (with padding as required) as path probes.

XXX << Future versions of this specification might define a parameter contained in the INIT and INIT ACK chunk to indicate the MTU to the peer. However, multihoming makes this a bit complex, so it might not be worth doing.>>

5.2.1. SCTP/IP4 and SCTP/IPv6

The base protocol is specified in [RFC4960].

5.2.1.1. Sending SCTP Probe Packets

Probe packets consist of an SCTP common header followed by a HEARTBEAT chunk and a PAD chunk. The PAD chunk is used to control the length of the probe packet. The HEARTBEAT chunk is used to trigger the sending of a HEARTBEAT ACK chunk. The reception of the HEARTBEAT ACK chunk acknowledges reception of a successful probe.

The HEARTBEAT chunk carries a Heartbeat Information parameter which should include, besides the information suggested in [RFC4960], the probing size, which is the MTU size the complete datagram will add up to. The size of the PAD chunk is therefore computed by reducing the probing size by the IPv4 or IPv6 header size, the SCTP common header, the HEARTBEAT request and the PAD chunk header. The payload of the PAD chunk contains arbitrary data.

To avoid fragmentation of retransmitted data, probing starts right after the handshake, before data is sent. Assuming normal behaviour (i.e., the PMTU is smaller than or equal to the interface MTU), this process will take a few round trip time periods depending on the number of PMTU sizes probed. The Heartbeat timer can be used to implement the PROBE_TIMER.

5.2.1.2. Validating the Path with SCTP

Since SCTP provides an acknowledged PL, a sender does NOT implement the REACHABILITY_TIMER while in the PROBE_DONE state.

5.2.1.3. PTB Message Handling by SCTP

Normal ICMP verification 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 verified, the router Link MTU indicated in the PTB message SHOULD be used with the PLPMTUD algorithm, providing that the reported Link MTU is less than the current probe size.

5.2.2. DPLPMTUD for SCTP/UDP

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

5.2.2.1. Sending SCTP/UDP Probe Packets

Packet probing can be performed as specified in Section 5.2.1.1. The maximum payload is reduced by 8 bytes, which has to be considered when filling the PAD chunk.

5.2.2.2. Validating the Path with SCTP/UDP

Since SCTP provides an acknowledged PL, a sender does MUST NOT implement the REACHABILITY_TIMER while in the PROBE_DONE state.

5.2.2.3. Handling of PTB Messages by SCTP/UDP

Normal ICMP verification 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 verification is completed, the router Link MTU size indicated in the PTB message SHOULD be used with the PLPMTUD algorithm providing that the reported Link MTU is less than the current probe size.

5.2.3. DPLPMTUD for SCTP/DTLS

The Datagram Transport Layer Security (DTLS) encapsulation of SCTP is specified in [I-D.ietf-tsvwg-sctp-dtls-encaps]. It is used for data channels in WebRTC implementations.

5.2.3.1. Sending SCTP/DTLS Probe Packets

Packet probing can be done as specified in Section 5.2.1.1.

5.2.3.2. Validating the Path with SCTP/DTLS

Since SCTP provides an acknowledged PL, a sender does MUST NOT implement the REACHABILITY_TIMER while in the PROBE_DONE state.

5.2.3.3. Handling of PTB Messages by SCTP/DTLS

It is not possible to perform normal ICMP verification as specified in [RFC4960], since even if the ICMP message payload contains sufficient information, the reflected SCTP common header would be encrypted. Therefore it is not possible to process PTB messages at the PL.

5.3. Other IETF Transports

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

XXX << This section will be completed in a future revision of this ID >>

5.4. DPLPMTUD by Applications

Applications that use the Datagram API (e.g., applications built directly or indirectly on UDP) can implement DPLPMTUD. Some primitives used by DPLPMTUD might not be available via this interface (e.g., the ability to access the PMTU cache, or interpret received ICMP PTB messages).

In addition, it is important that PMTUD is not performed by multiple protocol layers.

XXX << This section will be completed in a future revision of this ID >>

6. Acknowledgements

This work was partially funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644334 (NEAT). The views expressed are solely those of the author(s).

7. IANA Considerations

This memo includes no request to IANA.

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

If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

8. Security Considerations

The security considerations for the use of UDP and SCTP are provided in the references RFCs. Security guidance for applications using UDP is provided in the UDP-Guidelines [RFC8085].

PTB messages could potentially be used to cause a node to inappropriately reduce the effective PMTU. A node supporting PLPMTUD SHOULD/MUST appropriately verify 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.

XXX Determine if parallel forwarding paths needs to be considred XXX

A node performing PLPMTUD could experience conflicting information about the size of supported probe packets. This could occur when there are multiple paths are concurrently in use and these exhibit a different PMTU. If not considered, this could result in data being blackholed when the effective PMTU is larger than the smallest PMTU across the current paths.

9. References

9.1. Normative References

[I-D.ietf-quic-transport]
Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Internet-Draft draft-ietf-quic-transport-04, June 2017.

- [I-D.ietf-tsvwg-sctp-dtls-encaps]
Tuexen, M., Stewart, R., Jesup, R. and S. Loreto, "DTLS Encapsulation of SCTP Packets", Internet-Draft draft-ietf-tsvwg-sctp-dtls-encaps-09, January 2015.
- [I-D.ietf-tsvwg-udp-options]
Touch, J., "Transport Options for UDP", Internet-Draft draft-ietf-tsvwg-udp-options-01, June 2017.
- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <<http://www.rfc-editor.org/info/rfc768>>.
- [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, <<http://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>>.
- [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>>.
- [RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <<http://www.rfc-editor.org/info/rfc2460>>.
- [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, <<http://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>>.
- [RFC8085] Eggert, L., Fairhurst, G. and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <<http://www.rfc-editor.org/info/rfc8085>>.
- [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>>.

9.2. Informative References

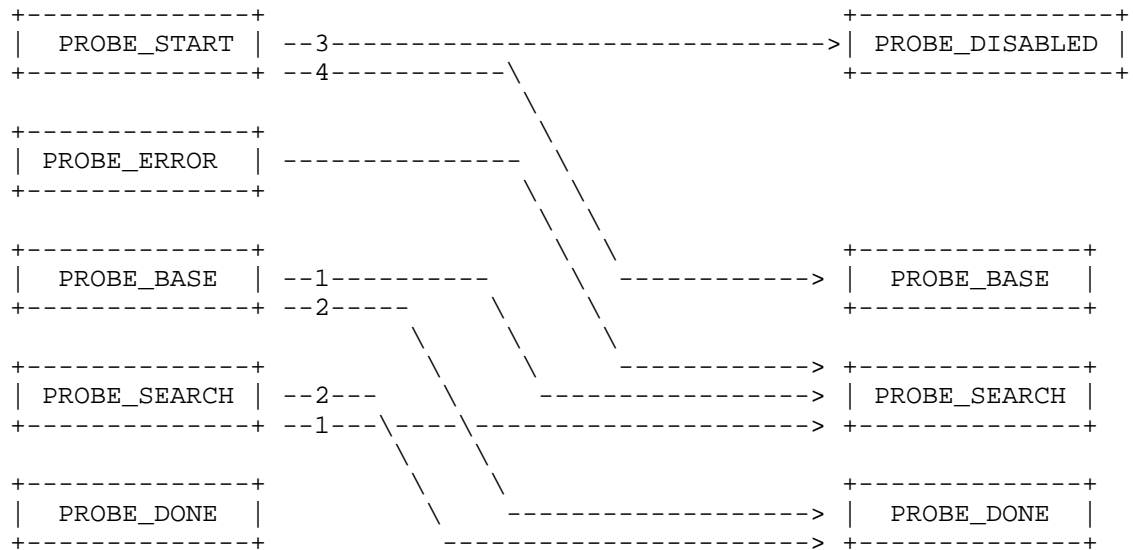
- [RFC1191] Mogul, J.C. and S.E. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990, <<http://www.rfc-editor.org/info/rfc1191>>.
- [RFC2923] Lahey, K., "TCP Problems with Path MTU Discovery", RFC 2923, DOI 10.17487/RFC2923, September 2000, <<https://www.rfc-editor.org/info/rfc2923>>.
- [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>>.
- [RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, DOI 10.17487/RFC4821, March 2007, <<http://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, <<http://www.rfc-editor.org/info/rfc4890>>.

Appendix A. Event-driven state changes

This appendix contains an informative description of key events:

Path Setup: When a new path is initiated, the state is set to PROBE_START. As soon as the path is confirmed, the state changes to PROBE_BASE and the probing mechanism for this path is started. A probe packet with the size of the BASE_PMTU is sent.

Arrival of an Acknowledgment: Depending on the probing state, the reaction differs according to Figure 4, which is just a simplification of Figure 1 focusing on this event.



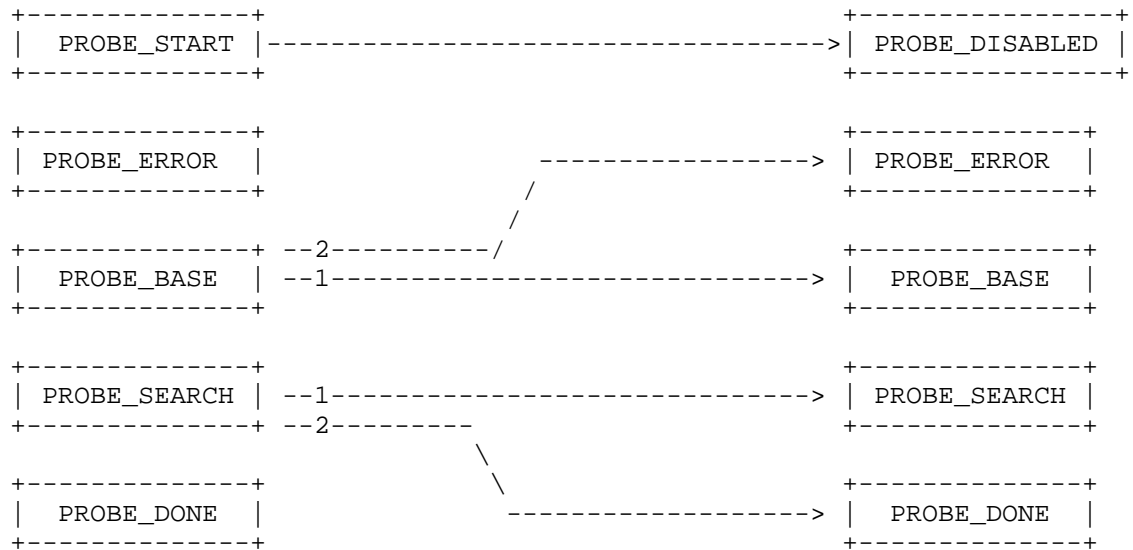
Condition 1: The maximum PMTU size has not yet been reached.

Condition 2: The maximum PMTU size has been reached. Condition 3:

Probe Timer expires and PROBE_COUNT = MAX_PROBES. Condition 4:

PROBE_ACK received.

Probing timeout: The PROBE_COUNT is initialised to zero each time the value of PROBED_SIZE is changed. The PROBE_TIMER is started each time a probe packet is sent. It is stopped when an acknowledgment arrives that confirms delivery of a probe packet. If the probe packet is not acknowledged before, the PROBE_TIMER expires, the PROBE_ERROR_COUNTER is incremented. When the PROBE_COUNT equals the value MAX_PROBES, the state is changed, otherwise a new probe packet of the same size (PROBED_SIZE) is resent. The state transitions are illustrated in Figure 5. This shows a simplification of Figure 1 with a focus only on this event.



Condition 1: The maximum number of probe packets has not been reached. Condition 2: The maximum number of probe packets has been reached.

PMTU raise timer timeout: The path through the network can change over time. It is impossible to discover whether a path change has increased in the actual PMTU by exchanging packets less than or equal to the effective PMTU. This requires PLPMTUD to periodically send a probe packet to detect whether a larger PMTU is possible. This probe packet is generated by the `PMTU_RAISE_TIMER`. When the timer expires, probing is restarted with the `BASE_PMTU` and the state is changed to `PROBE_BASE`.

Arrival of an ICMP message: The active probing of the path can be supported by the arrival of PTB messages sent by routers or middleboxes with a link MTU that is smaller than the probe packet size. If the PTB message includes the router link MTU, three cases can be distinguished:

1. The indicated link MTU in the PTB message is between the already probed and effective MTU and the probe that triggered the PTB message.
2. The indicated link MTU in the PTB message is smaller than the effective PMTU.
3. The indicated link MTU in the PTB message is equal to the `BASE_PMTU`.

In first case, the PROBE_BASE state transitions to the PROBE_ERROR state. In the PROBE_SEARCH state, a new probe packet is sent with the sized reported by the PTB message. Its result is handled according to the former events.

The second case could be a result of a network re-configuration. If the reported link MTU in the PTB message is greater than the BASE_MTU, the probing starts again with a value of PROBE_BASE. Otherwise, the method enters the state PROBE_ERROR.

In the third case, the maximum possible PMTU has been reached. This is probed again, because there could be a link further along the path with a still smaller MTU.

Note: Not all routers include the link MTU size when they send a PTB message. If the PTB message does not indicate the link MTU, the probe is handled in the same way as condition 2 of Figure 5.

Appendix B. Revision Notes

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

Individual draft -00:

- o Comments and corrections are welcome directly to the authors or via the IETF TSVWG working group mailing list.
- o This update is proposed for WG comments.

Individual draft -01:

- o Contains the first representation of the algorithm, showing the states and timers
- o This update is proposed for WG comments.

Individual draft -02:

- o Contains updated representation of the algorithm, and textual corrections.
- o The text describing when to set the effective PMTU has not yet been verified by the authors
- o To determine security to off-path-attacks: We need to decide whether a received PTB message SHOULD/MUST be verified? The text on how to handle a PTB message indicating a link MTU larger than the probe has yet not been verified by the authors
- o No text currently describes how to handle inconsistent results from arbitrary re-routing along different parallel paths

- o This update is proposed for WG comments.

Authors' Addresses

Godred Fairhurst
University of Aberdeen
School of Engineering
Fraser Noble Building
Aberdeen, AB24 3U
UK

Email: gorry@erg.abdn.ac.uk

Tom Jones
University of Aberdeen
School of Engineering
Fraser Noble Building
Aberdeen, AB24 3U
UK

Email: tom@erg.abdn.ac.uk

Michael Tuexen
Muenster University of Applied Sciences
Stegerwaldstrasse 39
Steinfurt, 48565
DE

Email: tuexen@fh-muenster.de

Irene Ruengeler
Muenster University of Applied Sciences
Stegerwaldstrasse 39
Steinfurt, 48565
DE

Email: i.ruengeler@fh-muenster.de

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@cspcrkins.org
URI: <https://cspcrkins.org/>

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: April 25, 2019

Baker

Finzi
TTTech Computertechnik AG
Frances
ISAE-SUPAERO
Kuhn
CNES
Lochin
Mifdaoui
ISAE-SUPAERO
October 22, 2018

Priority Switching Scheduler
draft-finzi-priority-switching-scheduler-04

Abstract

We detail the implementation of a network rate scheduler based on both a packet-based implementation of the generalized processor sharing (GPS) and a strict priority policies. This credit based scheduler called Priority Switching Scheduler (PSS), inherits from the standard Strict Priority Scheduler (SP) but dynamically changes the priority of one or several queues. Usual scheduling architectures often combine rate schedulers with SP to implement DiffServ service classes. Furthermore, usual implementations of rate scheduler schemes (such as WRR, DRR, ...) do not allow to efficiently guarantee the capacity dedicated to both AF and DF DiffServ classes as they mostly provide soft bounds. This means excessive margin is used to ensure the capacity requested and this impacts the number of additional users that could be accepted in the network. PSS allows a more predictable output rate per traffic class and is a one fit all scheme allowing to enable both SP and rate scheduling policies within a single algorithm.

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	2
1.1. Context and Motivation	2
1.2. Definitions and Acronyms	3
1.3. Priority Switching Scheduler in a nutshell	3
2. Priority Switching Scheduler	5
2.1. Specification	5
2.2. Implementation with three traffic classes and one controlled queue	9
2.3. Implementation with n controlled queues	10
3. Usecase: benefit of using PSS in a Diffserv core network	12
3.1. Motivation	12
3.2. New service offered	14
4. Security Considerations	14
5. Acknowledgements	15
6. References	15
6.1. Normative References	15
6.2. Informative References	15
Authors' Addresses	16

1. Introduction

1.1. Context and Motivation

To enable DiffServ traffic classes and share the capacity offered by a link, many schedulers have been developed such as Strict Priority, Weighted Fair Queuing, Weighted Round Robin or Deficit Round Robin.

In the context of a core network router architecture aiming at managing various kind of traffic classes, scheduling architectures require to combine a Strict Priority (to handle real-time traffic) and a rate scheduler (WFQ, WRR, ... to handle non-real time traffic) as proposed in [RFC5865]. For all these solutions, the output rate of a given queue often depends on the amount of traffic managed by other queues. PSS aims at reducing the uncertainty of the output rate of selected queues, we call them in the following controlled queues. Additionally, compared to previous cited schemes, the scheduling scheme proposed is simpler to implement as PSS allows to both enable Strict Priority and Fair Queuing services; is more flexible following the wide possibilities offered by this setting; and does not require a virtual clock as for instance, WFQ.

1.2. Definitions and Acronyms

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].

- o AF: Assured Forwarding;
- o BLS: Burst Limiting Shaper;
- o DRR: Deficit Round Robin
- o DF: Default Forwarding;
- o EF: Expedited Forwarding;
- o PSS: Priority Switching Scheduler;
- o QoS: Quality-of-Service;
- o FQ: Fair Queuing
- o SP: Strict Priority
- o WFQ: Weighted Fair Queuing
- o WRR: Weighted Round Robin

1.3. Priority Switching Scheduler in a nutshell

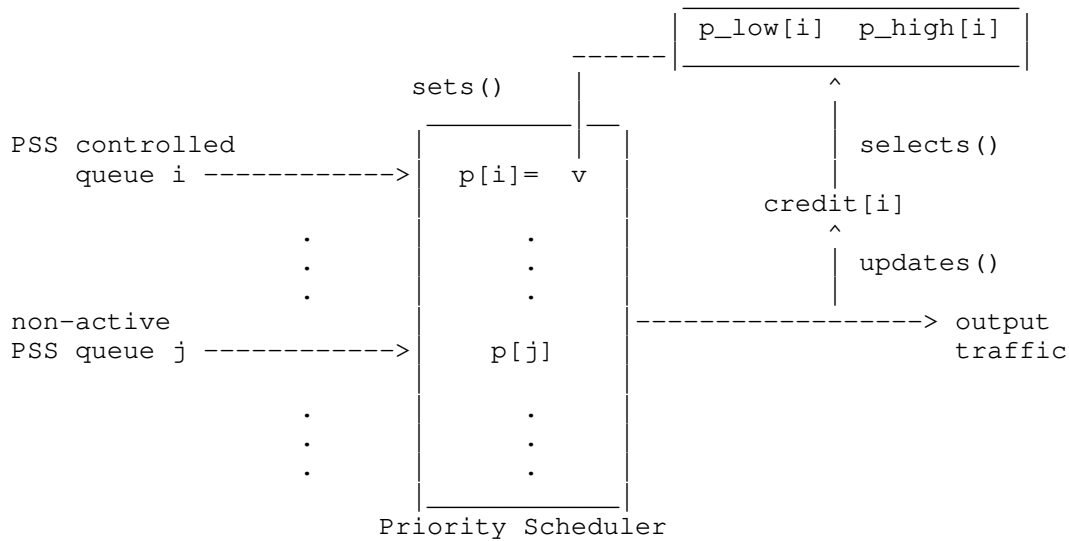


Figure 1: PSS in a nutshell

As illustrated in Figure 1, the principle of PSS is based on the use of credit counters (detailed in the following) to change the priority of one or several queues. Each controlled queue i is characterized by a current priority state $p[i]$, which can take two priority values: $\{p_high[i], p_low[i]\}$ where $p_high[i]$ the highest priority value and $p_low[i]$ the lowest. This idea follows a proposal made by the TSN Task group named Burst Limiting Shaper [BLS]. For each controlled queue i , each current priority $p[i]$ changes between $p_low[i]$ and $p_high[i]$ depending on the associated credit counter $credit[i]$. Then a Priority Scheduler is used for the dequeuing process, i.e., among the queues with available traffic, the first packet of the queue with the highest priority is dequeued.

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

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

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

2. Priority Switching Scheduler

2.1. Specification

For the sake of clarity and to ease the understanding of the PSS algorithm, we consider the case where only one queue is a controlled queue. This corresponds to three traffic classes EF, AF and DF where AF is the controlled queue as shown in Figure Figure 2.

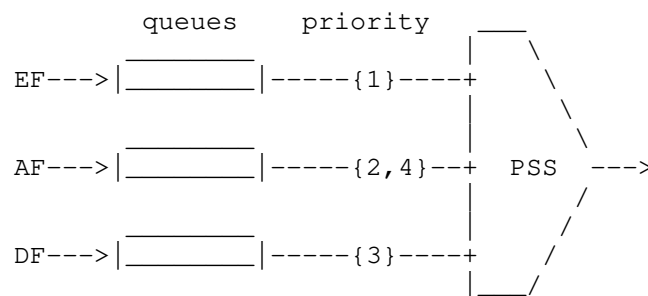


Figure 2: PSS with three traffic classes

As previously explained, the PSS algorithm defines for the controlled queue a low priority denoted p_{low} , and a high priority denoted p_{high} associated to a credit counter denoted $credit$, which manages the priority switching. Considering Figure 2, the priority $p[AF]$ of the controlled queue AF will be switched between two priorities where $p_{high}[AF] = 2$ and $p_{low}[AF] = 4$. The generalisation of PSS algorithm to n controlled queues is given in Section 2.3.

Then, each credit counter is defined by:

- o a minimum level: 0;
- o a maximum level: LM;
- o a resume level: LR such as $0 \leq LR < LR$;
- o a reserved capacity: BW;
- o an idle slope: $I_{idle} = C * BW$, where C is the link output capacity;

- o a sending slope: $I_{\text{send}} = C - I_{\text{idle}}$;

The available capacity (denoted C) is mostly impacted by the guaranteed capacity BW . Hence, BW should be set to the desired capacity plus a margin taking into account the additional packet due to non-preemption as explained below:

the value of LM can negatively impact on the guaranteed available capacity. The maximum level determines the size of the maximum sending windows, i.e., the maximum uninterrupted transmission time of the controlled queue packets before a priority switching. The impact of the non-preemption is as a function of the value of LM . The smaller the LM , the larger the impact of the non-preemption is. For example, if the number of packets varies between 4 and 5, the variation of the output traffic is around 25% (i.e. going from 4 to 5 corresponds to a 25% increase). If the number of packets sent varies between 50 and 51, the variation of the output traffic is around 2%.

The credit allows to keep track of the packet transmissions. However, keeping track the transmission raises an issue in two cases: when the credit is saturated at LM or at 0. In both cases, packets are transmitted without gained or consumed credit. Nevertheless, the resume level can be used to decrease the times when the credit is saturated at 0. If the resume level LR is 0, then as soon as the credit reaches 0, the priority is switched and the credit saturates at 0 due to the non-preemption of the current packet. On the contrary, if $LR > 0$, then during the transmission of the non-preempted packet, the credit keeps on decreasing before reaching 0 as illustrated in Figure 3.

Hence, the proposed value for LR is $L_{\text{max}} * BW$, with L_{max} the maximum packet size of the controlled queue. With this value, there is no credit saturation at 0 due to non-preemption.

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

The priority change depends on the credit counter as follows:

- o initially, the credit counter starts at 0;
- o the change of priority $p[i]$ of controlled queue i occurs in two cases:
 - * if $p[i]$ is currently set to $p_{\text{high}}[i]$ and $\text{credit}[i]$ reaches LM ;
 - * if $p[i]$ is currently set to $p_{\text{low}}[i]$ and $\text{credit}[i]$ reaches LR ;

- o when a packet of the controlled queue is transmitted, the credit increases (is consumed) with a rate I_{send} , else the credit decreases (is gained) with a rate I_{idle} ;
- o when the credit reaches LM, it remains at this level until the end of the transmission of the current packet (if any);
- o when the credit reaches LR and the transmission of the current packet is finished, in the absence of new packets to transmit in the controlled queue, it keeps decreasing at the rate I_{idle} until it reaches 0. Finally, the credit remains to 0 until the start of the transmission of a new packet.

Figure 3 and Figure 4 give two examples of credit and priority changes of a given queue. First, Figure 3 gives an example when the controlled queue sends its traffic continuously until the priority changes (this traffic is represented with @ below the x-axis of this figure). Then, the credit reaches LM and the last packet is transmitted although the priority have changed. Other traffic is thus sent (represented by o) uninterruptedly until the priority changes back. Figure 4 illustrates a more complex behaviour. First, this figure shows when a packet with a priority higher than $p_{\text{high}}[i]$ is available, this packet is sent before the traffic of queue i . Secondly, when no traffic with a priority lower than $p_{\text{low}}[i]$ is available, then traffic of queue i can be sent. This highlights the non-blocking nature of PSS and that $p[i] = p_{\text{high}}[i]$ (resp. $p[i] = p_{\text{low}}[i]$) does not necessarily mean that traffic of queue i is being sent (resp. not being sent).

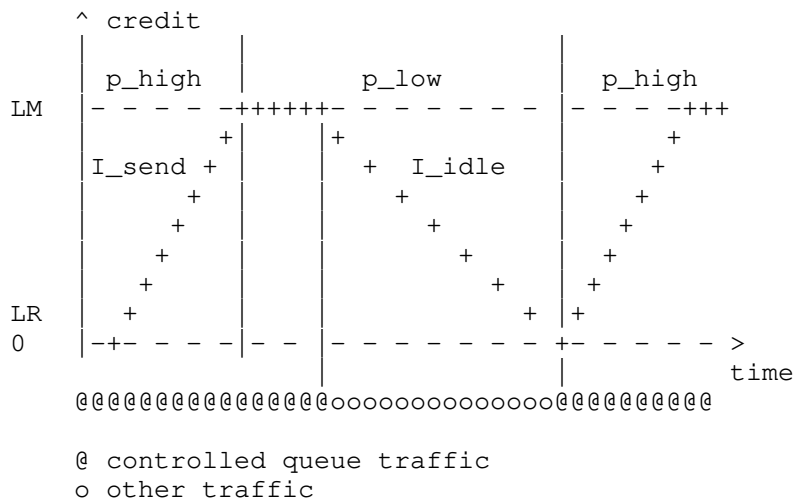


Figure 3: First example of queue credit evolution and priority switching.

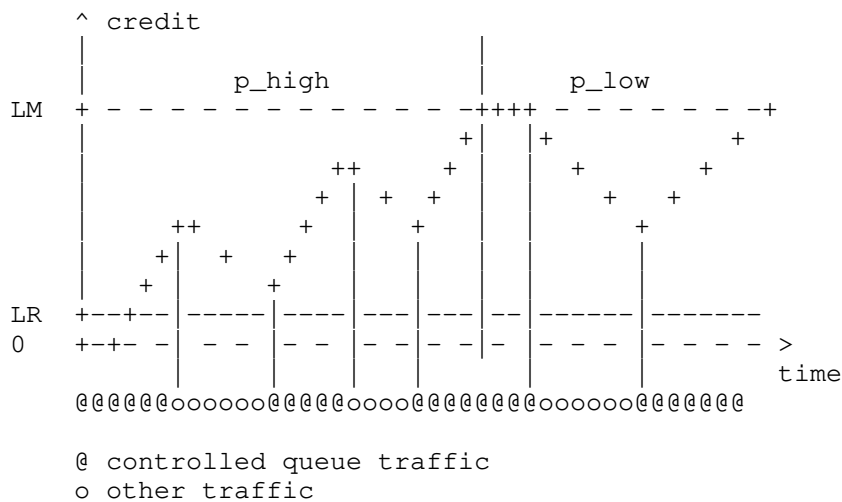


Figure 4: Second example of queue credit evolution and priority switching.

Finally, for the dequeuing process, a Priority Scheduler selects the appropriate packet using the current priority values. In other words, among the queues with packets enqueued, the first packet of the queue with the highest priority is dequeued (usual principle of SP).

2.2. Implementation with three traffic classes and one controlled queue

The new dequeuing algorithm is presented in the PSS Algorithm in Figure 5 and consists in a modification of the standard SP. The credit of the controlled queue and the dequeuing timer denoted `timerDQ` are initialized to zero. The initial priority is set to the highest value `p_high`. First, we compute the difference between the current time and the time stored in `timerDQ` (line #3). The duration `dtime` represents the time elapsed since the last credit update, during which no packet from the controlled queue was sent, we call this the idle time. Then, if `dtime > 0`, the credit is updated by removing the credit gained during the idle time that just occurred (lines #4 and #5). Next, `timerDQ` is set to the current time to keep track of the last time the credit was updated (line #6). If the credit reaches `LR`, the priority changes to its high value (lines #7 and #8). Then, with the updated priorities, SP algorithm performs as usual: each queue is checked for dequeuing, highest priority first (lines #12 and #13). When the queue selected is the controlled queue, the credit expected to be consumed is added to the credit variable (line #16). The time taken for the packet to be dequeued is added to the variable `timerDQ` (line #17) so the transmission time of the packet will not be taken into account in the idle time `dtime` (line #3). If the credit reaches `LM`, the priority changes to its low value (lines #18 and #19). Finally, the packet is dequeued (line #22).

```

Inputs: credit, timerDQ, C, LM, LR, BW, p_high, p_low
1  currentTime = getCurrentTime()
3  dtime = currentTime - timerDQ
4      if dtime > 0 then:
5          credit = max(credit - dtime * C * BW, 0)
6          timerDQ = currentTime
7          if credit < LR and p = p_low then:
8              p = p_high
9          end if
10     end if
11 end for
12 for each priority level, highest first do:
13     if length(queue[i]) > 0 then:
15         if queue[i] is the controlled queue then:
16             credit =
17                 min(LM, credit + size(head(queue[i])) * (1 - BW))
18             timerDQ = currentTime + size(head(queue[i]))/C
19             if credit >= LM and p = p_high then:
20                 p = p_low
21             end if
22         end if
23         dequeue(head(queue[i]))
24         break
25     end if
26 end for

```

Figure 5: PSS algorithm

PSS algorithm implements the following functions:

- o `getCurrentTime()` uses a timer to return the current time;
- o `length(q)` returns the length of the queue `q`;
- o `head(q)` returns the first packet of queue `q`;
- o `size(f)` returns the size of packet `f`;
- o `dequeue(f)` activates the dequeuing event of packet `f`.

2.3. Implementation with n controlled queues

The algorithm can be updated to support `n` controlled queues. In this context, the credits of each queue `i` must be stored in the table `creditList[i]`. Each controlled queue `i` has its own dequeuing timer stored in the table `timerDQList[i]`. Likewise for each controlled queue, `LM[i]`, `LR[i]`, `BW[i]`, `p_low[i]` and `p_high[i]` are respectively stored in `LMList[i]`, `LRList[i]`, `BWList[i]`, `p_lowList[i]` and

$p_highList[i]$. A controlled queue i is characterized by $p_lowList[i] > p_highList[i]$ (as priority 0 is the highest priority for SP). The current priority of a controlled queue is stored in $p[i]$. Each controlled queue must have distinct priorities.

As an example, Figure Figure 6 extends Figure 2 to n controlled queues.

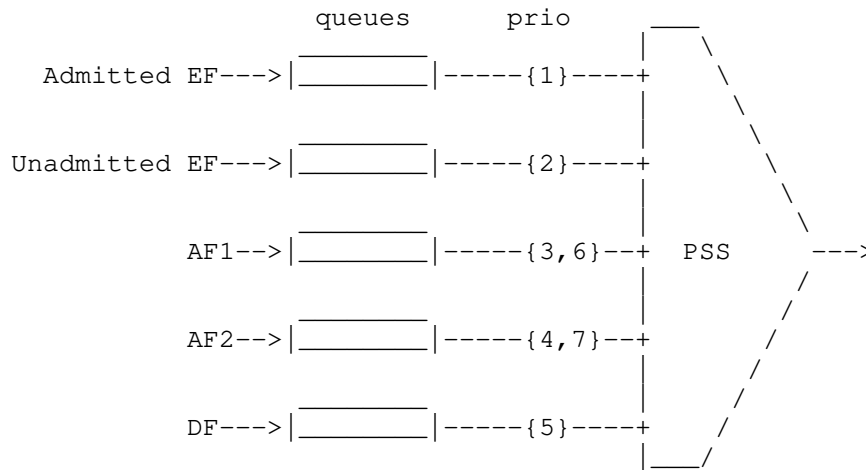


Figure 6: PSS with three traffic classes

```

Inputs: creditList[], timerDQList[], C, LMList[], LRList[],
        BWList[], p_highList[], p_lowList[]
1  for each queue i with p_highList[i] < p_lowList[i] do:
2      currentTime = getCurrentTime()
3      dtime = currentTime - timerDQList[i]
4      if dtime > 0 then:
5          creditList[i] =
            max(creditList[i] - dtime * C * BWList[i], 0)
6          timerDQList[i] = currentTime
7          if credit[i] < LRList[i] and p[i] = p_lowList[i] then:
8              p[i] = p_highList[i]
9          end if
10     end if
11 end for
12 for each priority level pl, highest first do:
13     if length(queue(pl)) > 0 then:
14         i = queue(pl)
15         if p_highList[i] < p_lowList[i] then:
16             creditList[i] =
                min(LMList[i],
                    creditList[i] + size(head(i)) * (1 - BWList[i]))
17             timerDQList[i] = currentTime + size(head(i))/C
18             if creditList[i] >= LMList[i]
                and p[i] = p_highList[i] then:
19                 p[i] = p_lowList[i]
20             end if
21         end if
22         dequeue(head(i))
23         break
24     end if
25 end for

```

Figure 7: PSS algorithm

The general PSS algorithm also implements the following function:

- o queue(pl) returns the queue i associated to priority pl.

3. Usecase: benefit of using PSS in a Diffserv core network

3.1. Motivation

The DiffServ architecture defined in [RFC4594] and [RFC2475] proposes a scalable mean to deliver IP quality of service (QoS) based on handling traffic aggregates. This architecture follows the philosophy that complexity should be delegated to the network edges while simple functionalities should be located in the core network.

Thus, core devices only perform differentiated aggregate treatments based on the marking set by edge devices.

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

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

To prevent a starvation and ensure to both DF and AF a minimum service rate, the router architecture proposed in [RFC5865] uses a rate scheduler between AF and DF classes to share the residual capacity left by the EF class. Nevertheless, one drawback of using a rate scheduler is the high impact of EF traffic on AF and DF. Indeed, the residual capacity shared by AF and DF classes is directly impacted by the EF traffic variation. As a consequence, the AF and DF class services are difficult to predict in terms of available capacity and latency. To overcome these limitations and make AF service more predictable, we propose here to use the newly defined Priority Switching Scheduler (PSS).

Figure 8 shows an example of the Data Plane Priority core network router presented in [RFC5865] modified with a PSS. The EF queues have the highest priorities to offer the best service to real-time traffic. The priority changes set the AF priorities either higher (3,4) or lower (6,7) than CS0 (5), leading to capacity sharing (CS0 refers to Class Selector codepoints 0 and is usually referred to DF as explained in [RFC7657]). Another example with only 3 queues is described in [Globecom17]. Thank to the increase predictability, for the same minimum guaranteed rate, the PSS reserves a lower percentage

of the capacity than a rate scheduler. This leaves more remaining capacity that can be guaranteed to other users.

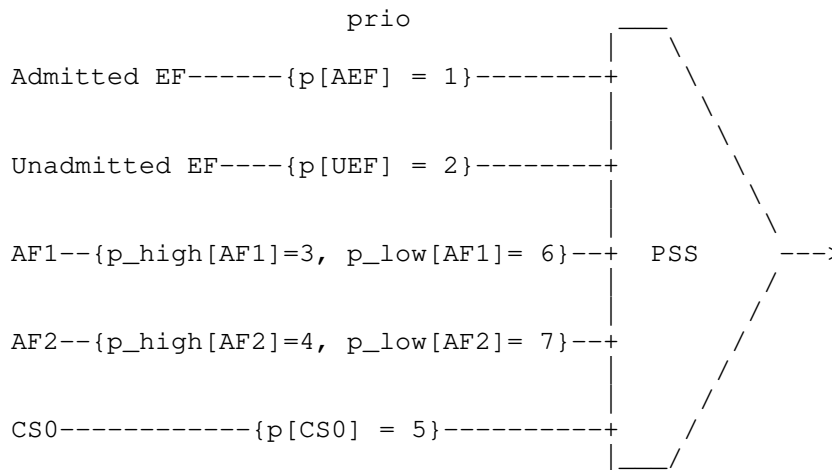


Figure 8: PSS applied to Data Plane Priority (we borrow the syntax from RCF5865)

3.2. New service offered

The new service we seek to obtain is:

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

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

4. Security Considerations

There are no specific security exposure with PSS that would extend those inherent in default FIFO queuing or in static priority scheduling systems. However, following the DiffServ usecase proposed in this memo and in particular the illustration of the integration of PSS as a possible implementation of the architecture proposed in

[RFC5865], most of the security considerations from [RFC5865] and more generally from the differentiated services architecture described in [RFC2475] still hold.

5. Acknowledgements

This document was the result of collaboration and discussion among a large number of people. In particular the authors wish to thank David Black, Ruediger Geib, Vincent Roca for reviewing this draft and Victor Perrier for the TUN/TAP implementation of PSS. At last but not least, a very special thanks to Fred Baker for his help.

6. References

6.1. 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>>.

6.2. Informative References

- [BLS] Gotz, F-J., "Traffic Shaper for Control Data Traffic (CDT)", IEEE 802 AVB Meeting , 2012.
- [Globecom17] Finzi, A., Lochin, E., Mifdaoui, A., and F. Frances, "Improving RFC5865 Core Network Scheduling with a Burst Limiting Shaper", Globecom , 2017, <<http://oatao.univ-toulouse.fr/18448/>>.
- [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>>.
- [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>>.
- [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>>.

[RFC7657] Black, D., Ed. and P. Jones, "Differentiated Services (Diffserv) and Real-Time Communication", RFC 7657, DOI 10.17487/RFC7657, November 2015, <<https://www.rfc-editor.org/info/rfc7657>>.

Authors' Addresses

Fred Baker
Santa Barbara, California 93117
USA

Email: FredBaker.IETF@gmail.com

Anais Finzi
TTTech Computertechnik AG
Schoenbrunner Strasse 7
Vienna 1040
Austria

Phone: 0043158534340
Email: anais.finzi@tttech.com

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

Email: fabrice.frances@isae-supaero.fr

Nicolas Kuhn
CNES
18 Avenue Edouard Belin
Toulouse 31400
France

Email: nicolas.kuhn@cnes.fr

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

Phone: 0033561338485
Email: emmanuel.lochin@isae-supero.fr

Ahlem Mifdaoui
ISAE-SUPAERO
10 Avenue Edouard Belin
Toulouse 31400
France

Email: ahlem.mifdaoui@isae-supero.fr

Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 14, 2018

L. Han, Ed.
G. Li
B. Tu
X. Tan
F. Li
R. Li
Huawei Technologies
J. Tantsura

K. Smith
Vodafone
October 11, 2017

IPv6 in-band signaling for the support of transport with QoS
draft-han-6man-in-band-signaling-for-transport-qos-00

Abstract

This document proposes a method to support the IP transport service that could guarantee a certain level of service quality in bandwidth and latency. The new transport service is fine-grained and could apply to individual or aggregated TCP/UDP flow(s).

Status of This Memo

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

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

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

This Internet-Draft will expire on April 14, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<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. IP and Transport Technologies	4
1.2. TCP Solution Analysis	4
1.2.1. TCP Overview and Evolution	4
1.2.2. TCP Solution Variants	5
1.2.3. Throughput Constraint	6
1.2.3.1. By Algorithm	6
1.2.3.2. By Fairness Principle	7
1.2.4. Latency Constraint	7
1.2.5. Summary of TCP Solution	7
1.3. Other Solution Analysis	8
1.4. New approach	8
1.4.1. IP Transport with quality of service	8
1.4.2. Design targets	9
1.4.3. Scope and assumption	9
2. Terminology	10
2.1. Definitions	10
3. Control plane	11
3.1. Sub-layer in IP for transport control	12
3.2. IP In-band signaling	13
3.3. Control mechanism	14
3.4. IPv6 Approach	15
3.4.1. Basic Control Scenarios for TCP	16
3.4.2. Details of In-band Signaling for TCP	17
3.5. Key Messages and Parameters in Control Protocol	20
3.5.1. Setup and Setup State Report messages	20
3.5.2. OAM	21
3.5.3. Forwarding State and Forwarding State Report messages	21
3.5.4. Flow Identifying Methods	21
3.5.5. Hop Number	23
3.5.6. Mapping Index, Size and Mapping Index List	23
3.5.7. QoS State and life of Time	23
3.5.8. Authentication	24
4. Data plane	24
4.1. Basic Capability	24
4.2. Forwarding State and Forwarding State Report	25
4.3. Flow Identification in Packet Forwarding	26
4.4. QoS Forwarding State Detection and Failure Handling	26

5. Other Issues	27
5.1. User and Application driven	27
5.2. Traffic Management in Host	28
5.3. Non-shortest-path	28
5.4. Heterogeneous Network	29
5.5. Proxy Control	29
6. Message Format	29
6.1. Setup Msg	29
6.2. Bandwidth Msg	31
6.3. Burst Msg	31
6.4. Latency Msg	31
6.5. Authentication Msg	32
6.6. OAM Msg	32
6.7. Forwarding State Msg	32
6.8. Setup State Report Msg	33
6.9. Forward State Report Msg	34
7. IANA Considerations	34
8. Security Considerations	35
9. Acknowledgements	35
10. References	36
10.1. Normative References	36
10.2. Informative References	36
Authors' Addresses	39

1. Introduction

Recently, more and more new applications for Internet are emerging. These applications have a common part that is 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, AR or VR applications may need a couple of hundred Mbps bandwidth (throughput) and a low single digit ms latency. Moreover, the difference of mean bit rate and peak bit rate is huge due to the compression algorithm [I-D.han-icrg-arvr-transport-problem].

Some future applications expect that network can provide a bounded latency service, such as tactile network [Tactile].

With the technology development in 5G and beyond, the wireless access network is also rising the demand for the Ultra-Reliable and Low-Latency Communications (URLLC), this also leads to the question if IP transport can provide such service in Evolved Packet Core (EPC) network. IP is becoming more and more important in EPC when the Multi-access Edge Computing (MEC) for 5G will require the cloud and data service moving closer to eNodeB.

Following sections will brief the current transport and QoS technologies, and analyze the limitations to support above new applications.

A new approach that could provide QoS for transport service will be proposed. The scope and criteria for the new technology will also be summarized.

1.1. IP and Transport Technologies

The traditional IP network can only provide the best-effort service. The transport layer (TCP/UDP) on top of IP are based on this fundamental architecture. The best-effort-only service has influenced the transport evolution for quite long time, and results in some widely accepted assumptions and solutions, such as:

1. The IP layer can only provide the basic P2P (point to point) or P2MP (point to multi-point) end-to-end connectivity in 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.

1.2. TCP Solution Analysis

As a most popular and widely used transport technology, TCP traffic is dominating in Internet from the born of Internet. It is important to analyze the TCP. This section will brief the TCP, its variation, and some key factors.

1.2.1. TCP Overview and Evolution

The major functionalities of TCP are flow control and congestion control.

The flow control is based on the sliding window algorithm. In each TCP segment, the receiver specifies in the receive window field the amount of additionally received data (in bytes) that it is willing to

buffer for the connection. The sending host can send only up to that amount of data before it must wait for an acknowledgment and window update from the receiving host.

The congestion control is algorithms to prevent the hosts and network device fall into congestion state while trying to achieve the maximum throughput. There are many algorithm variations developed so far.

All congestion control will use some congestion detection scheme to detect the congestion and adjust the rate of source to avoid the congestion.

No matter what congestion control algorithm is used, traditionally, all TCP solutions are pursuing three targets, high efficiency in bandwidth utilization, high fairness in bandwidth allocation, and fast convergence to the equilibrium state. [TCP_Targets]

Recently, with the growth of new TCP applications in data center, more and more solutions were proposed to solve bufferbloat, incast problems typically happened in data center. These solutions include DCTCP, PIE, CoDel, FQ-CoDel, etc. In addition to the three traditional targets mentioned above, these solutions have another target which is to minimize the latency.

1.2.2. TCP Solution Variants

There are many TCP variants and optimization solutions since TCP was introduced 40 years ago. We have collected major TCP variants including typical traditional solution and some new solutions proposed recently.

The traditional solutions:

These solutions are implemented on host only. They use different congestion detection and inference mechanism, either based on packet loss, RTT or both, to dynamically adjust the TCP window to do the congestion control, such as: TCP-reno [RFC2581], TCP-vegas [TCP-vegas], TCP-cubic [TCP-cubic], TCP-compound [I-D.sridharan-tcpm-ctcp], TIMELY [TIMELY], etc

The explicit rate solutions:

These solutions do not use the traditional black box mechanism executed at host to infer the TCP congestion status, instead, they rely on the rate calculation on routers to let host adjust accordingly. Both network devices and hosts must be changed. Typical solutions are: XCP [I-D.falk-xcp-spec], RCP [RCP]. Note, we put XCP and RCP as TCP here is referring to the scenario when XCP and RCP are used with TCP

The AQM solutions:

These solutions use AQM (Active Queue Management) techniques on routers to control the buffer size, thus control the congestion and minimize the latency indirectly. Both network devices and hosts must be changed. They include: DCTCP [I-D.ietf-tcpm-dctcp], PIE [I-D.ietf-aqm-pie], CoDel [I-D.ietf-aqm-codel], FQ-CoDel [I-D.ietf-aqm-fq-codel], etc.

The new concept solutions:

Unlike above categories, these solutions use completely new concepts and methods to either accurately calculate, or figure out the optimized rate and latency of TCP, such as: PERC [PERC], BBR [BBR], PCC [PCC], Fastpass [Fastpass], etc

1.2.3. Throughput Constraint

For the traditional TCP optimization solutions, the efficiency target is to obtain the high bandwidth utilization as much as possible to approach the link capacity. The link utilization is defined as the total throughput of all TCP flows on a network device to the network bandwidth for links.

For individual TCP flow, its actual throughput is not guaranteed at all. It depends on many factors, such as TCP algorithm used, the number of TCP flows sharing the same link, host CPU power, network device congestion status, delay in transmission, etc.

For traditional TCP, the real throughput for a flow is limited by three factors: The 1st one is the available maximum throughput at the physical layer, accounting for maximum theoretical bandwidth, network load, buffering configuration, maximum segment size, signal strength, etc; The another is related to congestion control algorithm; The 3rd is related to the TCP fairness principle. Below we will analyze the last two factors.

1.2.3.1. By Algorithm

No matter what algorithm is used, The TCP throughput is always related to some flow and network characteristics, such as the RTT (Round Trip Time) and PLR (packet loss ratio). For example, TCP-reno throughput is shown in the formula (3) in [Reno_throughput]; And TCP-cubic throughput is expressed in formula (21) in [Cubic_throughput].

This limit will prevent the link capacity to be utilized by all TCP flows. Each TCP flow may only get a few portion of the link bandwidth as the real throughput for application. Even there is one TCP flow in a link, the throughput for the TCP could be way below the link capacity for a network which RTT and PLR are high.

1.2.3.2. By Fairness Principle

TCP fairness is a de facto principle for all TCP solutions. By this rule, each router will process all TCP flows equally and fairly to allocate the required resource to all TCP flows. Different Fair Queuing algorithms were used, such as Packet based Round Robin, Core-Stateless Fair Queuing(CSFQ), WFQ, etc. The targets of all algorithms are to reach the so called max-min fairness [Fairness] of TCP in terms of bandwidth.

TCP fairness played an important and critical role in saving internet from collapse caused by congestions since TCP was introduced.

The analysis [RCP] on page 35 has given the formula of the fair share rate at bottleneck routers, the rate or throughput is capped for applications which required bandwidth are not satisfied under the rule of fairness.

1.2.4. Latency Constraint

TCP fairness will not process some TCP flows differently with others, or there is no TCP micro-flow handling.

As described above, for the traditional solutions and explicit rate solution, the latency is not considered as a target, thus no latency guarantee at all.

For AQM solutions and some new concept solutions which try to control the buffer bloat or flow latency, it can only provide the statistic bounded latency for all TCP flows. The latency is related to the queue size and other factors. And the real latency for specific flow(s) is not deterministic. It could be very small or pretty large due to the long tail effect if the flow is blocked by other slower TCP flows.

1.2.5. Summary of TCP Solution

The bandwidth and latency can hardly be satisfied simultaneously without micro flow handling and management. While trying to get higher bandwidth, it may lead to more queued packet in router and result in longer latency. While approaching shorter latency, it may cause the queue under run, and lead to the lower bandwidth.

As a summary, to support some special TCP applications that are very sensitive to bandwidth and/or latency, we need to handle those TCP flows differently with others, and the TCP fairness must be relaxed for these scenarios.

It must be noted that the fairness based transport service could satisfy most of the applications, and it is the most efficient and economical way for hardware implementation and the network bandwidth efficiency.

When providing some TCP flows with differentiated service, the traditional transport service must be able to coexist with the new service. The resource partitioning between different service is a operation and management job for service provider.

1.3. Other Solution Analysis

DiffServ

DiffServ [DiffServ] or Differentiated services is a network architecture that specifies a simple, scalable and coarse-grained mechanism for classifying and managing network traffic and providing QoS on modern IP networks. DiffServ is designed to support the QoS of aggregated traffic and normally is deployed in Service Provider networks. End user application cannot directly use DiffServ.

IntServ

IntServ [IntServ] or integrated services specifies more fine-grained QoS, which is often contrasted with DiffServ's coarse-grained control system. IntServ definitely can support the applications requiring special QoS guarantee if it is deployed in a network, supported by Host OS and integrated with application. However, IntServ works on a small-scale only. When you scale up the network, it is difficult to keep track of all of the reservations and session states. Thus, IntServ is not scalable. Another problem of IntServ is it is not application driven, tedious provisioning cross different network must be done earlier. The provisioning is slow and hard to maintain.

MPLS-TE

MPLS-TE can provide aggregated QoS or fine-grained QoS service for different class of traffic. Similar to DiffServ, MPLS-TE is majorly used for service providers network. It requires extra protocol sets like LDP, MPLS-TE, etc to operate. It is not practical to extend MPLS-TE to end user's desktop.

1.4. New approach

1.4.1. IP Transport with quality of service

Semiconductor chip technology has advanced a lot for last decades, the widely used network process can not only forward the packet in line speed, but also support fast packet processing for other

features, such as QoS for DiffServ/MPLS, Access Control List (ACL), fire wall, Deep Packet Inspection (DIP), etc. To treat some TCP/IP flows differently with others and give them specified resource are feasible now by using network processor.

Network processor is also able to do the general process to handle the simple control message for traffic management, such as signaling for hardware programming, congestion state report, OAM, etc.

This document proposes a mechanism to provide the capability of IP network to support the transport layer with quality of service. The solution is based on the QoS implemented in network processor. the proposal of the document is composed of two parts:

1. Control plane, it explains a transport control sub-layer for IP, the details of control mechanism.
2. Data plane, the realization of QoS in data forwarding, QoS and error handling.

1.4.2. Design targets

The new transport service is expected to satisfy following criteria:

1. End user or application can directly use and control the new service
2. The new service can coexist with the current transport service and is backward compatible.
3. The service provider can manage the new service.
4. Performance and scalability targets of new service are practical for vendors to achieve.
5. The new service is transport agnostic. Both TCP, UDP and other transport protocols on top of IP can use it

1.4.3. Scope and assumption

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 transport with QoS is aimed to be supplementary to the regular transport service. At the current situation, It is targeted for the applications that are bandwidth and/or latency

sensitive. It is not intended to replace the TCP 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 to extend the mechanism for inter-domain scenarios. Thus, the security and other inter-domain requirements are not critical. The basic security is good enough, 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 the application that really needs it, the number of supported applications/users are under controlled and cannot be unlimited. So, the scalability requirement for the new service is limited.
4. The new service must coexist with the regular transport service in the same hardware, and backward compatible. Also, a transport flow can switch without the service interruption between the regular transport support and new service.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2.1. Definitions

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 the sending of 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, a IP flow is identified by the source, destination IP address, the protocol number, the source and destination port number.

IP path

A 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 decided by another management entity, such as a central controller, or Path Computation Element (PCE) Communication Protocol (PCEP), etc

QoS channel

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

Cir

Committed Information Rate, this is the guaranteed bandwidth

Pir

Peak Information Rate. this is the up limit bandwidth. Whether a flow can reach the PIR depends on the implementation. To use resource more efficiently, the system normally does not guarantee the PIR, but allow the sharing of resource between flows.

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. Control plane

3.1. Sub-layer in IP for transport control

In order to provide some new features for the upper 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 the 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.

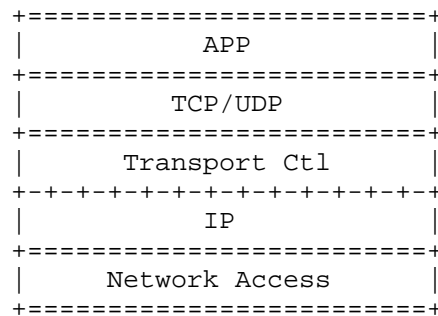


Figure 1: The new stack with a sub-layer in Layer 3

The new sub-layer is always bound with IP layer and can provide a 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.

IPv6 can realize the sub-layer easily by the IPv6 extension header [RFC8200].

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.

The document will focus on the solution for IPv6 by using different IPv6 extension header.

The control plane of the propose comprises of IP in-band signaling, and the detailed control mechanisms.

3.2. IP In-band signaling

There is no definition for IP in-band signaling. From the point of view of similarity to traditional telecommunication technology, the In-band signaling for IP is that the IP control messages are sharing some common header information as the data packet.

In this document, we introduce three types of "in-band signaling" for different signaling granularity:

Flow level In-band Signaling

The control message and data packet share the same flow identification. 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. For the flow level in-band signaling, the signaling is for the individual IP flow, and there is no aggregation at all.

Address level In-band Signaling

The control message and data packet 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

The control message and data packet 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 is the situation that the signaling is for the aggregated TCP or UDP flows that started and terminated at the same IP addresses.

Using In-band signaling, the control message can be embedded into any data packet, 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.

Note, the requirement of IP in-band signaling was proposed before by John Harper [I-D.harper-inband-signalling-requirements]. And the in-band QoS signaling for IPv6 was simply discussed in [I-D.roberts-inband-qos-ipv6]. Unfortunately, both works did not continue.

This document not only gives detailed solution for in-band signaling, but also try to address issues raised for the previous proposal, such as security, scalability and performance. Finally, experiments with proprietary hardware and chips are given in a presentation.

3.3. Control mechanism

The in-band signaling must be cooperated with a control method to achieve the QoS control. There are two categories of control, one is the closed-loop control and another is the open-loop control.

1. Closed-loop control is that the in-band signaling is sent in one direction and the feedback will return in the reverse direction. For example, the closed-loop control can be achieved by inserting the signaling information into a data packet sent in one direction, and the feedback information is carried in the data packet in reverse direction. The transport service with bi-direction data flow can use this mechanism, such as TCP and point-to-point UDP. In closed-loop control, a signaling message in one direction is processed at each router on the path. When the signaling message reaches the destination, the signaling message is processed by the protocol stack in the host, and the report information is generated. The report information is then

embedded into the flow data packet in the reverse direction and return to the host of the signaling source.

2. Open-loop control is that the in-band signaling is sent periodically in one direction without any feedback. The transport service with uni-direction data flow can use this mechanism, such as multicast by UDP. The transport service with bi-directional data flow can also use this mechanism when the simplicity of the control is wanted, i.e. no control feedback needed.

For both closed-loop and open-loop control, the signaling message for one direction is for the QoS programming for the direction. For example, the TCP-SYN or TCP data packet from client to server can carry the in-band signaling message to program the QoS for the direction of client to service. TCP-SYNACK or TCP data packet from server to client can carry the in-band signaling message to program the QoS for the server to client direction

Due to the nature that symmetric IP path between any source and destination cannot be guaranteed, in closed-loop control, the feedback information may take the different path as the in-band signaling path. The in-band signaling must not depend on the feedback information to accomplish the signaling work, such as the programming of hardware. This is one of the difference between in-band signaling and RSVP protocol.

For this document, we will only discuss the detailed mechanism for closed-loop control for TCP.

3.4. IPv6 Approach

The IPv6 In-band signaling could be realized by using the IPv6 extension header.

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]. We call this nodes as HbH-EH-aware nodes in document below. It 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.

3.4.1. Basic Control Scenarios for TCP

The finest grained QoS for TCP is flow level, this document will only focus on the solution of the flow level in-band signaling and its data plane. Other two types, address level and transport level QoS for TCP are briefly discussed in section 5.3.

The feature of TCP with flow level QoS comprises following control scenarios:

1. Setup: The setup is combined with the TCP 3-hand shaking, or any two directional TCP packets. When used with TCP 3-hand shaking, the 1st signaling embedded into HbH-EH is sent with TCP-SYN. It will be processed at HbH-EH-aware nodes on the path from source to destination. The signaling message includes the QoS requirements, such as max/min bandwidth, burst size, the latency, and the setup state. The setup state message is updated at HbH-EH-aware nodes to include the QoS programming and provisioning result and the necessary hardware reference information for IP forwarding with QoS. The 2nd signaling message is the TCP-SYNACK from server side, it includes the setup report message encoded as the Dst-EH. The setup report message is from the 1st TCP-SYN which represents the setup results on all HbH-EH-aware nodes on the path. The setup can even be started after TCP is established whenever the QoS service is required.
2. Dynamic control: this scenario is for the situation that previous QoS programming must be refreshed, modified or re-programmed. Normally, the signaling message can be embedded into HbH-EH for any TCP data packet or TCP-ACK packet. There are couple cases that the dynamic control is needed.

HW state refreshing

The HW state for QoS programming is data driven (see Section 4.1 for details). Its state will be refreshed if there is a data packet received. If there is no data received for a pre-configured time, the HW programming will be erased and the resource will be released.

HW programming modification

The HW QoS parameters can be modified if a new in-band signaling message is received and the embedded parameters are different with the old one that was used to program the HW. Section 3.4.2 will explain more about this scenario.

HW programming repairing

The IP path may be changed due to rerouting, link or node failures. This may result in the HW QoS programming failure. To repair any QoS programming failure, the new in-band signaling message can be embedded into any data packet and sent to the destination. All hops on the new path will be reprogrammed with the QoS parameters. Section 4.4 has more detailed discussion.

3. Congestion Control: For TCP protocol, if IP layer can provide a certain level of quality service guarantee, the congestion control algorithm will be impacted a lot. As for what is the new congestion control, it depends on the quality service implementation in hardware and the behavior of the application. This is simply discussed in section 5.2.

3.4.2. Details of In-band Signaling for TCP

This document introduces following type of message for in-band signaling and associated data forwarding, the detailed format of messages is expressed in Section 6,

- 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 Mapping Index 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 5.1

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 impcted 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-icrg-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 4.4.

3.5. Key Messages and Parameters in Control Protocol

The detailed message format is described in the section 6, the detailed explanation of key messages and parameters are below:

3.5.1. Setup and Setup State Report messages

Setup is the message used for following purpose:

- o Setup the QoS channel for a TCP when the TCP session is establishing.
- o Dynamic Control of the QoS channel for a established TCP session. See section 3.4.1

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 appropriate TCP packet and will be processed at each HbH-EH-aware node. For the simplicity, performance and scalability purpose, we can configure some hop to do the processing and some hops do not. For different QoS requirement and scenarios, different criteria can be used for the configuration of the hop to be HbH-EH-aware node, below are some factor to consider:

- 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. The throttle router is the device that a interested TCP session cannot get the enough bandwidth to support its application. The regular throttle routers include the BRAS (broadband remote access server) in broadband access network, the PGW (PDN Gateway) in LTE network, the TOR (Top of Rack) in data center. In more general case, any routers which aggregate traffic may become as a throttle router. Moreover, the direction of congestion must be considered. Normally, the congestion happens on the direction that more than one flows from multiple ingress links are aggregated and sent to one egress link. For other devices that the interested TCP session can get the enough bandwidth do not need to process the hop-by-hop EH.
- 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-cast 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.

3.5.2. OAM

OAM is a special in-band signaling message used for detection and diagnosis. It can be used before and after a QoS channel is established. Before a QoS channel is established, OAM message can be added as a HbH-EH to any IPv6 packet and used to detect:

- o IP path properties: Total hop number that is HbH-EH-aware node; The IP address of each HbH-EH-aware node.
- o Static properties at each HbH-EH-aware node: Protocol version; Supported Flow identifying methods; Mapping index size; Supported configuration range of bandwidth, latency, forwarding QoS state time.
- o Financial properties at each HbH-EH-aware node: Unit price for bandwidth; Unit price for service duration; Price for different latency.

After a QoS channel is established, OAM message can also be added as a HbH-EH to any IPv6 packet and used to detect and diagnose failures:

- o IP path dynamic properties: Total end to end latency
- o Dynamic properties at each HbH-EH-aware node: Queue size; Remained bandwidth; Dropped packet number by different reasons.
- o The detailed QoS forwarding failure reason.

3.5.3. Forwarding State and Forwarding State Report messages

Forwarding State and Forwarding State Report messages are used for data plane, See section 4.2.

3.5.4. Flow Identifying Methods

This is a parameter to program the HW for the flow identifying method. It is used for the QoS granularity definition and flow identification for QoS process. The QoS is enforced for a group of flows or a dedicated flow that can be identified by the same flow

identification. The QoS granularity is determined by the flow identification method during the setup and packet forwarding process. There are three levels of QoS granularities: Flow level, Address level and transport level. Each level of QoS granularity is realized by corresponding in-band signaling. The document focus on the flow level in-band signaling, other two level in-band signaling are discussed in the section 5.3.

There are two ways for the flow identifying method. One is by the tuples in IP header, another is by a local significant number (see mapping index) generated and maintained in a router. When "Mapping Index Size" (Mis) is zero, it means the "Flow identification method" (FI) is used for both control plane and data plane. When "Mis" is not zero, it means "FI" is only used in signaling, and the data plane will only use the "Mapping Index".

There are four types for "Flow identification method":

1. Individual Flow: Non-IPSec case: flow is identified by source and destination address, source and destination port number, and protocol number; IPSec case: flow is identified by source and destination address, flow label. For both case, FI = 0; the associated QoS is flow level, and QoS is guaranteed for a dedicated IP flow.
2. TCP flows: flow is identified by source and destination address, and TCP protocol number. The associated QoS is transport level, and QoS is guaranteed for TCP flows that have the same source and destination address. For this case, FI=1.
3. UDP flows: flow is identified by source and destination address, and UDP protocol number. The associated QoS is transport level, and QoS is guaranteed for UDP flows that have the same source and destination address. For this case, FI=2.
4. All flows: flow is identified by source and destination address. The associated QoS is address level, and QoS is guaranteed for all IP flows that have the same source and destination address. For this case, FI=3

The use of local generated number to identify flow is to speed up the flow lookup and QoS process for data plane. The number could be the MPLS label or a local tag for a MPLS capable router. The difference between this method and the MPLS switch is that there is no MPLS LDP protocol running and the IP packet does not need to be encapsulated as MPLS packet at the source host. When the MPLS label is used, the "Mapping Index Size" is 20 bits.

3.5.5. Hop Number

This is a parameter for the total number of hop that is HbH-EH-aware node on the path. it is the field "Hop_num" in Setup message. It is used to locate the bit position for "Setup State" and the "Mapping Index" in "Mapping Index List". The value of "Hop_num" must be decremented at each hop. And at the receive 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 by the OAM message.

3.5.6. Mapping Index, Size and Mapping Index List

Mapping Index is the local significant number generated and maintained in a router, and The "Mapping Index List" is just a list of "Mapping Index" for all hops that are HbH-EH-aware nodes on the IP path.

Mapping Index Size is the size for each mapping index in the Mapping Index List. The source host must know Mapping Index Size, and setup the initial value in the Setup message. The exact Mapping Index Size can be detected by the OAM message.

When a router receives a HbH-EH, it may generate a mapping index for the flow(s) that is defined by the Flow Identifying Method in "FL". Then the router must attach the mapping index value to the end of the Mapping Index List. After the packet reaches the destination host, the Mapping Index List will be that the 1st router's mapping index as the list header, and the last router's mapping index as the list tail.

3.5.7. QoS State and life of Time

After the chip 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.

3.5.8. 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, Mis, 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

4. Data plane

To support the QoS feature, there are couple of important requirements and schemes for implementations. These include the basic capability for the hardware, the scheme for the data forwarding, QoS processing, state report, etc.

Section 4.1 will talk about the basic capability for data plane, and section 4.2 will discuss the messages used for data plane after the QoS channel is established.

4.1. Basic Capability

The document only proposes the protocol used for control, and it is independent of the implementation of the system. However, 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). 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 QoS 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.

4.2. Forwarding State and Forwarding State Report

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 (Mis = 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 mapping index as identification for QoS process (Mis != 0).
 - * The hardware was programmed to use the tuples in IP header as identification for QoS process (Mis = 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 mapping index 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 to the host in the reverse direction.

4.3. 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 "Mis" is zero or the Mapping index 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. See section 4.3
2. Hardware was programmed to use mapping index to identify flows. This is indicated by that the "Mis" is not zero. When a packet is received, the mapping index 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.

4.4. QoS Forwarding State Detection and Failure Handling

QoS forwarding may be failed 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 5.1

5. Other Issues

Above document only covers the details for the QoS support of individual TCP session by using the flow level in-band signaling. Due to the extensive scope of in-band signaling, there are many other associated issues for IP transport control. Below lists some of them, and we only brief the solution but do not go to details.

The details of each topic can be expressed in other drafts.

5.1. 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 in-band signaling is 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.

5.2. Traffic Management in Host

In order to accommodate in-band signaling and the QoS transport service, the OS on a host must 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. Another is congestion control for TCP flows:

1. The current traffic management in a host manages traffic from different TCP/UDP session going out host link(s), in the way similar to routers to send traffic out. 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 will be much simpler. 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 the 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.

5.3. Non-shortest-path

The above method for the transport service with QoS is for the normal IP flows passing along the shortest path determined by the IGP or BGP. However, the IP shortest path may not be the best path in terms of the QoS. For example, the original IP path may not have enough bandwidth for a transport QoS service. The latency of the IP path is not the minimum in the network. There are two problems involved. One is how to find the best path for a QoS criteria, bandwidth or

latency. Another is how to setup the transport QoS for a non-shortest-path.

The 1st problem is out of scope of this document and many technologies have been discovered or are in research.

The 2nd problem can be solved by combining the segment routing and in-band signaling. The use of the HbH-EH and Dst-EH is independent of the type of IP path, thus can be used with segment routing for any path determined by source. Note, the HbH-EH-aware nodes may not be different as the explicit IPv6 address in the segment routing header.

5.4. Heterogeneous Network

When IP network is crossing a non-IP network, such as MPLS or Ethernet network, the in-band signaling needs to be interworking with that network. The behavior, protocol and rules in the interworking with non-IP network is not the problem this document will address. More study and research need to be done, and new draft should be written to solve the problem.

5.5. Proxy Control

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

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

6. Message Format

6.1. Setup 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 0 0 | ver | FI | R | Mis | P | Time | Hop_num | u | Total_latency |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     State for each hop index                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     Mapping index list for hops                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

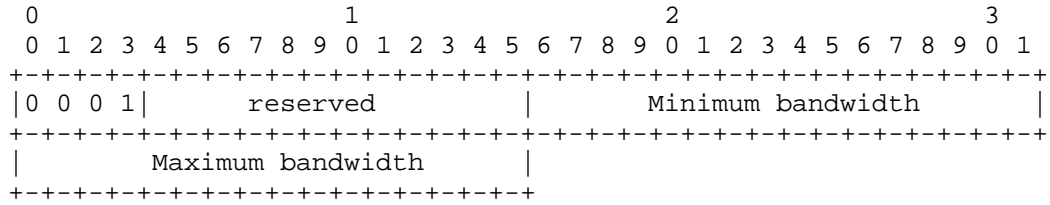
Type = 0, Setup state;
Version: The version of the protocol for the QoS
FI: Flow identification method,
0: 5 tuples; 1: src,dst,TCP; 2: src,dst,UDP; 3: src,dst
R: If the destination host report the received Setup state to
the src address by Destination EH. 0: dont report; 1: report
Mis: Mapping index size; 0: 0bits, 1: 16bits, 2: 20bits, 3: 32bits
P: Programming the HW for QoS; 0: program HW for the QoS from
src to dst; 1: De-program HW for the QoS from src to dst
Time: The life time of QoS forwarding state in second.
Hop_num: The total hop number on the path set by host. It must be
decremented at each hop after the processing.
u: the unit of latency, 0: ms; 1: us
Total_latency : Latency accumulated from each hop, each hop will
add the latency in the device to this value.
Setup state for each hop index: each bit is the setup state on
each hop on the path, 0: failed; 1: success. The 1st hop is at the
most significant bit.
Mapping index list for hops: the mapping index list for all hops
on the path, each index bit size is defined in Mis. The 1st
mapping index is at the top of the stack. Each hop add its mapping
index at the correct position indexed by the current hop number
for the router.

Figure 2: The Setup message

The Setup message is embedded into the hop-by-hop EH to setup the QoS in the device on the IP forwarding path. At each hop, if the router is configured to process the header and to enforce the QoS, it must retrieve the hardware required information from the header, and then update some fields in the header.

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 mapping index list is empty before the 1st hop receives the in-band signaling. Each hop then fill up the associated mapping index into the correct place determined by the index of the hop.

6.2. Bandwidth Msg



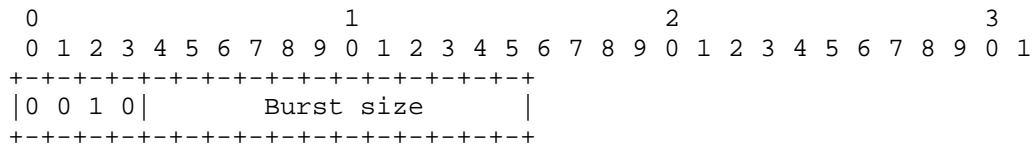
Type = 1,

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

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

Figure 3: The Bandwidth message

6.3. Burst Msg

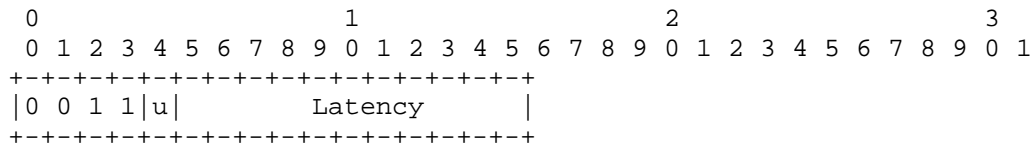


Type = 2,

Burst size : The burst size, unit M bytes

Figure 4: The burst message

6.4. Latency Msg



Type = 3,

u: the unit of the latency

0: ms; 1: us

Latency: Expected maximum latency for each hop

Figure 5: The Latency message

6.5. Authentication 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 1 0 0| MAC_ALG | res | MAC data (variable length) |
+-----+-----+-----+-----+-----+-----+-----+-----+ . . . +

```

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 6: The Authentication message

6.6. OAM Msg

```

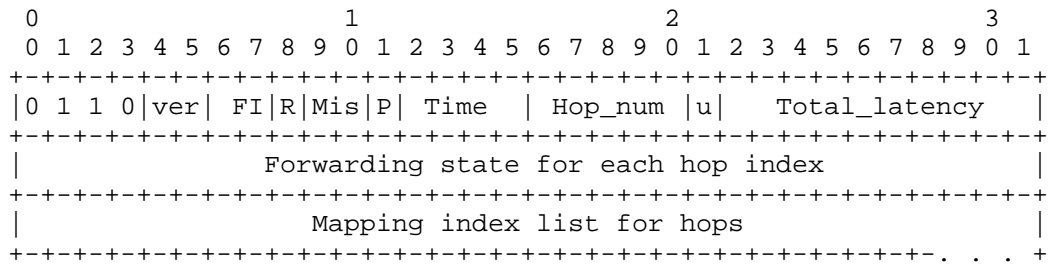
      0                               1                               2                               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|0 1 0 1| OAM_t | OAM_len | OAM data (variable length) |
+-----+-----+-----+-----+-----+-----+-----+-----+ . . . +

```

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 7: The OAM message

6.7. Forwarding State Msg



Type = 6, Forwarding state;

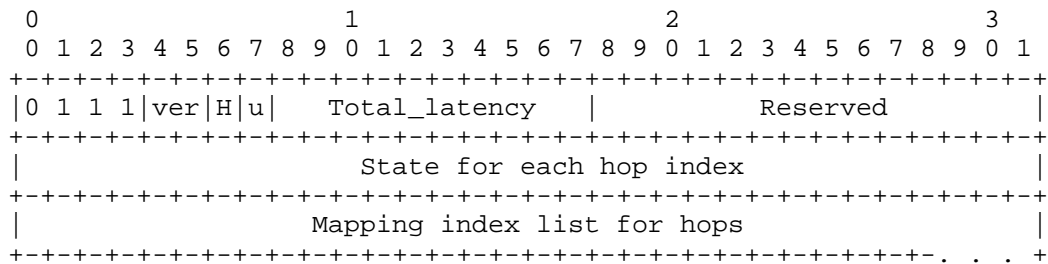
All parameter definitions and process in the 1st row are same in the setup message.

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.

Mapping index list for hops: the mapping index list for all hops on the path, each index bit size is defined in Mis. The list is from the setup report message.

Figure 8: The Forwarding State message

6.8. Setup State Report Msg



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:

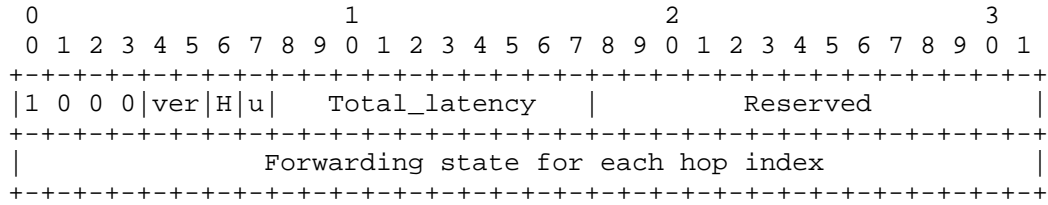
u, Total_latency;

State for each hop index

Mapping index list for hops.

Figure 9: The Setup State Report message

6.9. Forward State Report Msg



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 10: The Fwd State Report message

7. 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 6 in this doc

Figure 11: 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 message	Section 6.1
1	BANDWIDTH	Bandwidth message	Section 6.2
2	BURST	Burst message	Section 6.3
3	LATENCY	Latency message	Section 6.4
4	AUTH	Authentication message	Section 6.5
5	OAM	OAM message	Section 6.6
6	FWD STATE	Forward state	Section 6.7
7	SETUP REPORT	Setup state report	Section 6.8
8	FWD REPORT	Forwarding state report	Section 6.9

Figure 12: The In-band Signaling Sub Type

8. Security Considerations

There is no security issue introduced by this document

9. Acknowledgements

We like to thank Huawei's Nanjing research team leaded by Feng Li to provide the Product on Concept (POC) development and test, the team member includes Fengxin Sun, Xingwang Zhou, Weiguang Wang. We also like to thank other people involved in the discussion of solution: Tao Ma from Future Network Streategy dept.

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>>.
- [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

- [BBR] Neal Cardwell, et al, Google, "BBR Congestion Control", 2016, <<https://www.ietf.org/proceedings/97/slides/slides-97-iccr-g-bbr-congestion-control-02.pdf>>.
- [Cubic_throughput] Wei Bao, et al. The University of British Columbia, Vancouver, Canada, IEEE Globecom 2010 proceedings, "A Model for Steady State Throughput of TCP CUBIC", 2010, <https://www.researchgate.net/publication/224211021_A_Model_for_Steady_State_Throughput_of_TCP_CUBIC>.
- [DiffServ] wiki, "Differentiated services", 2016, <https://en.wikipedia.org/wiki/Differentiated_services>.
- [Fairness] Jain, R., et al. DEC Research Report TR-301, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems", 1984, <<http://www1.cse.wustl.edu/~jain/papers/ftp/fairness.pdf>>.
- [Fastpass] Jonathan Perry, et al, MIT, "Fastpass: A Centralized ?Zero-Queue? Datacenter Network", 2014, <<http://fastpass.mit.edu/Fastpass-SIGCOMM14-Perry.pdf>>.

- [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.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-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.
- [IntServ] wiki, "Integrated services", 2016,
<https://en.wikipedia.org/wiki/Integrated_services>.
- [IPv6_Parameters]
IANA, "Internet Protocol Version 6 (IPv6) Parameters",
2015, <[https://www.iana.org/assignments/ipv6-parameters/
ipv6-parameters.xhtml#ipv6-parameters-2](https://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xhtml#ipv6-parameters-2)>.
- [PCC] Mo Dong, et al, University of Illinois at Urbana-
Champaign, Hebrew University of Jerusalem, "PCC: Re-
architecting Congestion Control for Consistent High
Performance", 2014, <<https://arxiv.org/abs/1409.7092>>.
- [PERC] Lavanya Jose, et al, Stanford University, MIT, Microsoft,
"High Speed Networks Need Proactive Congestion Control",
2016, <[http://web.stanford.edu/~lavanyaj/papers/
perc-hotnets15.pdf](http://web.stanford.edu/~lavanyaj/papers/perc-hotnets15.pdf)>.
- [RCP] Nandita Dukkipati, Ph.D. Thesis, Department of Electrical
Engineering, Stanford University, "Rate Control Protocol
(RCP): Congestion control to make flows complete quickly",
2007,
<<http://yuba.stanford.edu/~nanditad/thesis-NanditaD.pdf>>.
- [Reno_throughput]
Matthew Mathis, et al, Pittsburgh Supercomputing Center,
"The Macroscopic Behavior of the TCP Congestion Avoidance
Algorithm", 1997,
<[https://cseweb.ucsd.edu/classes/wi01/cse222/papers/
mathis-tcpmodel-ccr97.pdf](https://cseweb.ucsd.edu/classes/wi01/cse222/papers/mathis-tcpmodel-ccr97.pdf)>.
- [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-cubic]
Ha, S., Rhee, I., and L. Xu, "CUBIC: A New TCP-Friendly
High-Speed TCP Variant", 2008.

[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.

[TIMELY]

Radhika Mittal, et al. Google, Inc., "TIMELY: RTT-based Congestion Control for the Datacenter", 2010, <<http://conferences.sigcomm.org/sigcomm/2015/pdf/papers/p537.pdf>>.

Authors' Addresses

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

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

Guoping Li
Huawei Technologies
Beijing
China

Email: liguoping@huawei.com

Boyan Tu
Huawei Technologies
Beijing
China

Email: tuboyan@huawei.com

Xuefei Tan
Huawei Technologies
Beijing
China

Email: tanxuefei@huawei.com

Frank Li
Huawei Technologies
Nanjing
China

Email: frank.lifeng@huawei.com

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

Email: renwei.li@huawei.com

Jeff Tantsura

Email: jefftant.ietf@gmail.com

Kevin Smith
Vodafone
UK

Email: Kevin.Smith@vodafone.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 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.

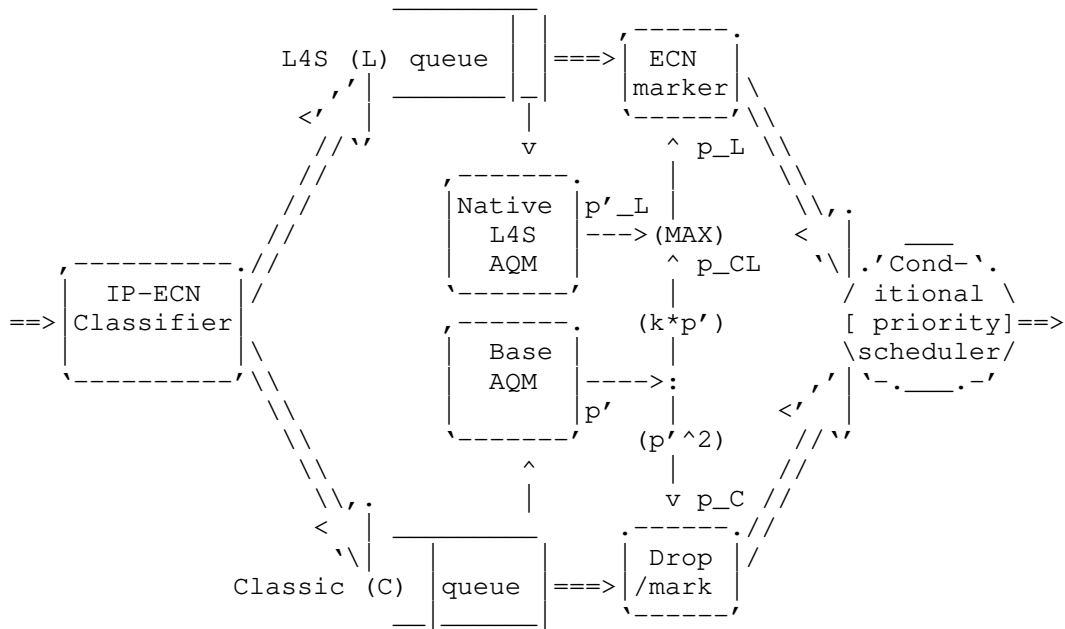


Figure 1: DualQ Coupled AQM Schematic

Legend: \implies traffic flow; \dashrightarrow 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 Tilman 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 $< 2 \text{ MTU}$ (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-iccrq-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

Transport Area Working Group
Internet-Draft
Updates: 3819 (if approved)
Intended status: Best Current Practice
Expires: November 26, 2021

B. Briscoe
Independent
J. Kaippallimalil
Futurewei
May 25, 2021

Guidelines for Adding Congestion Notification to Protocols that
Encapsulate IP
draft-ietf-tsvwg-ecn-encap-guidelines-16

Abstract

The purpose of this document is to guide the design of congestion notification in any lower layer or tunnelling protocol that encapsulates IP. The aim is for explicit congestion signals to propagate consistently from lower layer protocols into IP. Then the IP internetwork layer can act as a portability layer to carry congestion notification from non-IP-aware congested nodes up to the transport layer (L4). Following these guidelines should assure interworking among IP layer and lower layer congestion notification mechanisms, whether specified by the IETF or other standards bodies. This document updates the advice to subnetwork designers about ECN in RFC 3819.

Status of This Memo

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

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

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

This Internet-Draft will expire on November 26, 2021.

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
1.1. Update to RFC 3819	5
1.2. Scope	5
2. Terminology	7
3. Modes of Operation	9
3.1. Feed-Forward-and-Up Mode	9
3.2. Feed-Up-and-Forward Mode	11
3.3. Feed-Backward Mode	12
3.4. Null Mode	14
4. Feed-Forward-and-Up Mode: Guidelines for Adding Congestion Notification	14
4.1. IP-in-IP Tunnels with Shim Headers	15
4.2. Wire Protocol Design: Indication of ECN Support	16
4.3. Encapsulation Guidelines	18
4.4. Decapsulation Guidelines	20
4.5. Sequences of Similar Tunnels or Subnets	22
4.6. Reframing and Congestion Markings	22
5. Feed-Up-and-Forward Mode: Guidelines for Adding Congestion Notification	23
6. Feed-Backward Mode: Guidelines for Adding Congestion Notification	24
7. IANA Considerations	25
8. Security Considerations	25
9. Conclusions	26
10. Acknowledgements	27
11. Contributors	27
12. Comments Solicited	27
13. References	27
13.1. Normative References	27
13.2. Informative References	28
Appendix A. Changes in This Version (to be removed by RFC Editor)	33
Authors' Addresses	38

1. Introduction

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

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

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

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

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

Some lower layer technologies (e.g. MPLS, Ethernet) are used to form subnetworks with IP-aware nodes only at the edges. These networks are often sized so that it is rare for interior queues to overflow. However, until recently this was more due to the inability of TCP to

saturate the links. For many years, fixes such as window scaling [RFC7323] proved hard to deploy. And the Reno variant of TCP has remained in widespread use despite its inability to scale to high flow rates. However, now that modern operating systems are finally capable of saturating interior links, even the buffers of well-provisioned interior switches will need to signal episodes of queuing.

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

Similarly, ECN propagation is yet to be defined for many tunnelling protocols. [RFC6040] defines how ECN should be propagated for IP-in-IPv4 [RFC2003], IP-in-IPv6 [RFC2473] and IPsec [RFC4301] tunnels, but there are numerous other tunnelling protocols with a shim and/or a layer 2 header between two IP headers (v4 or v6). Some address ECN propagation between the IP headers, but many do not. This document gives guidance on how to address ECN propagation for future tunnelling protocols, and a companion standards track specification [I-D.ietf-tsvwg-rfc6040update-shim] updates those existing IP-shim-(L2)-IP protocols that are under IETF change control and still widely used.

Incremental deployment is the most delicate aspect when adding support for ECN. The original ECN protocol in IP [RFC3168] was carefully designed so that a congested buffer would not mark a packet (rather than drop it) unless both source and destination hosts were ECN-capable. Otherwise its congestion markings would never be detected and congestion would just build up further. However, to support congestion marking below the IP layer or within tunnels, it is not sufficient to only check that the two layer 4 transport endpoints support ECN; correct operation also depends on the decapsulator at each subnet or tunnel egress faithfully propagating congestion notifications to the higher layer. Otherwise, a legacy decapsulator might silently fail to propagate any ECN signals from the outer to the forwarded header. Then the lost signals would never be detected and again congestion would build up further. The guidelines given later require protocol designers to carefully consider incremental deployment, and suggest various safe approaches for different circumstances.

Of course, the IETF does not have standards authority over every link layer protocol. So this document gives guidelines for designing propagation of congestion notification across the interface between IP and protocols that may encapsulate IP (i.e. that can be layered beneath IP). Each lower layer technology will exhibit different issues and compromises, so the IETF or the relevant standards body

must be free to define the specifics of each lower layer congestion notification scheme. Nonetheless, if the guidelines are followed, congestion notification should interwork between different technologies, using IP in its role as a 'portability layer'.

Therefore, the capitalized terms 'SHOULD' or 'SHOULD NOT' are often used in preference to 'MUST' or 'MUST NOT', because it is difficult to know the compromises that will be necessary in each protocol design. If a particular protocol design chooses not to follow a 'SHOULD (NOT)' given in the advice below, it MUST include a sound justification.

It has not been possible to give common guidelines for all lower layer technologies, because they do not all fit a common pattern. Instead they have been divided into a few distinct modes of operation: feed-forward-and-upward; feed-upward-and-forward; feed-backward; and null mode. These modes are described in Section 3, then in the subsequent sections separate guidelines are given for each mode.

1.1. Update to RFC 3819

This document updates the brief advice to subnetwork designers about ECN in [RFC3819], by replacing the last two paragraphs of Section 13 with the following sentence:

By following the guidelines in [this document], subnetwork designers can enable a layer-2 protocol to participate in congestion control without dropping packets via propagation of explicit congestion notification (ECN [RFC3168]) to receivers.

and adding [this document] as an informative reference. {RFC Editor: Please replace both instances of [this document] above with the number of the present RFC when published.}

1.2. Scope

This document only concerns wire protocol processing of explicit notification of congestion. It makes no changes or recommendations concerning algorithms for congestion marking or for congestion response, because algorithm issues should be independent of the layer the algorithm operates in.

The default ECN semantics are described in [RFC3168] and updated by [RFC8311]. Also the guidelines for AQM designers [RFC7567] clarify the semantics of both drop and ECN signals from AQM algorithms. [RFC4774] is the appropriate best current practice specification of how algorithms with alternative semantics for the ECN field can be

partitioned from Internet traffic that uses the default ECN semantics. There are two main examples for how alternative ECN semantics have been defined in practice:

- o RFC 4774 suggests using the ECN field in combination with a Diffserv codepoint such as in PCN [RFC6660], Voice over 3G [UTRAN] or Voice over LTE (VoLTE) [LTE-RA];
- o RFC 8311 suggests using the ECT(1) codepoint of the ECN field to indicate alternative semantics such as for the experimental Low Latency Low Loss Scalable throughput (L4S) service [I-D.ietf-tsvwg-ecn-l4s-id]).

The aim is that the default rules for encapsulating and decapsulating the ECN field are sufficiently generic that tunnels and subnets will encapsulate and decapsulate packets without regard to how algorithms elsewhere are setting or interpreting the semantics of the ECN field. [RFC6040] updates RFC 4774 to allow alternative encapsulation and decapsulation behaviours to be defined for alternative ECN semantics. However it reinforces the same point - that it is far preferable to try to fit within the common ECN encapsulation and decapsulation behaviours, because expecting all lower layer technologies and tunnels to be updated is likely to be completely impractical.

Alternative semantics for the ECN field can be defined to depend on the traffic class indicated by the DSCP. Therefore correct propagation of congestion signals could depend on correct propagation of the DSCP between the layers and along the path. For instance, if the meaning of the ECN field depends on the DSCP (as in PCN or VoLTE) and if the outer DSCP is stripped on decapsulation, as in the pipe model of [RFC2983], the special semantics of the ECN field would be lost. Similarly, if the DSCP is changed at the boundary between Diffserv domains, the special ECN semantics would also be lost. This is an important implication of the localized scope of most Diffserv arrangements. In this document, correct propagation of traffic class information is assumed, while what 'correct' means and how it is achieved is covered elsewhere (e.g. RFC 2983) and is outside the scope of the present document.

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

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

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

2. Terminology

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

Further terminology used within this document:

Protocol data unit (PDU): Information that is delivered as a unit among peer entities of a layered network consisting of protocol control information (typically a header) and possibly user data (payload) of that layer. The scope of this document includes layer 2 and layer 3 networks, where the PDU is respectively termed a frame or a packet (or a cell in ATM). PDU is a general term for any of these. This definition also includes a payload with a shim header lying somewhere between layer 2 and 3.

Transport: The end-to-end transmission control function, conventionally considered at layer-4 in the OSI reference model. Given the audience for this document will often use the word transport to mean low level bit carriage, whenever the term is used it will be qualified, e.g. 'L4 transport'.

Encapsulator: The link or tunnel endpoint function that adds an outer header to a PDU (also termed the 'link ingress', the 'subnet ingress', the 'ingress tunnel endpoint' or just the 'ingress' where the context is clear).

Decapsulator: The link or tunnel endpoint function that removes an outer header from a PDU (also termed the 'link egress', the 'subnet egress', the 'egress tunnel endpoint' or just the 'egress' where the context is clear).

Incoming header: The header of an arriving PDU before encapsulation.

Outer header: The header added to encapsulate a PDU.

Inner header: The header encapsulated by the outer header.

Outgoing header: The header forwarded by the decapsulator.

CE: Congestion Experienced [RFC3168]

ECT: ECN-Capable (L4) Transport [RFC3168]

Not-ECT: Not ECN-Capable (L4) Transport [RFC3168]

Load Regulator: For each flow of PDUs, the transport function that is capable of controlling the data rate. Typically located at the data source, but in-path nodes can regulate load in some congestion control arrangements (e.g. admission control, policing nodes or transport circuit-breakers [RFC8084]). Note the term "a function capable of controlling the load" deliberately includes a transport that does not actually control the load responsively but ideally it ought to (e.g. a sending application without congestion control that uses UDP).

ECN-PDU: A PDU at the IP layer or below with a capacity to signal congestion that is part of a congestion control feedback loop within which all the nodes necessary to propagate the signal back to the Load Regulator are capable of doing that propagation. An IP packet with a non-zero ECN field implies that the endpoints are ECN-capable, so this would be an ECN-PDU. However, ECN-PDU is intended to be a general term for a PDU at lower layers, as well as at the IP layer.

Not-ECN-PDU: A PDU at the IP layer or below that is part of a congestion control feedback-loop within which at least one node necessary to propagate any explicit congestion notification signals back to the Load Regulator is not capable of doing that propagation.

3. Modes of Operation

This section sets down the different modes by which congestion information is passed between the lower layer and the higher one. It acts as a reference framework for the following sections, which give normative guidelines for designers of explicit congestion notification protocols, taking each mode in turn:

Feed-Forward-and-Up: Nodes feed forward congestion notification towards the egress within the lower layer then up and along the layers towards the end-to-end destination at the transport layer. The following local optimisation is possible:

Feed-Up-and-Forward: A lower layer switch feeds-up congestion notification directly into the higher layer (e.g. into the ECN field in the IP header), irrespective of whether the node is at the egress of a subnet.

Feed-Backward: Nodes feed back congestion signals towards the ingress of the lower layer and (optionally) attempt to control congestion within their own layer.

Null: Nodes cannot experience congestion at the lower layer except at ingress nodes (which are IP-aware or equivalently higher-layer-aware).

3.1. Feed-Forward-and-Up Mode

Like IP and MPLS, many subnet technologies are based on self-contained protocol data units (PDUs) or frames sent unreliably. They provide no feedback channel at the subnetwork layer, instead relying on higher layers (e.g. TCP) to feed back loss signals.

In these cases, ECN may best be supported by standardising explicit notification of congestion into the lower layer protocol that carries the data forwards. Then a specification is needed for how the egress of the lower layer subnet propagates this explicit signal into the forwarded upper layer (IP) header. This signal continues forwards until it finally reaches the destination transport (at L4). Then typically the destination will feed this congestion notification back to the source transport using an end-to-end protocol (e.g. TCP). This is the arrangement that has already been used to add ECN to IP-in-IP tunnels [RFC6040], IP-in-MPLS and MPLS-in-MPLS [RFC5129].

This mode is illustrated in Figure 1. Along the middle of the figure, layers 2, 3 and 4 of the protocol stack are shown, and one packet is shown along the bottom as it progresses across the network from source to destination, crossing two subnets connected by a

router, and crossing two switches on the path across each subnet. Congestion at the output of the first switch (shown as *) leads to a congestion marking in the L2 header (shown as C in the illustration of the packet). The chevrons show the progress of the resulting congestion indication. It is propagated from link to link across the subnet in the L2 header, then when the router removes the marked L2 header, it propagates the marking up into the L3 (IP) header. The router forwards the marked L3 header into subnet 2, and when it adds a new L2 header it copies the L3 marking into the L2 header as well, as shown by the 'C's in both layers (assuming the technology of subnet 2 also supports explicit congestion marking).

Note that there is no implication that each 'C' marking is encoded the same; a different encoding might be used for the 'C' marking in each protocol.

Finally, for completeness, we show the L3 marking arriving at the destination, where the host transport protocol (e.g. TCP) feeds it back to the source in the L4 acknowledgement (the 'C' at L4 in the packet at the top of the diagram).

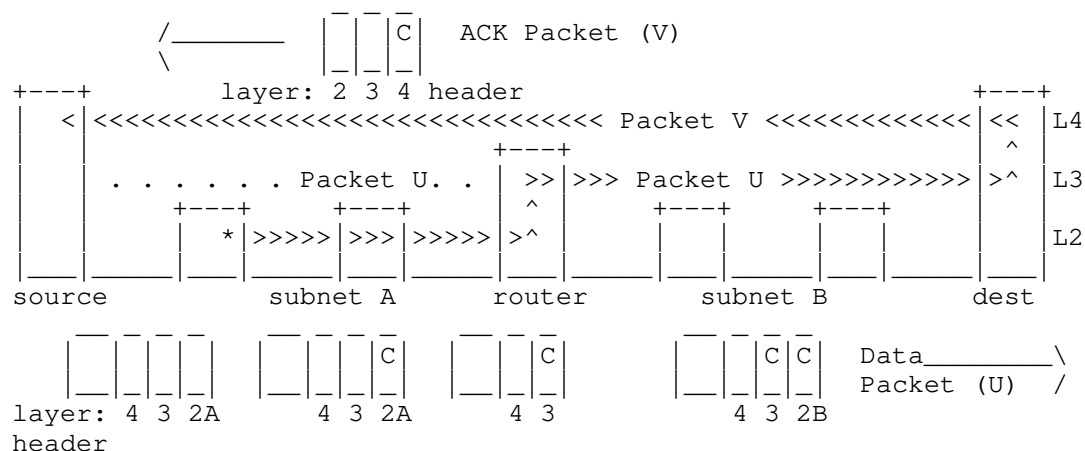


Figure 1: Feed-Forward-and-Up Mode

Of course, modern networks are rarely as simple as this text-book example, often involving multiple nested layers. For example, a 3GPP mobile network may have two IP-in-IP (GTP [GTPv1]) tunnels in series and an MPLS backhaul between the base station and the first router. Nonetheless, the example illustrates the general idea of feeding congestion notification forward then upward whenever a header is removed at the egress of a subnet.

Note that the FECN (forward ECN) bit in Frame Relay [Buck00] and the explicit forward congestion indication (EFCI [ITU-T.I.371]) bit in ATM user data cells follow a feed-forward pattern. However, in ATM, this arrangement is only part of a feed-forward-and-backward pattern at the lower layer, not feed-forward-and-up out of the lower layer--the intention was never to interface to IP ECN at the subnet egress. To our knowledge, Frame Relay FECN is solely used to detect where more capacity should be provisioned.

3.2. Feed-Up-and-Forward Mode

Ethernet is particularly difficult to extend incrementally to support explicit congestion notification. One way to support ECN in such cases has been to use so called 'layer-3 switches'. These are Ethernet switches that dig into the Ethernet payload to find an IP header and manipulate or act on certain IP fields (specifically Diffserv & ECN). For instance, in Data Center TCP [RFC8257], layer-3 switches are configured to mark the ECN field of the IP header within the Ethernet payload when their output buffer becomes congested. With respect to switching, a layer-3 switch acts solely on the addresses in the Ethernet header; it does not use IP addresses, and it does not decrement the TTL field in the IP header.

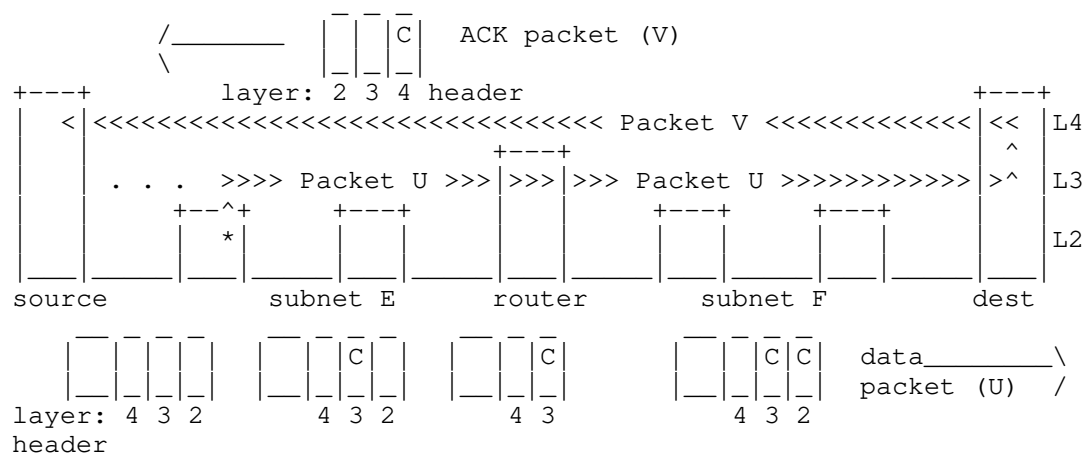


Figure 2: Feed-Up-and-Forward Mode

By comparing Figure 2 with Figure 1, it can be seen that subnet E (perhaps a subnet of layer-3 Ethernet switches) works in feed-up-and-forward mode by notifying congestion directly into L3 at the point of congestion, even though the congested switch does not otherwise act at L3. In this example, the technology in subnet F (e.g. MPLS) does

support ECN natively, so when the router adds the layer-2 header it copies the ECN marking from L3 to L2 as well.

3.3. Feed-Backward Mode

In some layer 2 technologies, explicit congestion notification has been defined for use internally within the subnet with its own feedback and load regulation, but typically the interface with IP for ECN has not been defined.

For instance, for the available bit-rate (ABR) service in ATM, the relative rate mechanism was one of the more popular mechanisms for managing traffic, tending to supersede earlier designs. In this approach ATM switches send special resource management (RM) cells in both the forward and backward directions to control the ingress rate of user data into a virtual circuit. If a switch buffer is approaching congestion or is congested it sends an RM cell back towards the ingress with respectively the No Increase (NI) or Congestion Indication (CI) bit set in its message type field [ATM-TM-ABR]. The ingress then holds or decreases its sending bit-rate accordingly.

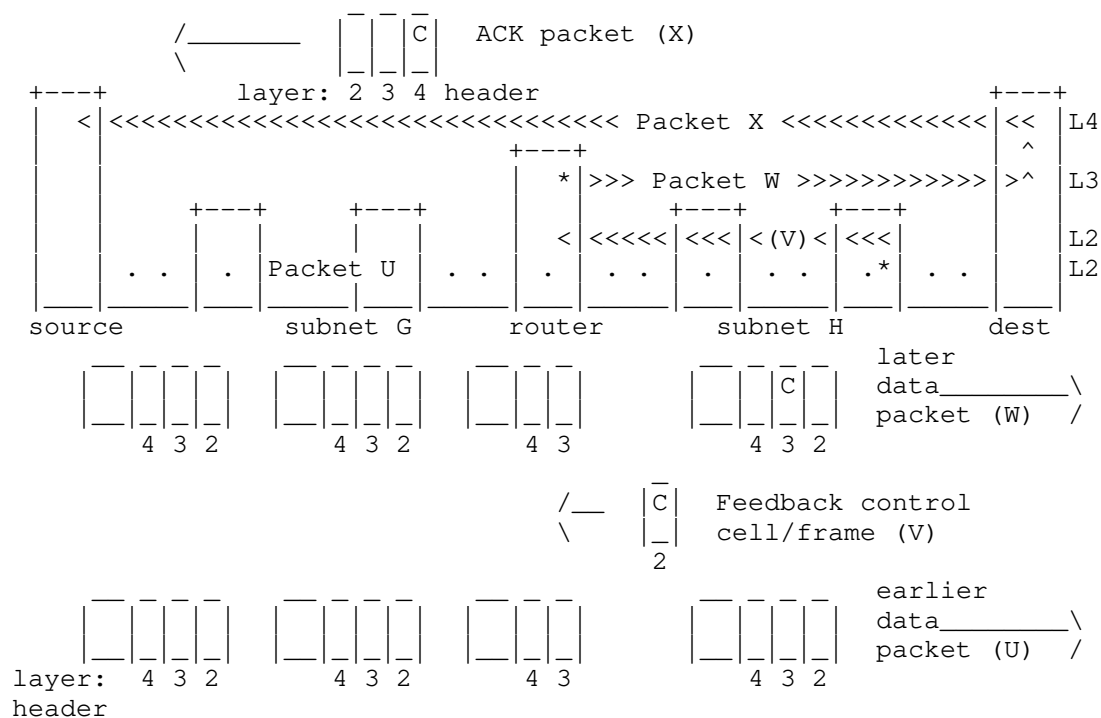


Figure 3: Feed-Backward Mode

ATM's feed-backward approach does not fit well when layered beneath IP's feed-forward approach--unless the initial data source is the same node as the ATM ingress. Figure 3 shows the feed-backward approach being used in subnet H. If the final switch on the path is congested (*), it does not feed-forward any congestion indications on packet (U). Instead it sends a control cell (V) back to the router at the ATM ingress.

However, the backward feedback does not reach the original data source directly because IP does not support backward feedback (and subnet G is independent of subnet H). Instead, the router in the middle throttles down its sending rate but the original data sources don't reduce their rates. The resulting rate mismatch causes the middle router's buffer at layer 3 to back up until it becomes congested, which it signals forwards on later data packets at layer 3 (e.g. packet W). Note that the forward signal from the middle router is not triggered directly by the backward signal. Rather, it is triggered by congestion resulting from the middle router's mismatched rate response to the backward signal.

In response to this later forward signalling, end-to-end feedback at layer-4 finally completes the tortuous path of congestion indications back to the origin data source, as before.

Quantized congestion notification (QCN [IEEE802.1Q]) would suffer from similar problems if extended to multiple subnets. However, from the start QCN was clearly characterized as solely applicable to a single subnet (see Section 6).

3.4. Null Mode

Often link and physical layer resources are 'non-blocking' by design. In these cases congestion notification may be implemented but it does not need to be deployed at the lower layer; ECN in IP would be sufficient.

A degenerate example is a point-to-point Ethernet link. Excess loading of the link merely causes the queue from the higher layer to back up, while the lower layer remains immune to congestion. Even a whole meshed subnetwork can be made immune to interior congestion by limiting ingress capacity and sufficient sizing of interior links, e.g. a non-blocking fat-tree network [Leiserson85]. An alternative to fat links near the root is numerous thin links with multi-path routing to ensure even worst-case patterns of load cannot congest any link, e.g. a Clos network [Clos53].

4. Feed-Forward-and-Up Mode: Guidelines for Adding Congestion Notification

Feed-forward-and-up is the mode already used for signalling ECN up the layers through MPLS into IP [RFC5129] and through IP-in-IP tunnels [RFC6040], whether encapsulating with IPv4 [RFC2003], IPv6 [RFC2473] or IPsec [RFC4301]. These RFCs take a consistent approach and the following guidelines are designed to ensure this consistency continues as ECN support is added to other protocols that encapsulate IP. The guidelines are also designed to ensure compliance with the more general best current practice for the design of alternate ECN schemes given in [RFC4774] and extended by [RFC8311].

The rest of this section is structured as follows:

- o Section 4.1 addresses the most straightforward cases, where [RFC6040] can be applied directly to add ECN to tunnels that are effectively IP-in-IP tunnels, but with shim header(s) between the IP headers.
- o The subsequent sections give guidelines for adding ECN to a subnet technology that uses feed-forward-and-up mode like IP, but it is

not so similar to IP that [RFC6040] rules can be applied directly. Specifically:

- * Sections 4.2, 4.3 and 4.4 respectively address how to add ECN support to the wire protocol and to the encapsulators and decapsulators at the ingress and egress of the subnet.
- * Section 4.5 deals with the special, but common, case of sequences of tunnels or subnets that all use the same technology
- * Section 4.6 deals with the question of reframing when IP packets do not map 1:1 into lower layer frames.

4.1. IP-in-IP Tunnels with Shim Headers

A common pattern for many tunnelling protocols is to encapsulate an inner IP header with shim header(s) then an outer IP header. A shim header is defined as one that is not sufficient alone to forward the packet as an outer header. Another common pattern is for a shim to encapsulate a layer 2 (L2) header, which in turn encapsulates (or might encapsulate) an IP header. [I-D.ietf-tsvwg-rfc6040update-shim] clarifies that RFC 6040 is just as applicable when there are shim(s) and possibly a L2 header between two IP headers.

However, it is not always feasible or necessary to propagate ECN between IP headers when separated by a shim. For instance, it might be too costly to dig to arbitrary depths to find an inner IP header, there may be little or no congestion within the tunnel by design (see null mode in Section 3.4 above), or a legacy implementation might not support ECN. In cases where a tunnel does not support ECN, it is important that the ingress does not copy the ECN field from an inner IP header to an outer. Therefore section 4 of [I-D.ietf-tsvwg-rfc6040update-shim] requires network operators to configure the ingress of a tunnel that does not support ECN so that it zeros the ECN field in the outer IP header.

Nonetheless, in many cases it is feasible to propagate the ECN field between IP headers separated by shim header(s) and/or a L2 header. Particularly in the typical case when the outer IP header and the shim(s) are added (or removed) as part of the same procedure. Even if the shim(s) encapsulate a L2 header, it is often possible to find an inner IP header within the L2 PDU and propagate ECN between that and the outer IP header. This can be thought of as a special case of the feed-up-and-forward mode (Section 3.2), so the guidelines for this mode apply (Section 5).

Numerous shim protocols have been defined for IP tunnelling. More recent ones e.g. Geneve [RFC8926] and Generic UDP Encapsulation (GUE) [I-D.ietf-intarea-gue] cite and follow RFC 6040. And some earlier ones, e.g. CAPWAP [RFC5415] and LISP [RFC6830], cite RFC 3168, which is compatible with RFC 6040.

However, as Section 9.3 of RFC 3168 pointed out, ECN support needs to be defined for many earlier shim-based tunnelling protocols, e.g. L2TPv2 [RFC2661], L2TPv3 [RFC3931], GRE [RFC2784], PPTP [RFC2637], GTP [GTPv1], [GTPv1-U], [GTPv2-C] and Teredo [RFC4380] as well as some recent ones, e.g. VXLAN [RFC7348], NVGRE [RFC7637] and NSH [RFC8300].

All these IP-based encapsulations can be updated in one shot by simple reference to RFC 6040. However, it would not be appropriate to update all these protocols from within the present guidance document. Instead a companion specification [I-D.ietf-tsvwg-rfc6040update-shim] has been prepared that has the appropriate standards track status to update standards track protocols. For those that are not under IETF change control [I-D.ietf-tsvwg-rfc6040update-shim] can only recommend that the relevant body updates them.

4.2. Wire Protocol Design: Indication of ECN Support

This section is intended to guide the redesign of any lower layer protocol that encapsulate IP to add native ECN support at the lower layer. It reflects the approaches used in [RFC6040] and in [RFC5129]. Therefore IP-in-IP tunnels or IP-in-MPLS or MPLS-in-MPLS encapsulations that already comply with [RFC6040] or [RFC5129] will already satisfy this guidance.

A lower layer (or subnet) congestion notification system:

1. SHOULD NOT apply explicit congestion notifications to PDUs that are destined for legacy layer-4 transport implementations that will not understand ECN, and
2. SHOULD NOT apply explicit congestion notifications to PDUs if the egress of the subnet might not propagate congestion notifications onward into the higher layer.

We use the term ECN-PDUs for a PDU on a feedback loop that will propagate congestion notification properly because it meets both the above criteria. And a Not-ECN-PDU is a PDU on a feedback loop that does not meet at least one of the criteria, and will therefore not propagate congestion notification properly. A

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

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

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

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

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

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

In MPLS, header space is extremely limited, therefore RFC5129 does not provide a field in the MPLS header to indicate whether the PDU is an ECN-PDU or a Not-ECN-PDU. Instead, interior nodes in a domain are allowed to set explicit congestion indications without checking whether the PDU is destined for a L4 transport that will understand them. Nonetheless, this is made safe by requiring that the network operator upgrades all decapsulating edges of a whole domain at once, as soon as even one switch within the domain is configured to mark rather than drop during congestion. Therefore, any edge node that might decapsulate a packet will be capable of checking whether the higher layer transport is ECN-capable. When decapsulating a CE-marked packet, if the decapsulator discovers that the higher layer (inner header) indicates the transport is not ECN-capable, it drops the packet--effectively on behalf of the earlier congested node (see Decapsulation Guideline 1 in Section 4.4).

It was only appropriate to define such an incremental deployment strategy because MPLS is targeted solely at professional operators, who can be expected to ensure that a whole subnetwork is consistently configured. This strategy might not be appropriate for other link technologies targeted at zero-configuration deployment or deployment by the general public (e.g. Ethernet). For such 'plug-and-play' environments it will be necessary to invent a failsafe approach that ensures congestion markings will never fall into black holes, no matter how inconsistently a system is put together. Alternatively, congestion notification relying on correct system configuration could be confined to flavours of Ethernet intended only for professional network operators, such as Provider Backbone Bridges (PBB [IEEE802.1Q]; previously 802.1ah).

ECN support in TRILL [I-D.ietf-trill-ecn-support] provides a good example of how to add ECN to a lower layer protocol without relying on careful and consistent operator configuration. TRILL provides an extension header word with space for flags of different categories depending on whether logic to understand the extension is critical. The congestion experienced marking has been defined as a 'critical ingress-to-egress' flag. So if a transit RBridge sets this flag and an egress RBridge does not have any logic to process it, it will drop it; which is the desired default action anyway. Therefore TRILL RBridges can be updated with support for ECN in no particular order and, at the egress of the TRILL campus, congestion notification will be propagated to IP as ECN whenever ECN logic has been implemented, or as drop otherwise.

QCN [IEEE802.1Q] is not intended to extend beyond a single subnet, or to interoperate with ECN. Nonetheless, the way QCN indicates to lower layer devices that the end-points will not understand QCN provides another example that a lower layer protocol designer might be able to mimic for their scenario. An operator can define certain Priority Code Points (PCPs [IEEE802.1Q]; previously 802.1p) to indicate non-QCN frames and an ingress bridge is required to map arriving not-QCN-capable IP packets to one of these non-QCN PCPs.

4.3. Encapsulation Guidelines

This section is intended to guide the redesign of any node that encapsulates IP with a lower layer header when adding native ECN support to the lower layer protocol. It reflects the approaches used in [RFC6040] and in [RFC5129]. Therefore IP-in-IP tunnels or IP-in-MPLS or MPLS-in-MPLS encapsulations that already comply with [RFC6040] or [RFC5129] will already satisfy this guidance.

1. Egress Capability Check: A subnet ingress needs to be sure that the corresponding egress of a subnet will propagate any

congestion notification added to the outer header across the subnet. This is necessary in addition to checking that an incoming PDU indicates an ECN-capable (L4) transport. Examples of how this guarantee might be provided include:

- * by configuration (e.g. if any label switches in a domain support ECN marking, [RFC5129] requires all egress nodes to have been configured to propagate ECN)
 - * by the ingress explicitly checking that the egress propagates ECN (e.g. an early attempt to add ECN support to TRILL used IS-IS to check path capabilities before adding ECN extension flags to each frame [RFC7780]).
 - * by inherent design of the protocol (e.g. by encoding ECN marking on the outer header in such a way that a legacy egress that does not understand ECN will consider the PDU corrupt or invalid and discard it, thus at least propagating a form of congestion signal).
2. Egress Fails Capability Check: If the ingress cannot guarantee that the egress will propagate congestion notification, the ingress SHOULD disable ECN at the lower layer when it forwards the PDU. An example of how the ingress might disable ECN at the lower layer would be by setting the outer header of the PDU to identify it as a Not-ECN-PDU, assuming the subnet technology supports such a concept.
 3. Standard Congestion Monitoring Baseline: Once the ingress to a subnet has established that the egress will correctly propagate ECN, on encapsulation it SHOULD encode the same level of congestion in outer headers as is arriving in incoming headers. For example it might copy any incoming congestion notification into the outer header of the lower layer protocol.

This ensures that bulk congestion monitoring of outer headers (e.g. by a network management node monitoring ECN in passing frames) will measure congestion accumulated along the whole upstream path - since the Load Regulator not just since the ingress of the subnet. A node that is not the Load Regulator SHOULD NOT re-initialize the level of CE markings in the outer to zero.

It would still also be possible to measure congestion introduced across one subnet (or tunnel) by subtracting the level of CE markings on inner headers from that on outer headers (see Appendix C of [RFC6040]). For example:

- * If this guideline has been followed and if the level of CE markings is 0.4% on the outer and 0.1% on the inner, 0.4% congestion has been introduced across all the networks since the load regulator, and 0.3% ($= 0.4\% - 0.1\%$) has been introduced since the ingress to the current subnet (or tunnel);
- * Without this guideline, if the subnet ingress had re-initialized the outer congestion level to zero, the outer and inner would measure 0.1% and 0.3%. It would still be possible to infer that the congestion introduced since the Load Regulator was 0.4% ($= 0.1\% + 0.3\%$). But only if the monitoring system somehow knows whether the subnet ingress re-initialized the congestion level.

As long as subnet and tunnel technologies use the standard congestion monitoring baseline in this guideline, monitoring systems will know to use the former approach, rather than having to "somehow know" which approach to use.

4.4. Decapsulation Guidelines

This section is intended to guide the redesign of any node that decapsulates IP from within a lower layer header when adding native ECN support to the lower layer protocol. It reflects the approaches used in [RFC6040] and in [RFC5129]. Therefore IP-in-IP tunnels or IP-in-MPLS or MPLS-in-MPLS encapsulations that already comply with [RFC6040] or [RFC5129] will already satisfy this guidance.

A subnet egress SHOULD NOT simply copy congestion notification from outer headers to the forwarded header. It SHOULD calculate the outgoing congestion notification field from the inner and outer headers using the following guidelines. If there is any conflict, rules earlier in the list take precedence over rules later in the list:

1. If the arriving inner header is a Not-ECN-PDU it implies the L4 transport will not understand explicit congestion markings.
Then:
 - * If the outer header carries an explicit congestion marking, drop is the only indication of congestion that the L4 transport will understand. If the congestion marking is the most severe possible, the packet MUST be dropped. However, if congestion can be marked with multiple levels of severity and the packet's marking is not the most severe, this requirement can be relaxed to: the packet SHOULD be dropped.

- * If the outer is an ECN-PDU that carries no indication of congestion or a Not-ECN-PDU the PDU SHOULD be forwarded, but still as a Not-ECN-PDU.
- 2. If the outer header does not support explicit congestion notification (a Not-ECN-PDU), but the inner header does (an ECN-PDU), the inner header SHOULD be forwarded unchanged.
- 3. In some lower layer protocols congestion may be signalled as a numerical level, such as in the control frames of quantized congestion notification (QCN [IEEE802.1Q]). If such a multi-bit encoding encapsulates an ECN-capable IP data packet, a function will be needed to convert the quantized congestion level into the frequency of congestion markings in outgoing IP packets.
- 4. Congestion indications might be encoded by a severity level. For instance increasing levels of congestion might be encoded by numerically increasing indications, e.g. pre-congestion notification (PCN) can be encoded in each PDU at three severity levels in IP or MPLS [RFC6660] and the default encapsulation and decapsulation rules [RFC6040] are compatible with this interpretation of the ECN field.

If the arriving inner header is an ECN-PDU, where the inner and outer headers carry indications of congestion of different severity, the more severe indication SHOULD be forwarded in preference to the less severe.

- 5. The inner and outer headers might carry a combination of congestion notification fields that should not be possible given any currently used protocol transitions. For instance, if Encapsulation Guideline 3 in Section 4.3 had been followed, it should not be possible to have a less severe indication of congestion in the outer than in the inner. It MAY be appropriate to log unexpected combinations of headers and possibly raise an alarm.

If a safe outgoing codepoint can be defined for such a PDU, the PDU SHOULD be forwarded rather than dropped. Some implementers discard PDUs with currently unused combinations of headers just in case they represent an attack. However, an approach using alarms and policy-mediated drop is preferable to hard-coded drop, so that operators can keep track of possible attacks but currently unused combinations are not precluded from future use through new standards actions.

4.5. Sequences of Similar Tunnels or Subnets

In some deployments, particularly in 3GPP networks, an IP packet may traverse two or more IP-in-IP tunnels in sequence that all use identical technology (e.g. GTP).

In such cases, it would be sufficient for every encapsulation and decapsulation in the chain to comply with RFC 6040. Alternatively, as an optimisation, a node that decapsulates a packet and immediately re-encapsulates it for the next tunnel MAY copy the incoming outer ECN field directly to the outgoing outer and the incoming inner ECN field directly to the outgoing inner. Then the overall behavior across the sequence of tunnel segments would still be consistent with RFC 6040.

Appendix C of RFC6040 describes how a tunnel egress can monitor how much congestion has been introduced within a tunnel. A network operator might want to monitor how much congestion had been introduced within a whole sequence of tunnels. Using the technique in Appendix C of RFC6040 at the final egress, the operator could monitor the whole sequence of tunnels, but only if the above optimisation were used consistently along the sequence of tunnels, in order to make it appear as a single tunnel. Therefore, tunnel endpoint implementations SHOULD allow the operator to configure whether this optimisation is enabled.

When ECN support is added to a subnet technology, consideration SHOULD be given to a similar optimisation between subnets in sequence if they all use the same technology.

4.6. Reframing and Congestion Markings

The guidance in this section is worded in terms of framing boundaries, but it applies equally whether the protocol data units are frames, cells or packets.

Where an AQM marks the ECN field of IP packets as they queue into a layer-2 link, there will be no problem with framing boundaries, because the ECN markings would be applied directly to IP packets. The guidance in this section is only applicable where an ECN capability is being added to a layer-2 protocol so that layer-2 frames can be ECN-marked by an AQM at layer-2. This would only be necessary where AQM will be applied at pure layer-2 nodes (without IP-awareness).

When layer-2 frame headers are stripped off and IP PDUs with different boundaries are forwarded, the provisions in RFC7141 for handling congestion indications when splitting or merging packets

apply (see Section 2.4 of [RFC7141]). Those provisions include: "The general rule to follow is that the number of octets in packets with congestion indications SHOULD be equivalent before and after merging or splitting." See RFC 7141 for the complete provisions and related discussion, including an exception to that general rule.

As also recommended in RFC 7141, the mechanism for propagating congestion indications SHOULD ensure that any new incoming congestion indication is propagated immediately, and not held awaiting possible arrival of further congestion indications sufficient to indicate congestion for all of the octets of an outgoing IP PDU.

5. Feed-Up-and-Forward Mode: Guidelines for Adding Congestion Notification

The guidance in this section is applicable, for example, when IP packets:

- o are encapsulated in Ethernet headers, which have no support for ECN;
- o are forwarded by the eNode-B (base station) of a 3GPP radio access network, which is required to apply ECN marking during congestion, [LTE-RA], [UTRAN], but the Packet Data Convergence Protocol (PDCP) that encapsulates the IP header over the radio access has no support for ECN.

This guidance also generalizes to encapsulation by other subnet technologies with no native support for explicit congestion notification at the lower layer, but with support for finding and processing an IP header. It is unlikely to be applicable or necessary for IP-in-IP encapsulation, where feed-forward-and-up mode based on [RFC6040] would be more appropriate.

Marking the IP header while switching at layer-2 (by using a layer-3 switch) or while forwarding in a radio access network seems to represent a layering violation. However, it can be considered as a benign optimisation if the guidelines below are followed. Feed-up-and-forward is certainly not a general alternative to implementing feed-forward congestion notification in the lower layer, because:

- o IPv4 and IPv6 are not the only layer-3 protocols that might be encapsulated by lower layer protocols
- o Link-layer encryption might be in use, making the layer-2 payload inaccessible

- o Many Ethernet switches do not have 'layer-3 switch' capabilities so they cannot read or modify an IP payload
- o It might be costly to find an IP header (v4 or v6) when it may be encapsulated by more than one lower layer header, e.g. Ethernet MAC in MAC ([IEEE802.1Q]; previously 802.1ah).

Nonetheless, configuring lower layer equipment to look for an ECN field in an encapsulated IP header is a useful optimisation. If the implementation follows the guidelines below, this optimisation does not have to be confined to a controlled environment such as within a data centre; it could usefully be applied on any network--even if the operator is not sure whether the above issues will never apply:

1. If a native lower-layer congestion notification mechanism exists for a subnet technology, it is safe to mix feed-up-and-forward with feed-forward-and-up on other switches in the same subnet. However, it will generally be more efficient to use the native mechanism.
 2. The depth of the search for an IP header SHOULD be limited. If an IP header is not found soon enough, or an unrecognized or unreadable header is encountered, the switch SHOULD resort to an alternative means of signalling congestion (e.g. drop, or the native lower layer mechanism if available).
 3. It is sufficient to use the first IP header found in the stack; the egress of the relevant tunnel can propagate congestion notification upwards to any more deeply encapsulated IP headers later.
6. Feed-Backward Mode: Guidelines for Adding Congestion Notification

It can be seen from Section 3.3 that congestion notification in a subnet using feed-backward mode has generally not been designed to be directly coupled with IP layer congestion notification. The subnet attempts to minimize congestion internally, and if the incoming load at the ingress exceeds the capacity somewhere through the subnet, the layer 3 buffer into the ingress backs up. Thus, a feed-backward mode subnet is in some sense similar to a null mode subnet, in that there is no need for any direct interaction between the subnet and higher layer congestion notification. Therefore no detailed protocol design guidelines are appropriate. Nonetheless, a more general guideline is appropriate:

A subnetwork technology intended to eventually interface to IP SHOULD NOT be designed using only the feed-backward mode, which is certainly best for a stand-alone subnet, but would need to be

modified to work efficiently as part of the wider Internet, because IP uses feed-forward-and-up mode.

The feed-backward approach at least works beneath IP, where the term 'works' is used only in a narrow functional sense because feed-backward can result in very inefficient and sluggish congestion control--except if it is confined to the subnet directly connected to the original data source, when it is faster than feed-forward. It would be valid to design a protocol that could work in feed-backward mode for paths that only cross one subnet, and in feed-forward-and-up mode for paths that cross subnets.

In the early days of TCP/IP, a similar feed-backward approach was tried for explicit congestion signalling, using source-quench (SQ) ICMP control packets. However, SQ fell out of favour and is now formally deprecated [RFC6633]. The main problem was that it is hard for a data source to tell the difference between a spoofed SQ message and a quench request from a genuine buffer on the path. It is also hard for a lower layer buffer to address an SQ message to the original source port number, which may be buried within many layers of headers, and possibly encrypted.

QCN (also known as backward congestion notification, BCN; see Sections 30--33 of [IEEE802.1Q]; previously known as 802.1Qau) uses a feed-backward mode structurally similar to ATM's relative rate mechanism. However, QCN confines its applicability to scenarios such as some data centres where all endpoints are directly attached by the same Ethernet technology. If a QCN subnet were later connected into a wider IP-based internetwork (e.g. when attempting to interconnect multiple data centres) it would suffer the inefficiency shown in Figure 3.

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

If a lower layer wire protocol is redesigned to include explicit congestion signalling in-band in the protocol header, care SHOULD be taken to ensure that the field used is specified as mutable during transit. Otherwise interior nodes signalling congestion would invalidate any authentication protocol applied to the lower layer header--by altering a header field that had been assumed as immutable.

The redesign of protocols that encapsulate IP in order to propagate congestion signals between layers raises potential signal integrity

concerns. Experimental or proposed approaches exist for assuring the end-to-end integrity of in-band congestion signals, e.g.:

- o Congestion exposure (ConEx) for networks to audit that their congestion signals are not being suppressed by other networks or by receivers, and for networks to police that senders are responding sufficiently to the signals, irrespective of the L4 transport protocol used [RFC7713].
- o A test for a sender to detect whether a network or the receiver is suppressing congestion signals (for example see 2nd para of Section 20.2 of [RFC3168]).

Given these end-to-end approaches are already being specified, it would make little sense to attempt to design hop-by-hop congestion signal integrity into a new lower layer protocol, because end-to-end integrity inherently achieves hop-by-hop integrity.

Section 6 gives vulnerability to spoofing as one of the reasons for deprecating feed-backward mode.

9. Conclusions

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

- o A wide range of tunnelling protocols including those with various forms of shim header between two IP headers, possibly also separated by a L2 header;
- o A wide range of subnet technologies, particularly those that work in the same 'feed-forward-and-up' mode that is used to support ECN in IP and MPLS.

Guidelines have been defined for supporting propagation of ECN between Ethernet and IP on so-called Layer-3 Ethernet switches, using a 'feed-up-and-forward' mode. This approach could enable other subnet technologies to pass ECN signals into the IP layer, even if they do not support ECN natively.

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

10. Acknowledgements

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

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

11. Contributors

Pat Thaler
Broadcom Corporation (retired)
CA
USA

Pat was a co-author of this draft, but retired before its publication.

12. 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.

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>>.

- [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>>.
- [RFC3819] Karn, P., Ed., Bormann, C., Fairhurst, G., Grossman, D., Ludwig, R., Mahdavi, J., Montenegro, G., Touch, J., and L. Wood, "Advice for Internet Subnetwork Designers", BCP 89, RFC 3819, DOI 10.17487/RFC3819, July 2004, <<https://www.rfc-editor.org/info/rfc3819>>.
- [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>>.
- [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, <<https://www.rfc-editor.org/info/rfc6040>>.
- [RFC7141] Briscoe, B. and J. Manner, "Byte and Packet Congestion Notification", BCP 41, RFC 7141, DOI 10.17487/RFC7141, February 2014, <<https://www.rfc-editor.org/info/rfc7141>>.

13.2. Informative References

- [ATM-TM-ABR] Cisco, "Understanding the Available Bit Rate (ABR) Service Category for ATM VCs", Design Technote 10415, June 2005.
- [Buck00] Buckwalter, J., "Frame Relay: Technology and Practice", Pub. Addison Wesley ISBN-13: 978-0201485240, 2000.
- [Clos53] Clos, C., "A Study of Non-Blocking Switching Networks", Bell Systems Technical Journal 32(2):406--424, March 1953.
- [GTPv1] 3GPP, "GPRS Tunnelling Protocol (GTP) across the Gn and Gp interface", Technical Specification TS 29.060.
- [GTPv1-U] 3GPP, "General Packet Radio System (GPRS) Tunnelling Protocol User Plane (GTPv1-U)", Technical Specification TS 29.281.

- [GTPv2-C] 3GPP, "Evolved General Packet Radio Service (GPRS) Tunneling Protocol for Control plane (GTPv2-C)", Technical Specification TS 29.274.
- [I-D.ietf-intarea-gue]
Herbert, T., Yong, L., and O. Zia, "Generic UDP Encapsulation", draft-ietf-intarea-gue-09 (work in progress), October 2019.
- [I-D.ietf-trill-ecn-support]
Eastlake, D. E. and B. Briscoe, "TRILL (Transparent Interconnection of Lots of Links): ECN (Explicit Congestion Notification) Support", draft-ietf-trill-ecn-support-07 (work in progress), February 2018.
- [I-D.ietf-tsvwg-ecn-l4s-id]
Schepper, K. D. and B. Briscoe, "Explicit Congestion Notification (ECN) Protocol for Ultra-Low Queuing Delay (L4S)", draft-ietf-tsvwg-ecn-l4s-id-14 (work in progress), March 2021.
- [I-D.ietf-tsvwg-rfc6040update-shim]
Briscoe, B., "Propagating Explicit Congestion Notification Across IP Tunnel Headers Separated by a Shim", draft-ietf-tsvwg-rfc6040update-shim-13 (work in progress), March 2021.
- [IEEE802.1Q]
IEEE, "IEEE Standard for Local and Metropolitan Area Networks--Virtual Bridged Local Area Networks--Amendment 6: Provider Backbone Bridges", IEEE Std 802.1Q-2018, July 2018, <<https://ieeexplore.ieee.org/document/8403927>>.
- [ITU-T.I.371]
ITU-T, "Traffic Control and Congestion Control in B-ISDN", ITU-T Rec. I.371 (03/04), March 2004, <<http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5454061>>.
- [Leiserson85]
Leiserson, C., "Fat-trees: universal networks for hardware-efficient supercomputing", IEEE Transactions on Computers 34(10):892-901, October 1985.
- [LTE-RA] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2", Technical Specification TS 36.300.

- [RFC2003] Perkins, C., "IP Encapsulation within IP", RFC 2003, DOI 10.17487/RFC2003, October 1996, <<https://www.rfc-editor.org/info/rfc2003>>.
- [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>>.
- [RFC2637] Hamzeh, K., Pall, G., Verthein, W., Taarud, J., Little, W., and G. Zorn, "Point-to-Point Tunneling Protocol (PPTP)", RFC 2637, DOI 10.17487/RFC2637, July 1999, <<https://www.rfc-editor.org/info/rfc2637>>.
- [RFC2661] Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G., and B. Palter, "Layer Two Tunneling Protocol "L2TP"", RFC 2661, DOI 10.17487/RFC2661, August 1999, <<https://www.rfc-editor.org/info/rfc2661>>.
- [RFC2784] Farinacci, D., Li, T., Hanks, S., Meyer, D., and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 2784, DOI 10.17487/RFC2784, March 2000, <<https://www.rfc-editor.org/info/rfc2784>>.
- [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>>.
- [RFC2983] Black, D., "Differentiated Services and Tunnels", RFC 2983, DOI 10.17487/RFC2983, October 2000, <<https://www.rfc-editor.org/info/rfc2983>>.
- [RFC3931] Lau, J., Ed., Townsley, M., Ed., and I. Goyret, Ed., "Layer Two Tunneling Protocol - Version 3 (L2TPv3)", RFC 3931, DOI 10.17487/RFC3931, March 2005, <<https://www.rfc-editor.org/info/rfc3931>>.
- [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>>.
- [RFC4380] Huitema, C., "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)", RFC 4380, DOI 10.17487/RFC4380, February 2006, <<https://www.rfc-editor.org/info/rfc4380>>.

- [RFC5415] Calhoun, P., Ed., Montemurro, M., Ed., and D. Stanley, Ed., "Control And Provisioning of Wireless Access Points (CAPWAP) Protocol Specification", RFC 5415, DOI 10.17487/RFC5415, March 2009, <<https://www.rfc-editor.org/info/rfc5415>>.
- [RFC6633] Gont, F., "Deprecation of ICMP Source Quench Messages", RFC 6633, DOI 10.17487/RFC6633, May 2012, <<https://www.rfc-editor.org/info/rfc6633>>.
- [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>>.
- [RFC6830] Farinacci, D., Fuller, V., Meyer, D., and D. Lewis, "The Locator/ID Separation Protocol (LISP)", RFC 6830, DOI 10.17487/RFC6830, January 2013, <<https://www.rfc-editor.org/info/rfc6830>>.
- [RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, DOI 10.17487/RFC7323, September 2014, <<https://www.rfc-editor.org/info/rfc7323>>.
- [RFC7348] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", RFC 7348, DOI 10.17487/RFC7348, August 2014, <<https://www.rfc-editor.org/info/rfc7348>>.
- [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>>.
- [RFC7637] Garg, P., Ed. and Y. Wang, Ed., "NVGRE: Network Virtualization Using Generic Routing Encapsulation", RFC 7637, DOI 10.17487/RFC7637, September 2015, <<https://www.rfc-editor.org/info/rfc7637>>.
- [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>>.

- [RFC7780] Eastlake 3rd, D., Zhang, M., Perlman, R., Banerjee, A., Ghanwani, A., and S. Gupta, "Transparent Interconnection of Lots of Links (TRILL): Clarifications, Corrections, and Updates", RFC 7780, DOI 10.17487/RFC7780, February 2016, <<https://www.rfc-editor.org/info/rfc7780>>.
- [RFC8084] Fairhurst, G., "Network Transport Circuit Breakers", BCP 208, RFC 8084, DOI 10.17487/RFC8084, March 2017, <<https://www.rfc-editor.org/info/rfc8084>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.
- [RFC8926] Gross, J., Ed., Ganga, I., Ed., and T. Sridhar, Ed., "Geneve: Generic Network Virtualization Encapsulation", RFC 8926, DOI 10.17487/RFC8926, November 2020, <<https://www.rfc-editor.org/info/rfc8926>>.
- [UTRAN] 3GPP, "UTRAN Overall Description", Technical Specification TS 25.401.

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

From ietf-12 to ietf-13

* Following 3rd tsvwg WGLC:

- + Formalized update to RFC 3819 in its own subsection (1.1) and referred to it in the abstract
- + Scope: Clarified that the specification of alternative ECN semantics using ECT(1) was not in RFC 4774, but rather in RFC 8311, and that the problem with using a DSCP to indicate alternative semantics has issues at domain boundaries as well as tunnels.
- + Terminology: tightened up definitions of ECN-PDU and Not-ECN-PDU, and removed definition of Congestion Baseline, given it was only used once.
- + Mentioned QCN where feed-backward is first introduced (S.3), referring forward to where it is discussed more deeply (S.4).
- + Clarified that IS-IS solution to adding ECN support to TRILL was not pursued
- + Completely rewrote the rationale for the guideline about a Standard Congestion Monitoring Baseline, to focus on standardization of the otherwise unknown scenario used, rather than the relative usefulness of the info in each approach
- + Explained the re-framing problem better and added fragmentation as another possible cause of the problem
- + Acknowledged new reviewers
- + Updated references, replaced citations of 802.1Qau and 802.1ah with rolled up 802.1Q, and added citations of Fat trees and Clos Networks
- + Numerous other editorial improvements

From ietf-11 to ietf-12

* Updated references

From ietf-10 to ietf-11

- * Removed short section (was 3) 'Guidelines for All Cases' because it was out of scope, being covered by RFC 4774. Expanded the Scope section (1.2) to explain all this. Explained that the default encap/decap rules already support certain alternative semantics, particularly all three of the alternative semantics for ECT(1): equivalent to ECT(0) , higher severity than ECT(0), and unmarked but implying different marking semantics from ECT(0).
- * Clarified why the QCN example was being given even though not about increment deployment of ECN
- * Pointed to the spoofing issue with feed-backward mode from the Security Considerations section, to aid security review.
- * Removed any ambiguity in the word 'transport' throughout

From ietf-09 to ietf-10

- * Updated section 5.1 on "IP-in-IP tunnels with Shim Headers" to be consistent with updates to draft-ietf-tsvwg-rfc6040update-shim.
- * Removed reference to the ECN nonce, which has been made historic by RFC 8311
- * Removed "Open Issues" Appendix, given all have been addressed.

From ietf-08 to ietf-09

- * Updated para in Intro that listed all the IP-in-IP tunnelling protocols, to instead refer to draft-ietf-tsvwg-rfc6040update-shim
- * Updated section 5.1 on "IP-in-IP tunnels with Shim Headers" to summarize guidance that has evolved as rfc6040update-shim has developed.

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

From ietf-06 to ietf-07:

- * Added the people involved in liaisons to the acknowledgements.

From ietf-05 to ietf-06:

- * Introduction: Added GUE and Geneve as examples of tightly coupled shims between IP headers that cite RFC 6040. And added VXLAN to list of those that do not.
- * Replaced normative text about tightly coupled shims between IP headers, with reference to new draft-ietf-tsvwg-rfc6040update-shim
- * Wire Protocol Design: Indication of ECN Support: Added TRILL as an example of a well-design protocol that does not need an indication of ECN support in the wire protocol.
- * Encapsulation Guidelines: In the case of a Not-ECN-PDU with a CE outer, replaced SHOULD be dropped, with explanations of when SHOULD or MUST are appropriate.
- * Feed-Up-and-Forward Mode: Explained examples more carefully, referred to PDCP and cited UTRAN spec as well as E-UTRAN.
- * Updated references.
- * Marked open issues as resolved, but did not delete Open Issues Appendix (yet).

From ietf-04 to ietf-05:

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

From ietf-03 to ietf-04:

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

From ietf-02 to ietf-03:

- * Updated references, ad cited RFC4774.

From ietf-01 to ietf-02:

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

From ietf-00 to ietf-01: Updated references.

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

From briscoe-03 to 04:

- * Re-arranged the introduction to describe the purpose of the document first before introducing ECN in more depth. And clarified the introduction throughout.
- * Added applicability to 3GPP TS 36.300.

From briscoe-02 to 03:

- * Scope section:
 - + Added dependence on correct propagation of traffic class information
 - + For the feed-backward mode, deemed multicast and anycast out of scope
- * Ensured all guidelines referring to subnet technologies also refer to tunnels and vice versa by adding applicability sentences at the start of sections 4.1, 4.2, 4.3, 4.4, 4.6 and 5.
- * Added Security Considerations on ensuring congestion signal fields are classed as immutable and on using end-to-end congestion signal integrity technologies rather than hop-by-hop.

From briscoe-01 to 02:

- * Added authors: JK & PT
- * Added
 - + Section 4.1 "IP-in-IP Tunnels with Tightly Coupled Shim Headers"
 - + Section 4.5 "Sequences of Similar Tunnels or Subnets"
 - + roadmap at the start of Section 4, given the subsections have become quite fragmented.
 - + Section 9 "Conclusions"

- * Clarified why transports are starting to be able to saturate interior links
- * Under Section 1.1, addressed the question of alternative signal semantics and included multicast & anycast.
- * Under Section 3.1, included a 3GPP example.
- * Section 4.2. "Wire Protocol Design":
 - + Altered guideline 2. to make it clear that it only applies to the immediate subnet egress, not later ones
 - + Added a reminder that it is only necessary to check that ECN propagates at the egress, not whether interior nodes mark ECN
 - + Added example of how QCN uses 802.1p to indicate support for QCN.
- * Added references to Appendix C of RFC6040, about monitoring the amount of congestion signals introduced within a tunnel
- * Appendix A: Added more issues to be addressed, including plan to produce a standards track update to IP-in-IP tunnel protocols.
- * Updated acks and references

From briscoe-00 to 01:

- * Intended status: BCP (was Informational) & updates 3819 added.
- * Briefer Introduction: Introductory para justifying benefits of ECN. Moved all but a brief enumeration of modes of operation to their own new section (from both Intro & Scope). Introduced incr. deployment as most tricky part.
- * Tightened & added to terminology section
- * Structured with Modes of Operation, then Guidelines section for each mode.
- * Tightened up guideline text to remove vagueness / passive voice / ambiguity and highlight main guidelines as numbered items.
- * Added Outstanding Document Issues Appendix

* Updated references

Authors' Addresses

Bob Briscoe
Independent
UK

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

John Kaippallimalil
Futurewei
5700 Tennyson Parkway, Suite 600
Plano, Texas 75024
USA

EMail: kjohn@futurewei.com

Transport Area Working Group
Internet-Draft
Updates: 3168, 4341, 4342, 5622, 6679
(if approved)
Intended status: Standards Track
Expires: May 17, 2018

D. Black
Dell EMC
November 13, 2017

Relaxing Restrictions on Explicit Congestion Notification (ECN)
Experimentation
draft-ietf-tsvwg-ecn-experimentation-08

Abstract

This memo updates RFC 3168, which specifies Explicit Congestion Notification (ECN) as an alternative to packet drops for indicating network congestion to endpoints. It relaxes restrictions in RFC 3168 that hinder experimentation towards benefits beyond just removal of loss. This memo summarizes the anticipated areas of experimentation and updates RFC 3168 to enable experimentation in these areas. An Experimental RFC in the IETF document stream is required to take advantage of any of these enabling updates. In addition, this memo makes related updates to the ECN specifications for RTP in RFC 6679 and for DCCP in RFC 4341, RFC 4342 and RFC 5622. This memo also records the conclusion of the ECN nonce experiment in RFC 3540, and provides the rationale for reclassification of RFC 3540 as Historic; this reclassification enables new experimental use of the ECT(1) codepoint.

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 17, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<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
1.1. ECN Terminology	3
1.2. Requirements Language	4
2. ECN Experimentation: Overview	4
2.1. Effective Congestion Control is Required	5
2.2. Network Considerations for ECN Experimentation	5
2.3. Operational and Management Considerations	6
3. ECN Nonce and RFC 3540	7
4. Updates to RFC 3168	8
4.1. Congestion Response Differences	8
4.2. Congestion Marking Differences	9
4.3. TCP Control Packets and Retransmissions	12
5. ECN for RTP Updates to RFC 6679	13
6. ECN for DCCP Updates to RFCs 4341, 4342 and 5622	15
7. Acknowledgements	15
8. IANA Considerations	16
9. Security Considerations	16
10. References	16
10.1. Normative References	16

10.2. Informative References	17
Author's Address	21

1. Introduction

This memo updates RFC 3168 [RFC3168] which specifies Explicit Congestion Notification (ECN) as an alternative to packet drops for indicating network congestion to endpoints. It relaxes restrictions in RFC 3168 that hinder experimentation towards benefits beyond just removal of loss. This memo summarizes the proposed areas of experimentation and updates RFC 3168 to enable experimentation in these areas. An Experimental RFC in the IETF document stream [RFC4844] is required to take advantage of any of these enabling updates. Putting all of these updates into a single document enables experimentation to proceed without requiring a standards process exception for each Experimental RFC that needs changes to RFC 3168, a Proposed Standard RFC.

There is no need for this memo to update RFC 3168 to simplify standardization of protocols and mechanisms that are documented in Standards Track RFCs, as any Standards Track RFC can update RFC 3168 directly without either relying on updates in this memo or using a standards process exception.

In addition, this memo makes related updates to the ECN specification for RTP [RFC6679] and for three DCCP profiles ([RFC4341], [RFC4342] and [RFC5622]) for the same reason. Each experiment is still required to be documented in one or more separate RFCs, but use of Experimental RFCs for this purpose does not require a process exception to modify any of these Proposed Standard RFCs when the modification falls within the bounds established by this memo (RFC 5622 is an Experimental RFC; it is modified by this memo for consistency with modifications to the other two DCCP RFCs).

Some of the anticipated experimentation includes use of the ECT(1) codepoint that was dedicated to the ECN nonce experiment in RFC 3540 [RFC3540]. This memo records the conclusion of the ECN nonce experiment and provides the explanation for reclassification of RFC 3540 as Historic in order to enable new experimental use of the ECT(1) codepoint.

1.1. ECN Terminology

ECT: ECN-Capable Transport. One of the two codepoints ECT(0) or ECT(1) in the ECN field [RFC3168] of the IP header (v4 or v6). An ECN-capable sender sets one of these to indicate that both transport end-points support ECN.

Not-ECT: The ECN codepoint set by senders that indicates that the transport is not ECN-capable.

CE: Congestion Experienced. The ECN codepoint that an intermediate node sets to indicate congestion. A node sets an increasing proportion of ECT packets to CE as the level of congestion increases.

1.2. 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 RFC 2119 [RFC2119].

2. ECN Experimentation: Overview

Three areas of ECN experimentation are covered by this memo; the cited Internet-Drafts should be consulted for the detailed goals and rationale of each proposed experiment:

Congestion Response Differences: An ECN congestion indication communicates a higher likelihood than a dropped packet that a short queue exists at the network bottleneck node [I-D.ietf-tcpm-alternativebackoff-ecn]. This difference suggests that for congestion indicated by ECN, a different sender congestion response (e.g., sender backs off by a smaller amount) may be appropriate by comparison to the sender response to congestion indicated by loss. Two examples of proposed sender congestion response changes are described in [I-D.ietf-tcpm-alternativebackoff-ecn] and [I-D.ietf-tsvwg-ecn-l4s-id] - the proposal in the latter draft couples the sender congestion response change to Congestion Marking Differences functionality (see next paragraph). These changes are at variance with RFC 3168's requirement that a sender's congestion control response to ECN congestion indications be the same as to drops. IETF approval, e.g., via an Experimental RFC in the IETF document stream, is required for any sender congestion response used in this area of experimentation. See Section 4.1 for further discussion.

Congestion Marking Differences: Congestion marking at network nodes can be configured to maintain very shallow queues in conjunction with a different sender response to congestion indications (CE marks), e.g., as proposed in [I-D.ietf-tsvwg-ecn-l4s-id]. The traffic involved needs to be identified by the senders to the network nodes in order to avoid damage to other network traffic whose senders do not expect the more frequent congestion marking used to maintain very shallow queues. Use of different ECN

codepoints, specifically ECT(0) and ECT(1), is a promising means of traffic identification for this purpose, but that technique is at variance with RFC 3168's requirement that ECT(0)-marked traffic and ECT(1)-marked traffic not receive different treatment in the network. IETF approval, e.g., via an Experimental RFC in the IETF document stream, is required for any differences in congestion marking or sender congestion response used in this area of experimentation. See Section 4.2 for further discussion.

TCP Control Packets and Retransmissions: RFC 3168 limits the use of ECN with TCP to data packets, excluding retransmissions. With the successful deployment of ECN in large portions of the Internet, there is interest in extending the benefits of ECN to TCP control packets (e.g., SYNs) and retransmitted packets, e.g., as proposed in [I-D.bagnulo-tcpm-generalized-ecn]. This is at variance with RFC 3168's prohibition of use of ECN for TCP control packets and retransmitted packets. See Section 4.3 for further discussion.

The scope of this memo is limited to these three areas of experimentation. This memo expresses no view on the likely outcomes of the proposed experiments and does not specify the experiments in detail. Additional experiments in these areas are possible, e.g., on use of ECN to support deployment of a protocol similar to DCTCP [I-D.ietf-tcpm-dctcp] beyond DCTCP's current applicability that is limited to data center environments. The purpose of this memo is to remove constraints in standards track RFCs that stand in the way of these areas of experimentation.

2.1. Effective Congestion Control is Required

Congestion control remains an important aspect of the Internet architecture [RFC2914]. Any Experimental RFC in the IETF document stream that takes advantage of this memo's updates to any RFC is required to discuss the congestion control implications of the experiment(s) in order to provide assurance that deployment of the experiment(s) does not pose a congestion-based threat to the operation of the Internet.

2.2. Network Considerations for ECN Experimentation

ECN functionality [RFC3168] is becoming widely deployed in the Internet and is being designed into additional protocols such as TRILL [I-D.ietf-trill-ecn-support]. ECN experiments are expected to coexist with deployed ECN functionality, with the responsibility for that coexistence falling primarily upon designers of experimental changes to ECN. In addition, protocol designers and implementers, as well as network operators, may desire to anticipate and/or support

ECN experiments. The following guidelines will help avoid conflicts with the areas of ECN experimentation enabled by this memo:

1. RFC 3168's forwarding behavior remains the preferred approach for routers that are not involved in ECN experiments, in particular continuing to treat the ECT(0) and ECT(1) codepoints as equivalent, as specified in Section 4.2 below.
2. Network nodes that forward packets SHOULD NOT assume that the ECN CE codepoint indicates that the packet would have been dropped if ECN were not in use. This is because Congestion Response Differences experiments employ different congestion responses to dropped packets by comparison to receipt of CE-marked packets (see Section 4.1 below), so CE-marked packets SHOULD NOT be arbitrarily dropped. A corresponding difference in congestion responses already occurs when the ECN field is used for Pre-Congestion Notification (PCN) [RFC6660].
3. A network node MUST NOT originate traffic marked with ECT(1) unless the network node is participating in a Congestion Marking Differences experiment that uses ECT(1), as specified in Section 4.2 below.

Some ECN experiments use ECN with packets where it has not been used previously, specifically TCP control packets and retransmissions, see Section 4.3 below, and in particular its new requirements for middlebox behavior. In general, any system or protocol that inspects or monitors network traffic SHOULD be prepared to encounter ECN usage on packets and traffic that currently do not use ECN.

ECN field handling requirements for tunnel encapsulation and decapsulation are specified in [RFC6040] which is in the process of being updated by [I-D.ietf-tsvwg-rfc6040update-shim]. Related guidance for encapsulations whose outer headers are not IP headers can be found in [I-D.ietf-tsvwg-ecn-encap-guidelines]. These requirements and guidance apply to all traffic, including traffic that is part of any ECN experiment.

2.3. Operational and Management Considerations

Changes in network traffic behavior that result from ECN experimentation are likely to impact network operations and management. Designers of ECN experiments are expected to anticipate possible impacts and consider how they may be dealt with. Specific topics to consider include possible network management changes or extensions, monitoring of the experimental deployment, collection of data for evaluation of the experiment and possible interactions with

other protocols, particularly protocols that encapsulate network traffic.

For further discussion, see [RFC5706]; the questions in Appendix A of RFC 5706 provide a concise survey of some important aspects to consider.

3. ECN Nonce and RFC 3540

As specified in RFC 3168, ECN uses two ECN Capable Transport (ECT) codepoints to indicate that a packet supports ECN, ECT(0) and ECT(1). RFC 3168 assigned the second codepoint, ECT(1), to support ECN nonce functionality that discourages receivers from exploiting ECN to improve their throughput at the expense of other network users. That ECN nonce functionality is fully specified in Experimental RFC 3540 [RFC3540]. This section explains why RFC 3540 is being reclassified as Historic and makes associated updates to RFC 3168.

While the ECN nonce works as specified, and has been deployed in limited environments, widespread usage in the Internet has not materialized. A study of the ECN behaviour of the top one million web servers using 2014 data [Trammell15] found that after ECN was negotiated, none of the 581,711 IPv4 servers tested were using both ECT codepoints, which would have been a possible sign of ECN nonce usage. Of the 17,028 IPv6 servers tested, 4 set both ECT(0) and ECT(1) on data packets. This might have been evidence of use of the ECN nonce by these 4 servers, but might equally have been due to erroneous re-marking of the ECN field by a middlebox or router.

With the emergence of new experimental functionality that depends on use of the ECT(1) codepoint for other purposes, continuing to reserve that codepoint for the ECN nonce experiment is no longer justified. In addition, other approaches to discouraging receivers from exploiting ECN have emerged, see Appendix B.1 of [I-D.ietf-tsvwg-ecn-l4s-id]. Therefore, in support of ECN experimentation with the ECT(1) codepoint, this memo:

- o Declares that the ECN nonce experiment [RFC3540] has concluded, and notes the absence of widespread deployment.
- o Updates RFC 3168 [RFC3168] to remove discussion of the ECN nonce and use of ECT(1) for that nonce.

The four primary updates to RFC 3168 that remove discussion of the ECN nonce and use of ECT(1) for that nonce are:

1. Remove the paragraph in Section 5 that immediately follows Figure 1; this paragraph discusses the ECN nonce as the motivation for two ECT codepoints.
2. Remove Section 11.2 "A Discussion of the ECN nonce." in its entirety.
3. Remove the last paragraph of Section 12, which states that ECT(1) may be used as part of the implementation of the ECN nonce.
4. Remove the first two paragraphs of Section 20.2, which discuss the ECN nonce and alternatives. No changes are made to the rest of Section 20.2, which discusses alternative uses for the fourth ECN codepoint.

In addition, other less substantive RFC 3168 changes are required to remove all other mentions of the ECN nonce and to remove implications that ECT(1) is intended for use by the ECN nonce; these specific text updates are omitted for brevity.

4. Updates to RFC 3168

The following subsections specify updates to RFC 3168 to enable the three areas of experimentation summarized in Section 2.

4.1. Congestion Response Differences

RFC 3168 specifies that senders respond identically to packet drops and ECN congestion indications. ECN congestion indications are predominately originated by Active Queue Management (AQM) mechanisms in intermediate buffers. AQM mechanisms are usually configured to maintain shorter queue lengths than non-AQM based mechanisms, particularly non-AQM drop-based mechanisms such as tail-drop, as AQM mechanisms indicate congestion before the queue overflows. While the occurrence of loss does not easily enable the receiver to determine if AQM is used, the receipt of an ECN Congestion Experienced (CE) mark conveys a strong likelihood that AQM was used to manage the bottleneck queue. Hence an ECN congestion indication communicates a higher likelihood than a dropped packet that a short queue exists at the network bottleneck node [I-D.ietf-tcpm-alternativebackoff-ecn]. This difference suggests that for congestion indicated by ECN, a different sender congestion response (e.g., sender backs off by a smaller amount) may be appropriate by comparison to the sender response to congestion indicated by loss. However, section 5 of RFC 3168 specifies that:

Upon the receipt by an ECN-Capable transport of a single CE packet, the congestion control algorithms followed at the end-

systems MUST be essentially the same as the congestion control response to a **single** dropped packet.

This memo updates this RFC 3168 text to allow the congestion control response (including the TCP Sender's congestion control response) to a CE-marked packet to differ from the response to a dropped packet, provided that the changes from RFC 3168 are documented in an Experimental RFC in the IETF document stream. The specific change to RFC 3168 is to insert the words "unless otherwise specified by an Experimental RFC in the IETF document stream" at the end of the sentence quoted above.

RFC 4774 [RFC4774] quotes the above text from RFC 3168 as background, but does not impose requirements based on that text. Therefore no update to RFC 4774 is required to enable this area of experimentation.

Section 6.1.2 of RFC 3168 specifies that:

If the sender receives an ECN-Echo (ECE) ACK packet (that is, an ACK packet with the ECN-Echo flag set in the TCP header), then the sender knows that congestion was encountered in the network on the path from the sender to the receiver. The indication of congestion should be treated just as a congestion loss in non-ECN-Capable TCP. That is, the TCP source halves the congestion window "cwnd" and reduces the slow start threshold "ssthresh".

This memo also updates this RFC 3168 text to allow the congestion control response (including the TCP Sender's congestion control response) to a CE-marked packet to differ from the response to a dropped packet, provided that the changes from RFC 3168 are documented in an Experimental RFC in the IETF document stream. The specific change to RFC 3168 is to insert the words "Unless otherwise specified by an Experimental RFC in the IETF document stream" at the beginning of the second sentence quoted above.

4.2. Congestion Marking Differences

Taken to its limit, an AQM algorithm that uses ECN congestion indications can be configured to maintain very shallow queues, thereby reducing network latency by comparison to maintaining a larger queue. Significantly more aggressive sender responses to ECN are needed to make effective use of such very shallow queues; Datacenter TCP (DCTCP) [I-D.ietf-tcpm-dctcp] provides an example. In this case, separate network node treatments are essential, both to prevent the aggressive low latency traffic from starving conventional traffic (if present) and to prevent any conventional traffic disruption to any lower latency service that uses the very shallow

queues. Use of different ECN codepoints is a promising means of identifying these two classes of traffic to network nodes, and hence this area of experimentation is based on the use of the ECT(1) codepoint to request ECN congestion marking behavior in the network that differs from ECT(0). It is essential that any such change in ECN congestion marking behavior be counterbalanced by use of a different IETF-approved congestion response to CE marks at the sender, e.g., as proposed in [I-D.ietf-tsvwg-ecn-l4s-id].

Section 5 of RFC 3168 specifies that:

Routers treat the ECT(0) and ECT(1) codepoints as equivalent.

This memo updates RFC 3168 to allow routers to treat the ECT(0) and ECT(1) codepoints differently, provided that the changes from RFC 3168 are documented in an Experimental RFC in the IETF document stream. The specific change to RFC 3168 is to insert the words "unless otherwise specified by an Experimental RFC in the IETF document stream" at the end of the above sentence.

When an AQM is configured to use ECN congestion indications to maintain a very shallow queue, congestion indications are marked on packets that would not have been dropped if ECN was not in use. Section 5 of RFC 3168 specifies that:

For a router, the CE codepoint of an ECN-Capable packet SHOULD only be set if the router would otherwise have dropped the packet as an indication of congestion to the end nodes. When the router's buffer is not yet full and the router is prepared to drop a packet to inform end nodes of incipient congestion, the router should first check to see if the ECT codepoint is set in that packet's IP header. If so, then instead of dropping the packet, the router MAY instead set the CE codepoint in the IP header.

This memo updates RFC 3168 to allow congestion indications that are not equivalent to drops, provided that the changes from RFC 3168 are documented in an Experimental RFC in the IETF document stream. The specific change is to change "For a router," to "Unless otherwise specified by an Experimental RFC in the IETF document stream" at the beginning of the first sentence of the above paragraph.

A larger update to RFC 3168 is necessary to enable sender usage of ECT(1) to request network congestion marking behavior that maintains very shallow queues at network nodes. When using loss as a congestion signal, the number of signals provided should be reduced to a minimum and hence only presence or absence of congestion is communicated. In contrast, ECN can provide a richer signal, e.g., to indicate the current level of congestion, without the disadvantage of

a larger number of packet losses. A proposed experiment in this area, Low Latency Low Loss Scalable throughput (L4S) [I-D.ietf-tsvwg-ecn-l4s-id] significantly increases the CE marking probability for ECT(1)-marked traffic in a fashion that would interact badly with existing sender congestion response functionality because that functionality assumes that the network marks ECT packets as frequently as it would drop Not-ECT packets. If network traffic that uses such a conventional sender congestion response were to encounter L4S's increased marking probability (and hence rate) at a network bottleneck queue, the resulting traffic throughput is likely to be much less than intended for the level of congestion at the bottleneck queue.

This memo updates RFC 3168 to remove that interaction for ECT(1). The specific update to Section 5 of RFC 3168 is to replace the following two paragraphs:

Senders are free to use either the ECT(0) or the ECT(1) codepoint to indicate ECT, on a packet-by-packet basis.

The use of both the two codepoints for ECT, ECT(0) and ECT(1), is motivated primarily by the desire to allow mechanisms for the data sender to verify that network elements are not erasing the CE codepoint, and that data receivers are properly reporting to the sender the receipt of packets with the CE codepoint set, as required by the transport protocol. Guidelines for the senders and receivers to differentiate between the ECT(0) and ECT(1) codepoints will be addressed in separate documents, for each transport protocol. In particular, this document does not address mechanisms for TCP end-nodes to differentiate between the ECT(0) and ECT(1) codepoints. Protocols and senders that only require a single ECT codepoint SHOULD use ECT(0).

with this paragraph:

Protocols and senders MUST use the ECT(0) codepoint to indicate ECT unless otherwise specified by an Experimental RFC in the IETF document stream. Protocols and senders MUST NOT use the ECT(1) codepoint to indicate ECT unless otherwise specified by an Experimental RFC in the IETF document stream. Guidelines for senders and receivers to differentiate between the ECT(0) and ECT(1) codepoints will be addressed in separate documents, for each transport protocol. In particular, this document does not address mechanisms for TCP end-nodes to differentiate between the ECT(0) and ECT(1) codepoints.

Congestion Marking Differences experiments SHOULD modify the network behavior for ECT(1)-marked traffic rather than ECT(0)-marked traffic

if network behavior for only one ECT codepoint is modified. Congestion Marking Differences experiments MUST NOT modify the network behavior for ECT(0)-marked traffic in a fashion that requires changes to sender congestion response to obtain desired network behavior. If a Congestion Marking Differences experiment modifies the network behavior for ECT(1)-marked traffic, e.g., CE-marking behavior, in a fashion that requires changes to sender congestion response to obtain desired network behavior, then the Experimental RFC in the IETF document stream for that experiment MUST specify:

- o The sender congestion response to CE marking in the network, and
- o Router behavior changes, or the absence thereof, in forwarding CE-marked packets that are part of the experiment.

In addition, this memo updates RFC 3168 to remove discussion of the ECN nonce, as noted in Section 3 above.

4.3. TCP Control Packets and Retransmissions

With the successful use of ECN for traffic in large portions of the Internet, there is interest in extending the benefits of ECN to TCP control packets (e.g., SYNs) and retransmitted packets, e.g., as proposed by ECN++ [I-D.bagnulo-tcpm-generalized-ecn].

RFC 3168 prohibits use of ECN for TCP control packets and retransmitted packets in a number of places:

- o "To ensure the reliable delivery of the congestion indication of the CE codepoint, an ECT codepoint MUST NOT be set in a packet unless the loss of that packet in the network would be detected by the end nodes and interpreted as an indication of congestion." (Section 5.2)
- o "A host MUST NOT set ECT on SYN or SYN-ACK packets." (Section 6.1.1)
- o "pure acknowledgement packets (e.g., packets that do not contain any accompanying data) MUST be sent with the not-ECT codepoint." (Section 6.1.4)
- o "This document specifies ECN-capable TCP implementations MUST NOT set either ECT codepoint (ECT(0) or ECT(1)) in the IP header for retransmitted data packets, and that the TCP data receiver SHOULD ignore the ECN field on arriving data packets that are outside of the receiver's current window." (Section 6.1.5)

- o "the TCP data sender MUST NOT set either an ECT codepoint or the CWR bit on window probe packets." (Section 6.1.6)

This memo updates RFC 3168 to allow the use of ECT codepoints on SYN and SYN-ACK packets, pure acknowledgement packets, window probe packets and retransmissions of packets that were originally sent with an ECT codepoint, provided that the changes from RFC 3168 are documented in an Experimental RFC in the IETF document stream. The specific change to RFC 3168 is to insert the words "unless otherwise specified by an Experimental RFC in the IETF document stream" at the end of each sentence quoted above.

In addition, beyond requiring TCP senders not to set ECT on TCP control packets and retransmitted packets, RFC 3168 is silent on whether it is appropriate for a network element, e.g. a firewall, to discard such a packet as invalid. For this area of ECN experimentation to be useful, middleboxes ought not to do that, therefore RFC 3168 is updated by adding the following text to the end of Section 6.1.1.1 on Middlebox Issues:

Unless otherwise specified by an Experimental RFC in the IETF document stream, middleboxes SHOULD NOT discard TCP control packets and retransmitted TCP packets solely because the ECN field in the IP header does not contain Not-ECT. An exception to this requirement occurs in responding to an attack that uses ECN codepoints other than Not-ECT. For example, as part of the response, it may be appropriate to drop ECT-marked TCP SYN packets with higher probability than TCP SYN packets marked with not-ECT. Any such exceptional discarding of TCP control packets and retransmitted TCP packets in response to an attack MUST NOT be done routinely in the absence of an attack and SHOULD only be done if it is determined that the use of ECN is contributing to the attack.

5. ECN for RTP Updates to RFC 6679

RFC 6679 [RFC6679] specifies use of ECN for RTP traffic; it allows use of both the ECT(0) and ECT(1) codepoints, and provides the following guidance on use of these codepoints in section 7.3.1 :

The sender SHOULD mark packets as ECT(0) unless the receiver expresses a preference for ECT(1) or for a random ECT value using the "ect" parameter in the "a=ecn-capable-rtp:" attribute.

The Congestion Marking Differences area of experimentation increases the potential consequences of using ECT(1) instead of ECT(0), and hence the above guidance is updated by adding the following two sentences:

Random ECT values MUST NOT be used, as that may expose RTP to differences in network treatment of traffic marked with ECT(1) and ECT(0) and differences in associated endpoint congestion responses. In addition, ECT(0) MUST be used unless otherwise specified in an Experimental RFC in the IETF document stream.

Section 7.3.3 of RFC 6679 specifies RTP's response to receipt of CE marked packets as being identical to the response to dropped packets:

The reception of RTP packets with ECN-CE marks in the IP header is a notification that congestion is being experienced. The default reaction on the reception of these ECN-CE-marked packets MUST be to provide the congestion control algorithm with a congestion notification that triggers the algorithm to react as if packet loss had occurred. There should be no difference in congestion response if ECN-CE marks or packet drops are detected.

In support of Congestion Response Differences experimentation, this memo updates this text in a fashion similar to RFC 3168 to allow the RTP congestion control response to a CE-marked packet to differ from the response to a dropped packet, provided that the changes from RFC 6679 are documented in an Experimental RFC in the IETF document stream. The specific change to RFC 6679 is to insert the words "Unless otherwise specified by an Experimental RFC in the IETF document stream" and reformat the last two sentences to be subject to that condition, i.e.:

The reception of RTP packets with ECN-CE marks in the IP header is a notification that congestion is being experienced. Unless otherwise specified by an Experimental RFC in the IETF document stream:

- * The default reaction on the reception of these ECN-CE-marked packets MUST be to provide the congestion control algorithm with a congestion notification that triggers the algorithm to react as if packet loss had occurred.
- * There should be no difference in congestion response if ECN-CE marks or packet drops are detected.

The second sentence of the immediately following paragraph in RFC 6679 requires a related update:

Other reactions to ECN-CE may be specified in the future, following IETF Review. Detailed designs of such additional reactions MUST be specified in a Standards Track RFC and be reviewed to ensure they are safe for deployment under any restrictions specified.

The update is to change "Standards Track RFC" to "Standards Track RFC or Experimental RFC in the IETF document stream" for consistency with the first update.

6. ECN for DCCP Updates to RFCs 4341, 4342 and 5622

The specifications of the three DCCP Congestion Control IDs (CCIDs) 2 [RFC4341], 3 [RFC4342] and 4 [RFC5622] contain broadly the same wording as follows:

each DCCP-Data and DCCP-DataAck packet is sent as ECN Capable with either the ECT(0) or the ECT(1) codepoint set.

This memo updates these sentences in each of the three RFCs as follows:

each DCCP-Data and DCCP-DataAck packet is sent as ECN Capable. Unless otherwise specified by an Experimental RFC in the IETF document stream, such DCCP senders MUST set the ECT(0) codepoint.

In support of Congestion Marking Differences experimentation (as noted in Section 3), this memo also updates all three of these RFCs to remove discussion of the ECN nonce. The specific text updates are omitted for brevity.

7. Acknowledgements

The content of this draft, including the specific portions of RFC 3168 that are updated draws heavily from [I-D.khademi-tsvwg-ecn-response], whose authors are gratefully acknowledged. The authors of the Internet Drafts describing the experiments have motivated the production of this memo - their interest in innovation is welcome and heartily acknowledged. Colin Perkins suggested updating RFC 6679 on RTP and provided guidance on where to make the updates.

The draft has been improved as a result of comments from a number of reviewers, including Ben Campbell, Brian Carpenter, Benoit Claise, Spencer Dawkins, Gorby Fairhurst, Sue Hares, Ingemar Johansson, Naeem Khademi, Mirja Kuehlewind, Karen Nielsen, Hilarie Orman, Eric Rescorla, Adam Roach and Michael Welzl. Bob Briscoe's thorough reviews of multiple versions of this memo resulted in numerous improvements including addition of the updates to the DCCP RFCs.

8. IANA Considerations

To reflect the reclassification of RFC 3540 as Historic, IANA is requested to update the Transmission Control Protocol (TCP) Header Flags registry (<https://www.iana.org/assignments/tcp-header-flags/tcp-header-flags.xhtml#tcp-header-flags-1>) to remove the registration of bit 7 as the NS (Nonce Sum) bit and add an annotation to the registry to state that bit 7 was used by Historic RFC 3540 as the NS (Nonce Sum) bit.

9. Security Considerations

As a process memo that only relaxes restrictions on experimentation, there are no protocol security considerations, as security considerations for any experiments that take advantage of the relaxed restrictions are discussed in the Internet-Drafts that propose the experiments.

However, effective congestion control is crucial to the continued operation of the Internet, and hence this memo places the responsibility for not breaking Internet congestion control on the experiments and the experimenters who propose them. This responsibility includes the requirement to discuss congestion control implications in an IETF document stream Experimental RFC for each experiment, as stated in Section 2.1; review of that discussion by the IETF community and the IESG prior to RFC publication is intended to provide assurance that each experiment does not break Internet congestion control.

See Appendix C.1 of [I-D.ietf-tsvwg-ecn-l4s-id] for discussion of alternatives to the ECN nonce.

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>>.
- [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, <<https://www.rfc-editor.org/info/rfc3168>>.
- [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>>.
- [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>>.
- [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>>.
- [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>>.

10.2. Informative References

- [I-D.bagnulo-tcpm-generalized-ecn]
Bagnulo, M. and B. Briscoe, "ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control Packets", draft-bagnulo-tcpm-generalized-ecn-04 (work in progress), May 2017.
- [I-D.ietf-tcpm-alternativebackoff-ecn]
Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst, "TCP Alternative Backoff with ECN (ABE)", draft-ietf-tcpm-alternativebackoff-ecn-03 (work in progress), October 2017.

- [I-D.ietf-tcpm-dctcp]
Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L.,
and G. Judd, "Datacenter TCP (DCTCP): TCP Congestion
Control for Datacenters", draft-ietf-tcpm-dctcp-10 (work
in progress), August 2017.
- [I-D.ietf-trill-ecn-support]
Eastlake, D. and B. Briscoe, "TRILL: ECN (Explicit
Congestion Notification) Support", draft-ietf-trill-ecn-
support-03 (work in progress), May 2017.
- [I-D.ietf-tsvwg-ecn-encap-guidelines]
Briscoe, B., Kaippallimalil, J., and P. Thaler,
"Guidelines for Adding Congestion Notification to
Protocols that Encapsulate IP", draft-ietf-tsvwg-ecn-
encap-guidelines-09 (work in progress), July 2017.
- [I-D.ietf-tsvwg-ecn-l4s-id]
Schepper, K. and B. Briscoe, "Identifying Modified
Explicit Congestion Notification (ECN) Semantics for
Ultra-Low Queuing Delay", draft-ietf-tsvwg-ecn-l4s-id-01
(work in progress), October 2017.
- [I-D.ietf-tsvwg-rfc6040update-shim]
Briscoe, B., "Propagating Explicit Congestion Notification
Across IP Tunnel Headers Separated by a Shim", draft-ietf-
tsvwg-rfc6040update-shim-05 (work in progress), November
2017.
- [I-D.khademi-tsvwg-ecn-response]
Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst,
"Updating the Explicit Congestion Notification (ECN)
Specification to Allow IETF Experimentation", draft-
khademi-tsvwg-ecn-response-01 (work in progress), July
2016.
- [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>>.
- [RFC4844] Daigle, L., Ed. and Internet Architecture Board, "The RFC
Series and RFC Editor", RFC 4844, DOI 10.17487/RFC4844,
July 2007, <<https://www.rfc-editor.org/info/rfc4844>>.

- [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>>.
- [RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <<https://www.rfc-editor.org/info/rfc6040>>.
- [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>>.
- [Trammell15]
Trammell, B., Kuehlewind, M., Boppart, D., Learmonth, I., Fairhurst, G., and R. Scheffenegger, "Enabling Internet-Wide Deployment of Explicit Congestion Notification".

In Proc Passive & Active Measurement (PAM'15) Conference (2015)

Appendix A. Change History

[To be removed before RFC publication.]

Changes from draft-ietf-tsvwg-ecn-experimentation-00 to -01:

- o Add mention of DCTCP as another protocol that could benefit from ECN experimentation (near end of Section 2).

Changes from draft-ietf-tsvwg-ecn-experimentation-01 to -02:

- o Generalize to describe rationale for areas of experimentation, with less focus on individual experiments
- o Add ECN terminology section
- o Change name of "ECT Differences" experimentation area to "Congestion Marking Differences"
- o Add overlooked RFC 3168 modification to Section 4.1
- o Clarify text for Experimental RFC exception to ECT(1) non-usage requirement

- o Add explanation of exception to "SHOULD NOT drop" requirement in 4.3
- o Rework RFC 3540 status change text to provide rationale for a separate status change document that makes RFC 3540 Historic. Don't obsolete RFC 3540.
- o Significant editorial changes based on reviews by Mirja Kuehlewind, Michael Welzl and Bob Briscoe.

Changes from draft-ietf-tsvwg-ecn-experimentation-02 to -03:

- o Remove change history prior to WG adoption.
- o Update L4S draft reference to reflect TSVWG adoption of draft.
- o Change the "SHOULD" for DCCP sender use of ECT(0) to a "MUST" (overlooked in earlier editing).
- o Other minor edits.

Changes from draft-ietf-tsvwg-ecn-experimentation-03 to -04:

- o Change name of "Generalized ECN" experimentation area to "TCP Control Packets and Retransmissions."
- o Add IANA Considerations text to request removal of the registration of the NS bit in the TCP header.

Changes from draft-ietf-tsvwg-ecn-experimentation-04 to -05:

- o Minor editorial changes from Area Director review

Changes from draft-ietf-tsvwg-ecn-experimentation-05 to -06:

- o Add summary of RFC 3168 changes to remove the ECN nonce, and use lower-case "nonce" instead of "Nonce" to match RFC 3168 usage.
- o Add security considerations sentence to indicate that review of Experimental RFCs prior to publication approval is the means to ensure that congestion control is not broken by experiments.
- o Other minor editorial changes from IETF Last Call

Changes from draft-ietf-tsvwg-ecn-experimentation-06 to -07:

- o Change draft title to make scope clear - this only covers relaxing of restrictions on ECN experimentation.

- o Any Experimental RFC that takes advantage of this memo has to be in the IETF document stream.
- o Added sections 2.2 and 2.3 on considerations for other protocols and O&M, relocated discussion of congestion control requirement to section 2.1 from section 4.4
- o Remove text indicating that ECT(1) may be assigned to L4S - the requirement for an Experimental RFC suffices to ensure that coordination with L4S will occur.
- o Improve explanation of attack response exception to not dropping packets "solely because the ECN field in the IP header does not contain Not-ECT" in Section 4.3
- o Fix L4S draft reference for discussion of ECN Nonce alternatives - it's Appendix C.1, not B.1.
- o Numerous additional editorial changes from IESG Evaluation

Changes from draft-ietf-tsvwg-ecn-experimentation-07 to -08:

- o Edits from another careful review by Bob Briscoe. The primary change is an editorial rewrite of Section 2.2 including changing its name to better reflect its content.

Author's Address

David Black
Dell EMC
176 South Street
Hopkinton, MA 01748
USA

Email: david.black@dell.com

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], [DCTH19]. 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 end-points: 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>.

- [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.1.5. 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 queueing 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 queueing 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.

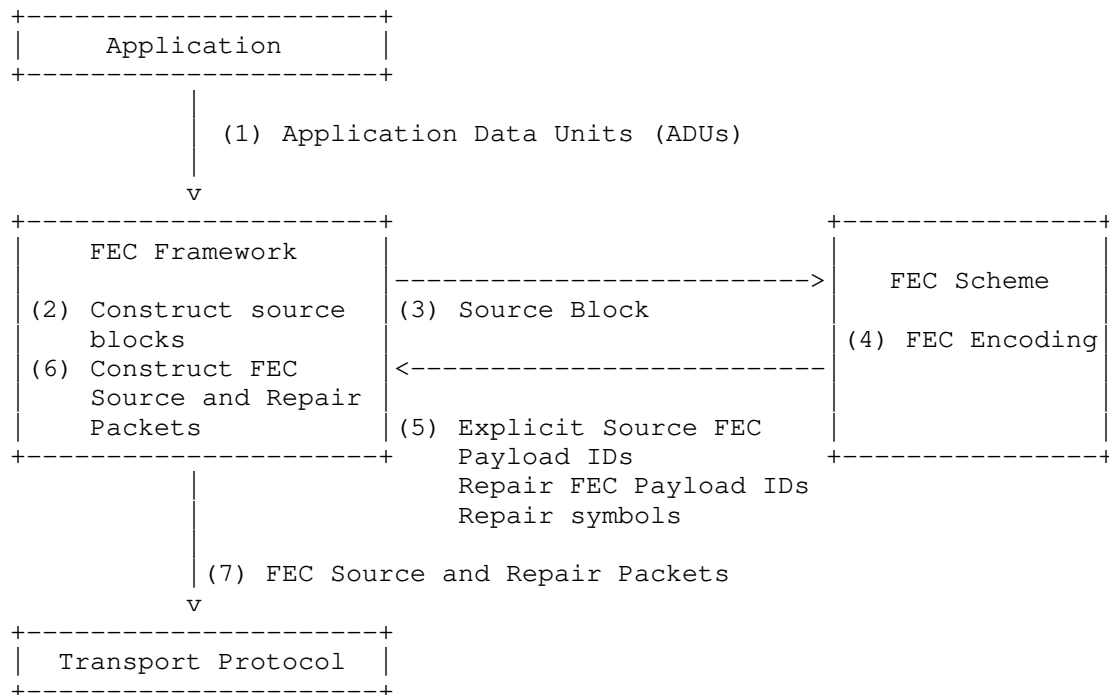


Figure 1: FECFRAME architecture at a sender.

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.

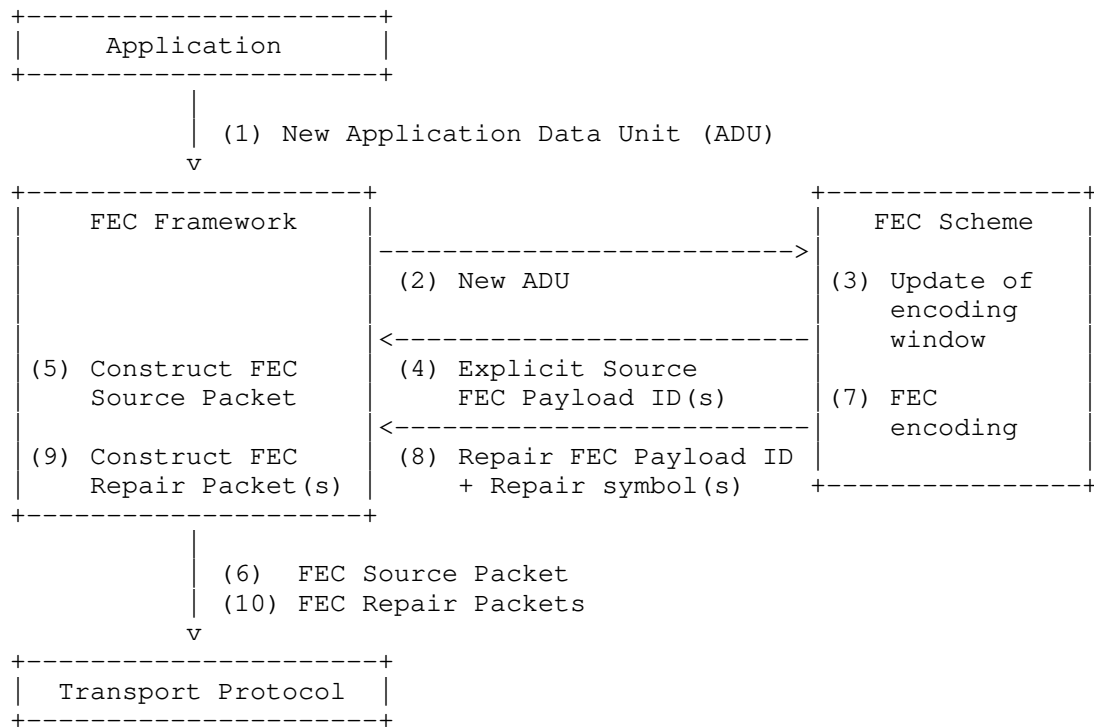


Figure 2: Sender Operation with Sliding Window FEC Codes

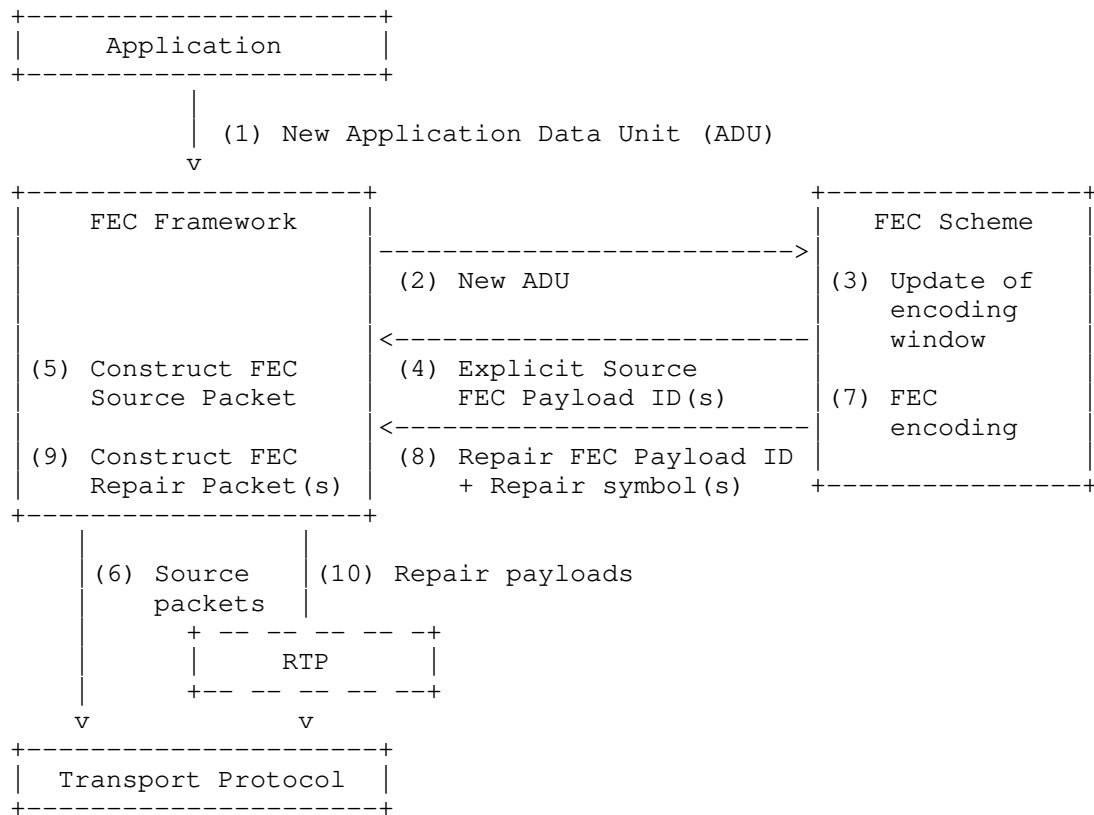


Figure 3: Sender Operation with Sliding Window FEC Codes and RTP Repair Flows

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.

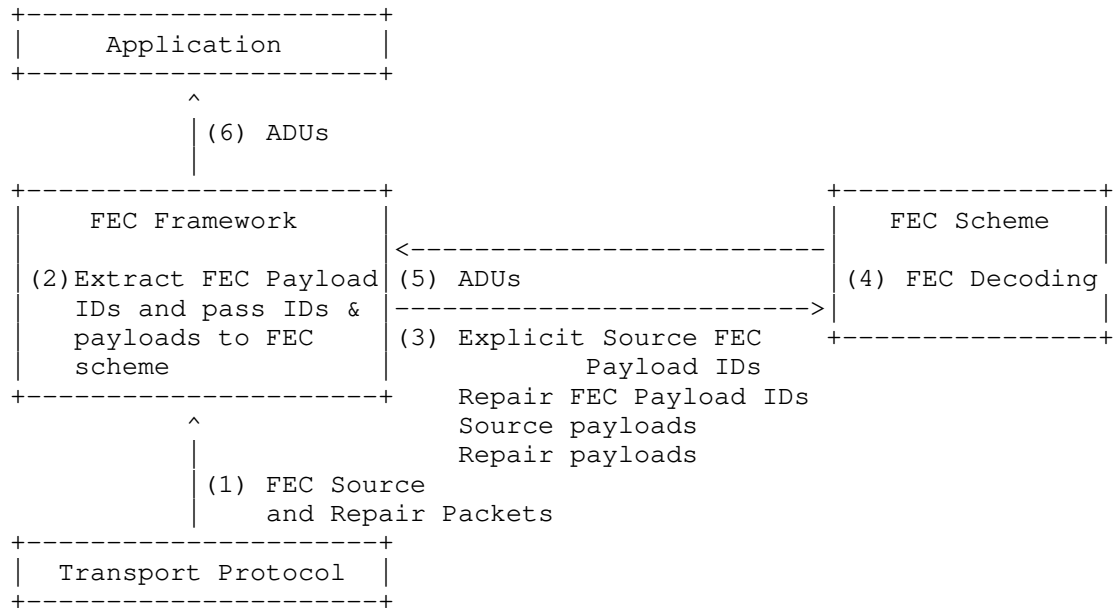


Figure 4: Receiver Operation with Sliding Window FEC Codes

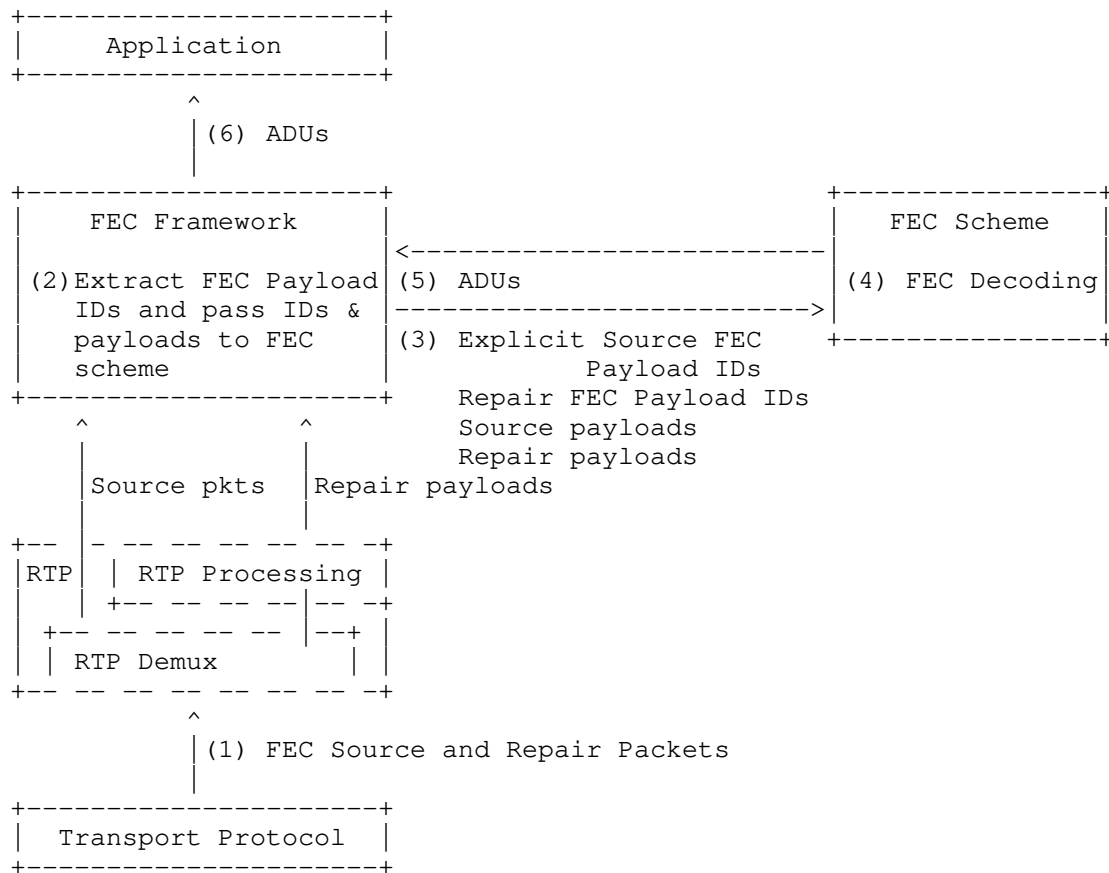


Figure 5: Receiver Operation with Sliding Window FEC Codes and RTP Repair Flows

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
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 queuing 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].

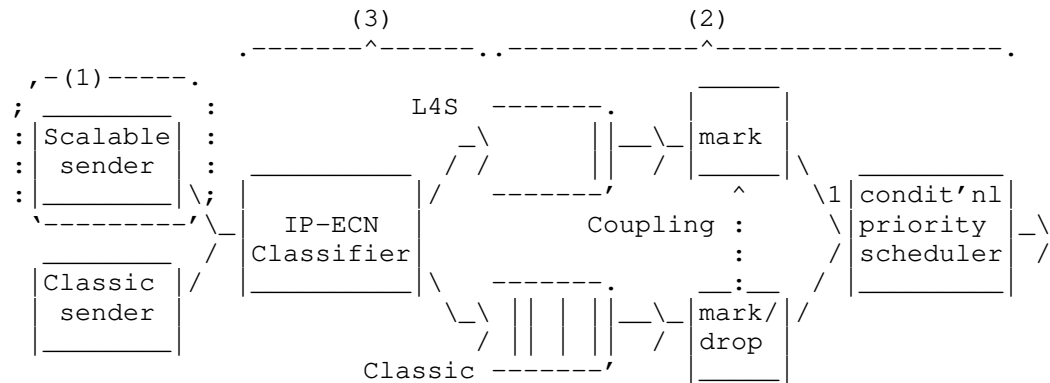


Figure 1: Components of an L4S DualQ Coupled AQM Solution: 1) Scalable Sending Host; 2) Isolation in separate network queues; and 3) Packet Identification Protocol

- 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.)

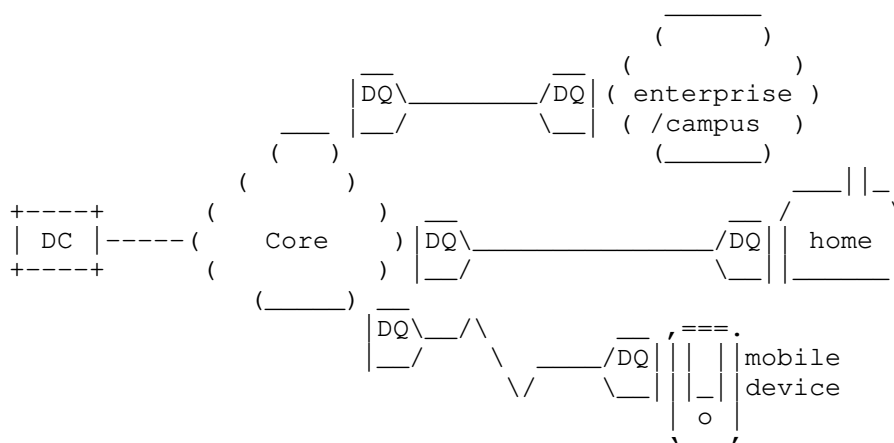


Figure 2: Likely location of DualQ (DQ) Deployments in common access topologies

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 FULLY WORKS UPSTREAM AND DOWNSTREAM		Upgrade DCTCP to TCP Prague

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-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>>.

[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.

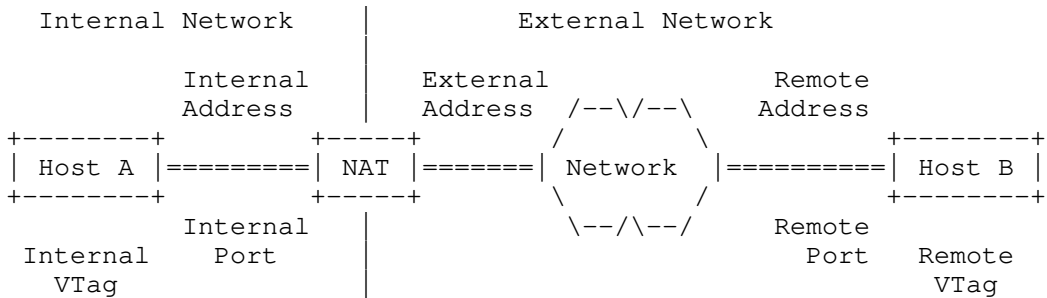


Figure 1: Basic Network Setup

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.

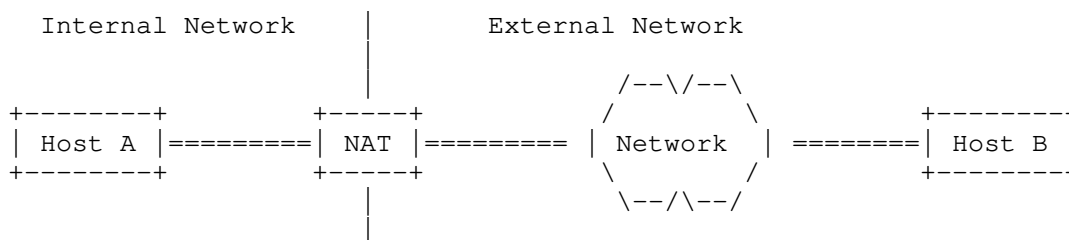


Figure 2: Single NAT Function Scenario

A variation of this case is shown in Figure 3, i.e., multiple NAT functions in the forwarding path between two endpoints.

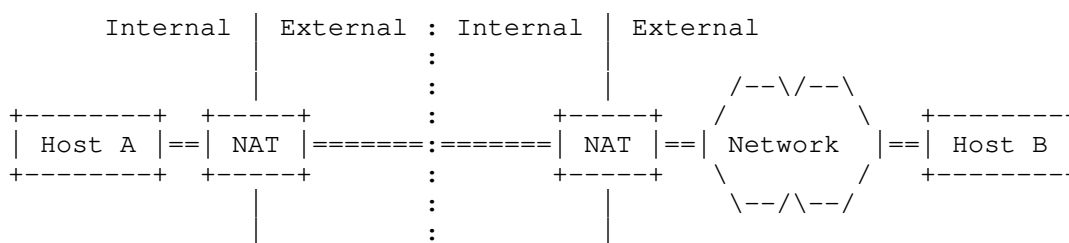


Figure 3: Serial NAT Functions Scenario

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.

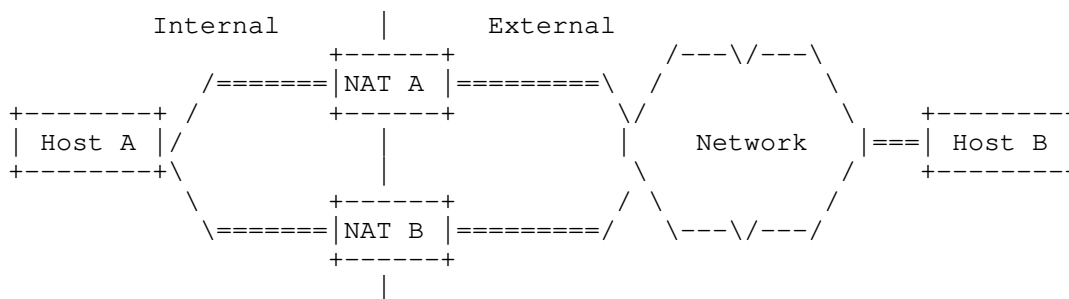


Figure 4: Parallel NAT Functions Scenario

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).

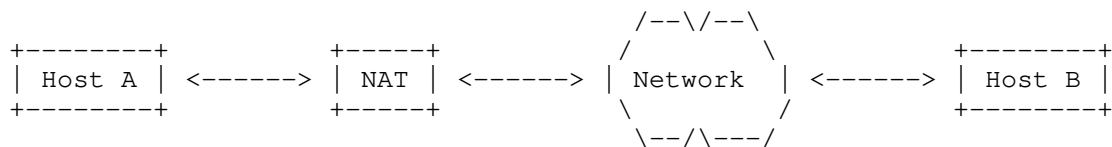
The modification of outgoing SCTP packets sent from an internal host is simple: the source address of the packets has to be replaced with the External-Address. It might also be necessary to establish some state in the NAT function to later handle incoming packets.

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

```

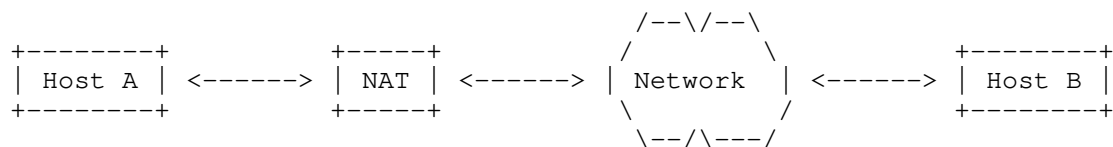
Normally a NAT binding table entry will be created.

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.

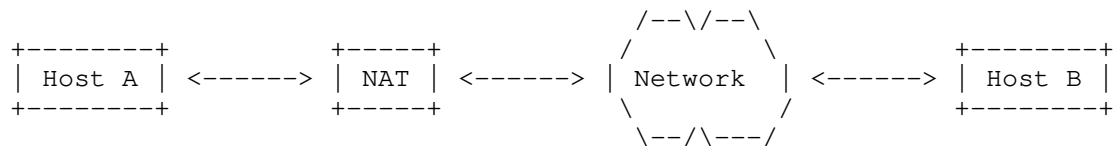


Int-Addr:Int-Port -----> Rem-Addr:Rem-Port
 Rem-VTag

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

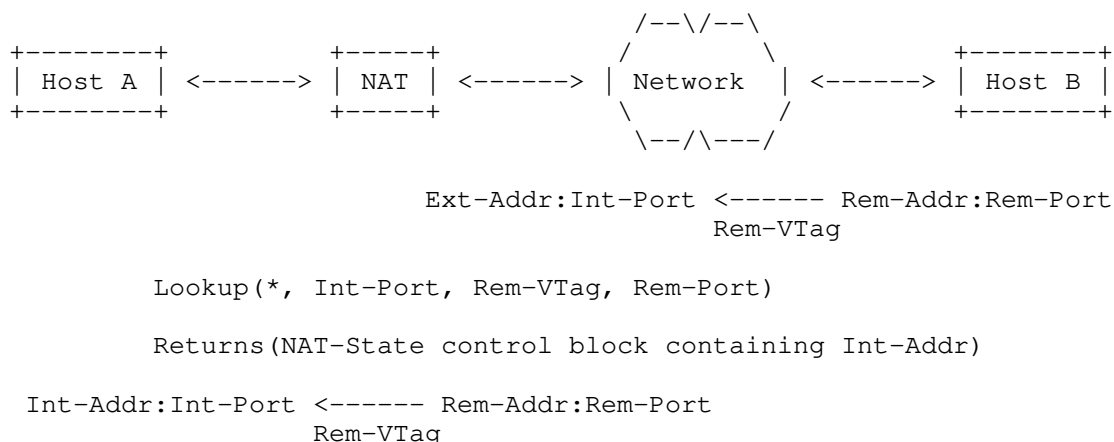
Lookup(Int-VTag, Int-Port, *, Rem-Port)
 Update(*, *, Initiate-Tag, *)

Returns(NAT-State control block containing Int-Addr)

INIT ACK[Initiate-Tag]
 Int-Addr:Int-Port <----- Rem-Addr:Rem-Port
 Int-VTag

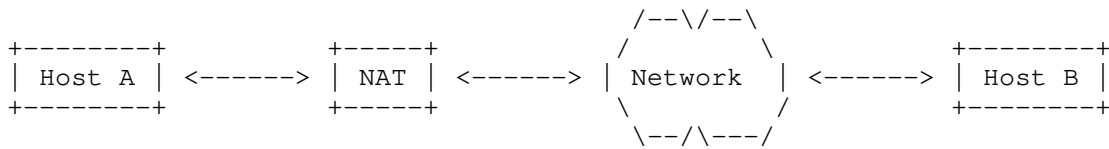
In the case where the Lookup function fails because it does not find an entry, the SCTP packet is dropped. If it succeeds, the Update routine inserts the Remote-VTag (the Initiate-Tag of the INIT ACK chunk) in the NAT-State control block.

The processing of incoming SCTP packets containing an ABORT or SHUTDOWN COMPLETE chunk with the T bit set is illustrated in the following figure.



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

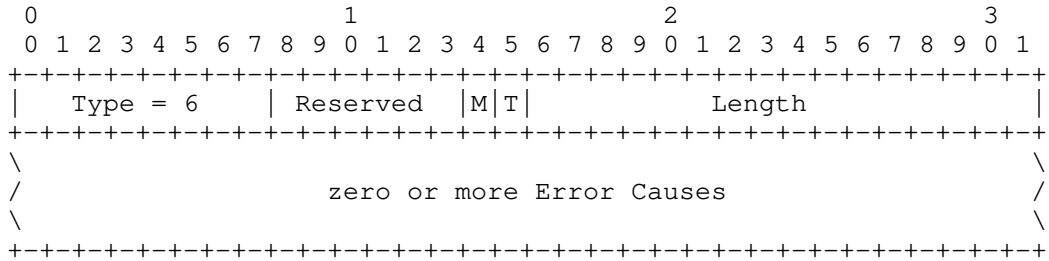
5. Data Formats

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.

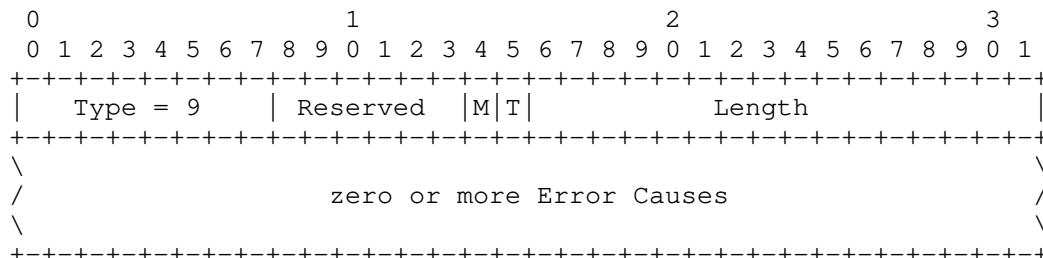
5.1.1. Extended ABORT Chunk



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



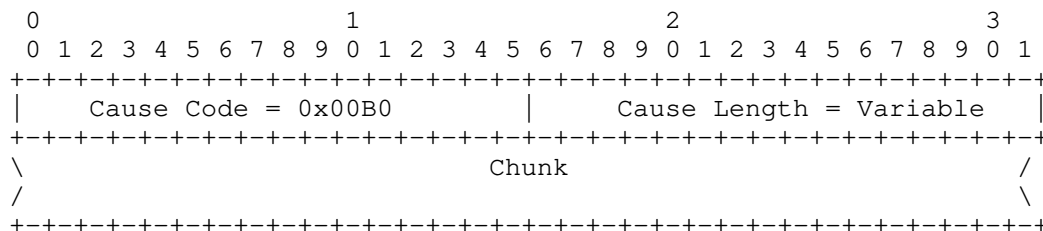
The ERROR chunk defined in [RFC4960] is extended to add the new 'M bit'. The M bit indicates to the receiver of the ERROR chunk that the chunk was not generated by the peer SCTP endpoint, but instead by a middle box.

[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)

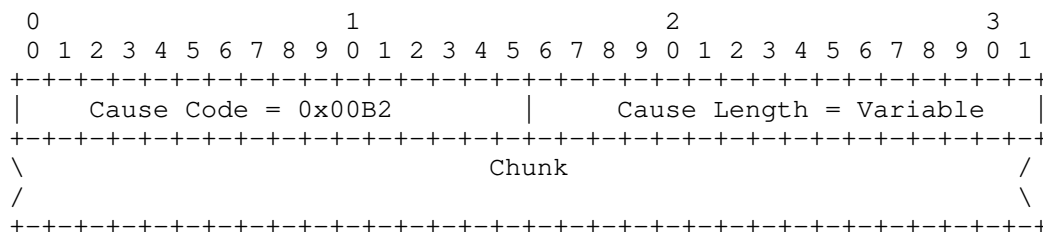
This field holds the length in bytes of the error cause. The value MUST be the length of the Cause-Specific Information plus 4.

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.

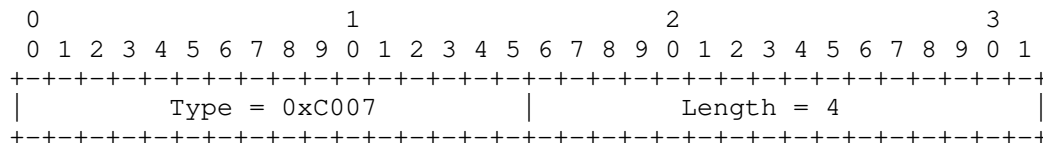
[NOTE to RFC-Editor: Assignment of cause code to be confirmed by IANA.]

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)

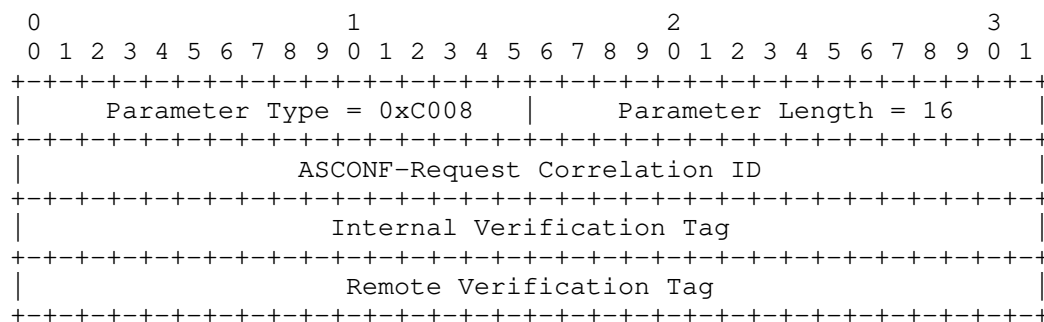
This field holds the length in bytes of the parameter. The value MUST be 4.

[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

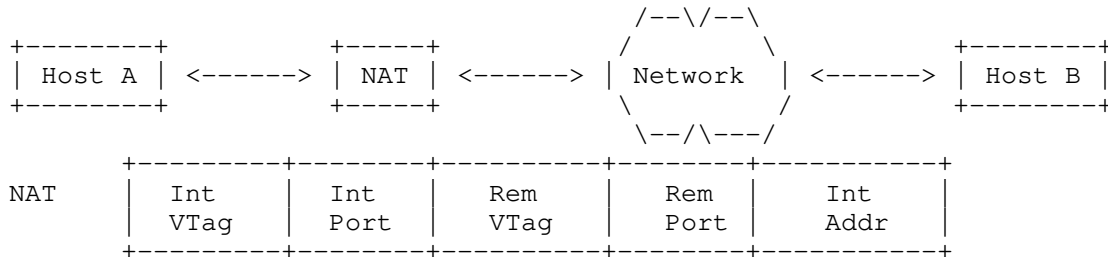
Please note that this section is informational only.

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
```

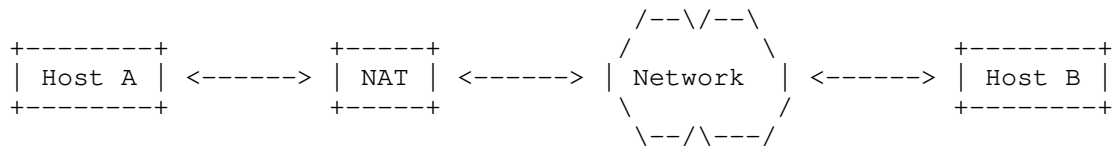
A NAT binding table entry is created, the source address is substituted and the packet is sent on:

NAT function creates entry:

	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
NAT	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.



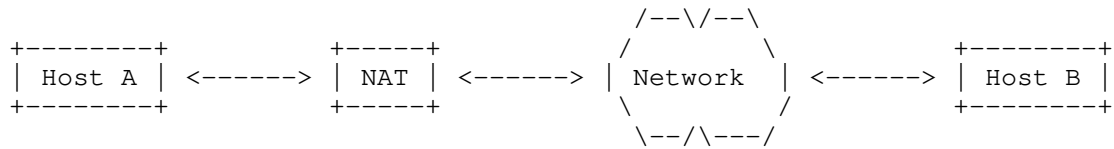
INIT ACK[Initiate-Tag = 5678]
 192.0.2.1:1 <----- 203.0.113.1:2
 Int-VTag = 1234

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

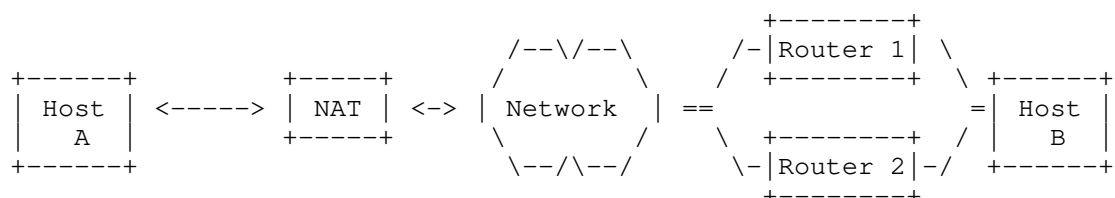
COOKIE ECHO
 192.0.2.1:1 -----> 203.0.113.1:2
 Rem-VTag = 5678

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
  
```

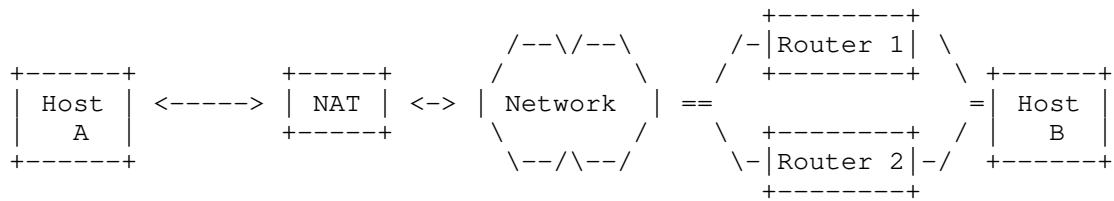
NAT function creates entry:

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:

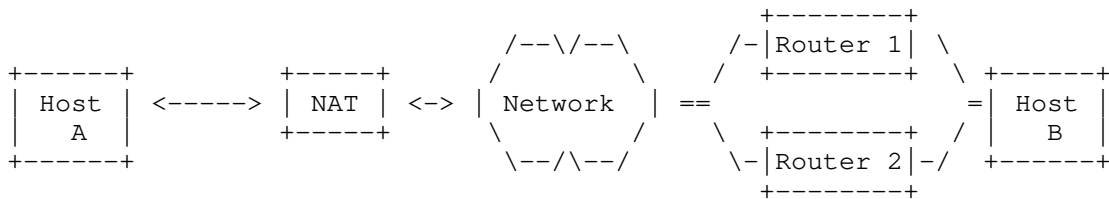
NAT					
	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	5678	2	10.0.0.1

```

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

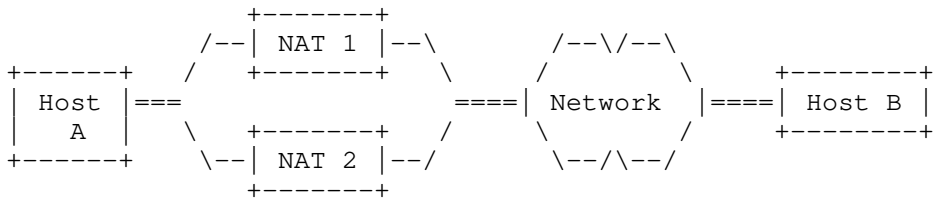
COOKIE ECHO
192.0.2.1:1 -----> 203.0.113.1:2
Rem-VTag = 5678

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:

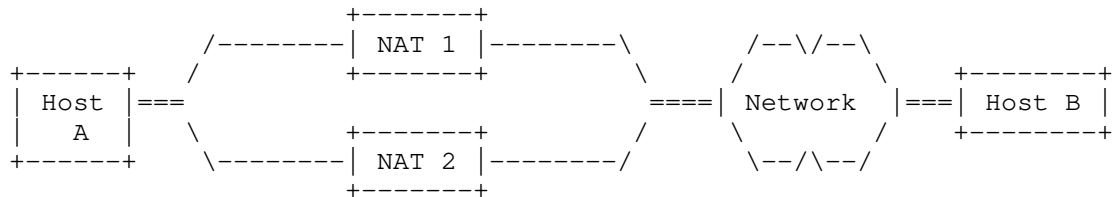
NAT 1	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	0	2	10.0.0.1

```

                                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:

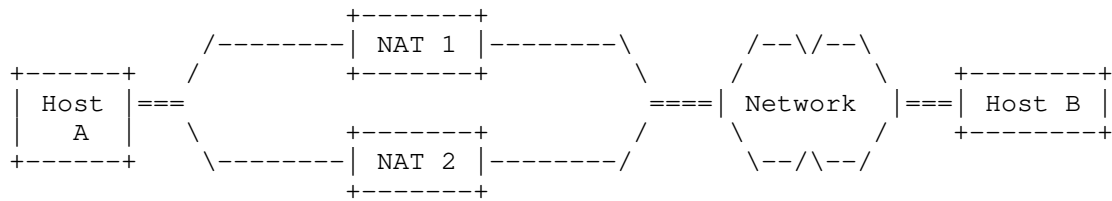
NAT 1	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr
	1234	1	5678	2	10.0.0.1

```

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.



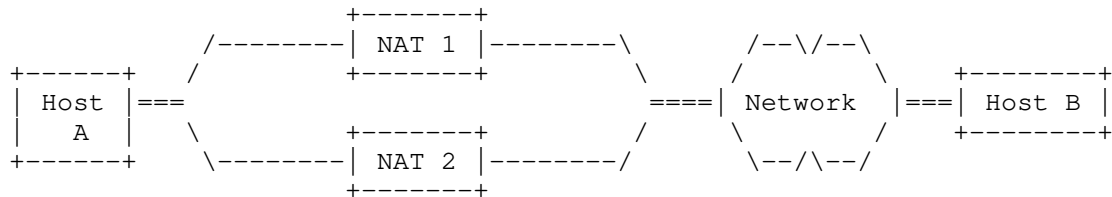
COOKIE ECHO
 10.0.0.1:1 -----> 203.0.113.1:2
 Rem-VTag = 5678

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

```

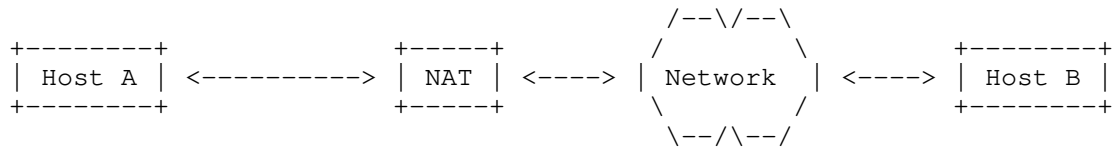
```

                        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.



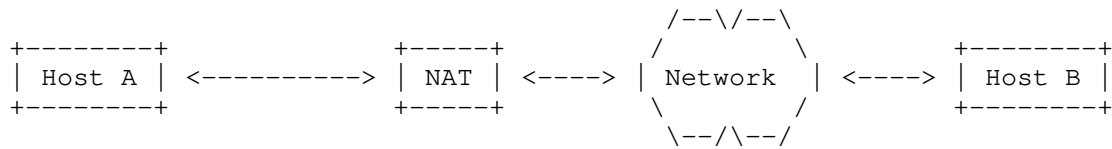
NAT	Int VTag	Int Port	Rem VTag	Rem Port	Int Addr

```

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

```

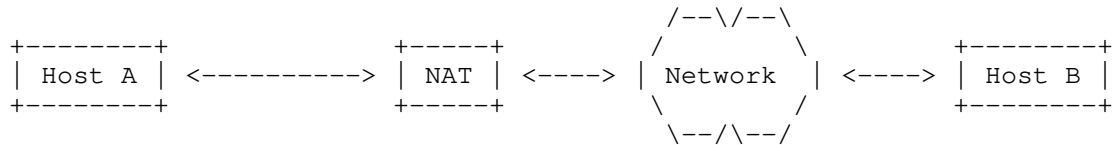
The NAT function cannot find an entry in the NAT binding table for the association. It sends a packet containing an ERROR chunk with the M bit set and the cause "NAT state missing".



```

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.



```

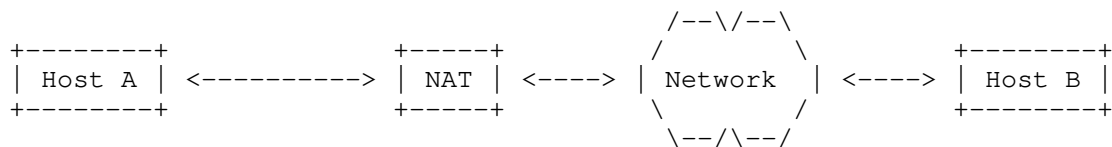
ASCONF [ADD-IP, DELETE-IP, Int-VTag=1234, Rem-VTag = 5678]
10.0.0.1:1 -----> 203.0.113.129:2
      Rem-VTag = 5678
  
```

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

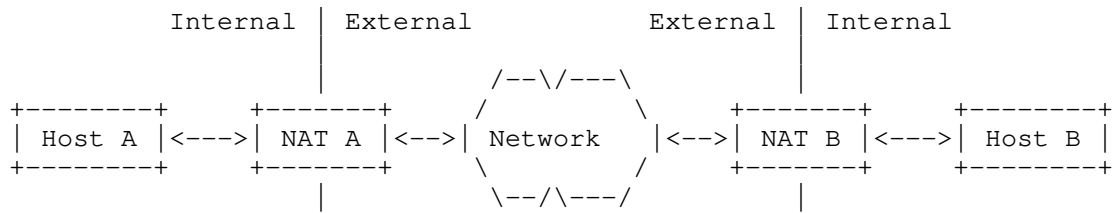
                                ASCONF ACK
                                10.1.0.1:1 <----- 203.0.113.129:2
                                Int-VTag = 1234

                                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

```

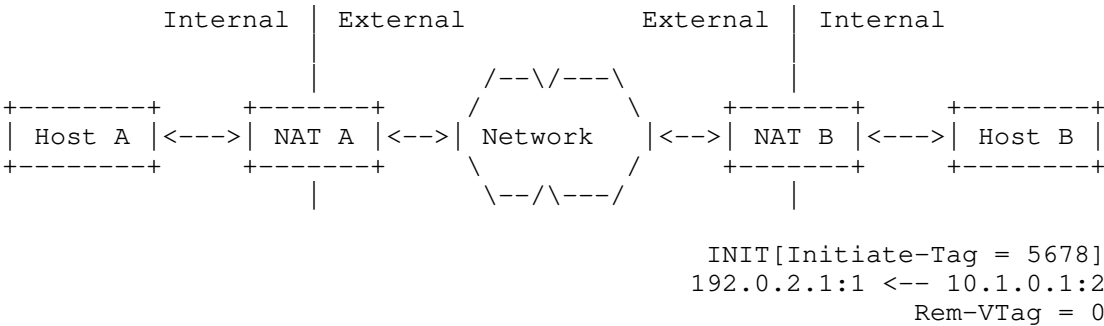
INIT[Initiate-Tag = 1234]
192.0.2.1:1 -----> 203.0.113.1:2
    Rem-VTag = 0

```

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
-------	-------------	-------------	-------------	-------------	-------------

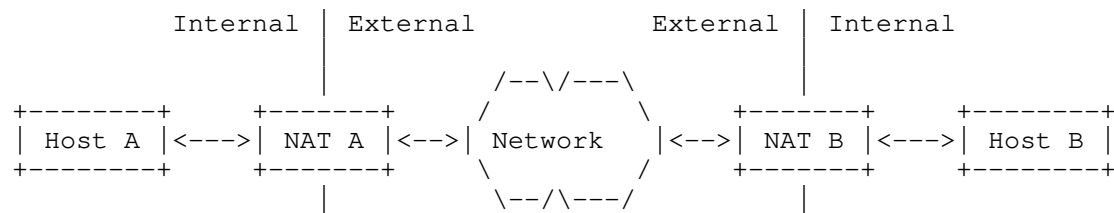
Now Host B sends a packet containing an INIT chunk, which is processed by NAT function B. Its parameters are used to create an entry.



NAT B	Int VTag	Int Port	Rem VTag	Rem Port	Int 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:

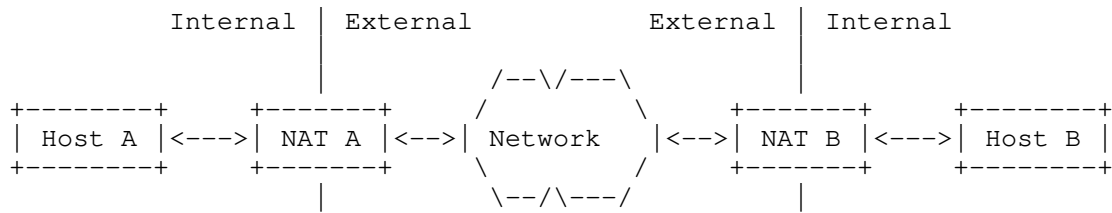


VTag != Int-VTag, but Rem-VTag == 0, find entry.

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
  
```

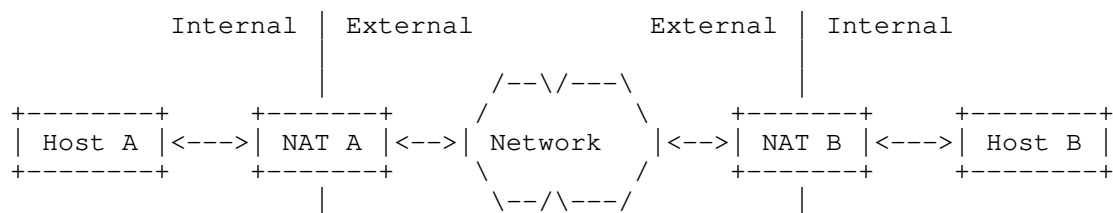
NAT function B updates entry:

NAT B	Int	Int	Rem	Rem	Int
	VTag	Port	VTag	Port	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
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)
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,
                                   /           Start=2,End=2]
                                   <-----/ (remove 6 from out-queue,
and mark 7 as "1" missing report)

```

 New text: (Section 6.7, Figure 9)

```

Endpoint A                                Endpoint Z
{App sends 3 messages; strm 0}
DATA [TSN=6,Strm=0,Seq=2] -----> (ack delayed)
(Start T3-rtx timer)

DATA [TSN=7,Strm=0,Seq=3] -----> X (lost)

DATA [TSN=8,Strm=0,Seq=4] -----> (gap detected,
                                   immediately send ack)
                                   /----- SACK [TSN Ack=6,Block=1,
                                   /           Start=2,End=2]
                                   <-----/
(remove 6 from out-queue,
and mark 7 as "1" missing report)

```

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

Old text: (Section 6.10)

An endpoint bundles chunks by simply including multiple chunks in one outbound SCTP packet. The total size of the resultant IP datagram, including the SCTP packet and IP headers, MUST be less 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

Transport Area Working Group
Internet-Draft
Updates: 6040, 2661, 2784, 3931, 4380,
7450 (if approved)
Intended status: Standards Track
Expires: November 25, 2021

B. Briscoe
Independent
May 24, 2021

Propagating Explicit Congestion Notification Across IP Tunnel Headers
Separated by a Shim
draft-ietf-tsvwg-rfc6040update-shim-14

Abstract

RFC 6040 on "Tunnelling of Explicit Congestion Notification" made the rules for propagation of ECN consistent for all forms of IP in IP tunnel. This specification updates RFC 6040 to clarify that its scope includes tunnels where two IP headers are separated by at least one shim header that is not sufficient on its own for wide area packet forwarding. It surveys widely deployed IP tunnelling protocols that use such shim header(s) and updates the specifications of those that do not mention ECN propagation (L2TPv2, L2TPv3, GRE, Teredo and AMT). This specification also updates RFC 6040 with configuration requirements needed to make any legacy tunnel ingress safe.

Status of This Memo

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

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

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

This Internet-Draft will expire on November 25, 2021.

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	2
2. Terminology	3
3. Scope of RFC 6040	3
3.1. Feasibility of ECN Propagation between Tunnel Headers . .	4
3.2. Desirability of ECN Propagation between Tunnel Headers .	5
4. Making a non-ECN Tunnel Ingress Safe by Configuration	5
5. ECN Propagation and Fragmentation/Reassembly	7
6. IP-in-IP Tunnels with Tightly Coupled Shim Headers	7
6.1. Specific Updates to Protocols under IETF Change Control .	10
6.1.1. L2TP (v2 and v3) ECN Extension	10
6.1.2. GRE	13
6.1.3. Teredo	14
6.1.4. AMT	15
7. IANA Considerations	17
8. Security Considerations	17
9. Comments Solicited	17
10. Acknowledgements	17
11. References	18
11.1. Normative References	18
11.2. Informative References	19
Author's Address	22

1. Introduction

RFC 6040 on "Tunnelling of Explicit Congestion Notification" [RFC6040] made the rules for propagation of Explicit Congestion Notification (ECN [RFC3168]) consistent for all forms of IP in IP tunnel.

A common pattern for many tunnelling protocols is to encapsulate an inner IP header (v4 or v6) with shim header(s) then an outer IP header (v4 or v6). Some of these shim headers are designed as generic encapsulations, so they do not necessarily directly encapsulate an inner IP header. Instead they can encapsulate headers such as link-layer (L2) protocols that in turn often encapsulate IP.

To clear up confusion, this specification clarifies that the scope of RFC 6040 includes any IP-in-IP tunnel, including those with shim header(s) and other encapsulations between the IP headers. Where necessary, it updates the specifications of the relevant encapsulation protocols with the specific text necessary to comply with RFC 6040.

This specification also updates RFC 6040 to state how operators ought to configure a legacy tunnel ingress to avoid unsafe system configurations.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119] when, and only when, they appear in all capitals, as shown here.

This specification uses the terminology defined in RFC 6040 [RFC6040].

3. Scope of RFC 6040

In section 1.1 of RFC 6040, its scope is defined as:

"...ECN field processing at encapsulation and decapsulation for any IP-in-IP tunnelling, whether IPsec or non-IPsec tunnels. It applies irrespective of whether IPv4 or IPv6 is used for either the inner or outer headers. ..."

This was intended to include cases where shim header(s) sit between the IP headers. Many tunnelling implementers have interpreted the scope of RFC 6040 as it was intended, but it is ambiguous. Therefore, this specification updates RFC 6040 by adding the following scoping text after the sentences quoted above:

It applies in cases where an outer IP header encapsulates an inner IP header either directly or indirectly by encapsulating other headers that in turn encapsulate (or might encapsulate) an inner IP header.

There is another problem with the scope of RFC 6040. Like many IETF specifications, RFC 6040 is written as a specification that implementations can choose to claim compliance with. This means it does not cover two important cases:

1. those cases where it is infeasible for an implementation to access an inner IP header when adding or removing an outer IP header;
2. those implementations that choose not to propagate ECN between IP headers.

However, the ECN field is a non-optional part of the IP header (v4 and v6). So any implementation that creates an outer IP header has to give the ECN field some value. There is only one safe value a tunnel ingress can use if it does not know whether the egress supports propagation of the ECN field; it has to clear the ECN field in any outer IP header to 0b00.

However, an RFC has no jurisdiction over implementations that choose not to comply with it or cannot comply with it, including all those implementations that pre-dated the RFC. Therefore it would have been unreasonable to add such a requirement to RFC 6040. Nonetheless, to ensure safe propagation of the ECN field over tunnels, it is reasonable to add requirements on operators, to ensure they configure their tunnels safely (where possible). Before stating these configuration requirements in Section 4, the factors that determine whether propagating ECN is feasible or desirable will be briefly introduced.

3.1. Feasibility of ECN Propagation between Tunnel Headers

In many cases shim header(s) and an outer IP header are always added to (or removed from) an inner IP packet as part of the same procedure. We call this a tightly coupled shim header. Processing the shim and outer together is often necessary because the shim(s) are not sufficient for packet forwarding in their own right; not unless complemented by an outer header. In these cases it will often be feasible for an implementation to propagate the ECN field between the IP headers.

In some cases a tunnel adds an outer IP header and a tightly coupled shim header to an inner header that is not an IP header, but that in turn encapsulates an IP header (or might encapsulate an IP header). For instance an inner Ethernet (or other link layer) header might encapsulate an inner IP header as its payload. We call this a tightly coupled shim over an encapsulating header.

Digging to arbitrary depths to find an inner IP header within an encapsulation is strictly a layering violation so it cannot be a required behaviour. Nonetheless, some tunnel endpoints already look within a L2 header for an IP header, for instance to map the Diffserv codepoint between an encapsulated IP header and an outer IP header

[RFC2983]. In such cases at least, it should be feasible to also (independently) propagate the ECN field between the same IP headers. Thus, access to the ECN field within an encapsulating header can be a useful and benign optimization. The guidelines in section 5 of [I-D.ietf-tsvwg-ecn-encap-guidelines] give the conditions for this layering violation to be benign.

3.2. Desirability of ECN Propagation between Tunnel Headers

Developers and network operators are encouraged to implement and deploy tunnel endpoints compliant with RFC 6040 (as updated by the present specification) in order to provide the benefits of wider ECN deployment [RFC8087]. Nonetheless, propagation of ECN between IP headers, whether separated by shim headers or not, has to be optional to implement and to use, because:

- o Legacy implementations of tunnels without any ECN support already exist
- o A network might be designed so that there is usually no bottleneck within the tunnel
- o If the tunnel endpoints would have to search within an L2 header to find an encapsulated IP header, it might not be worth the potential performance hit

4. Making a non-ECN Tunnel Ingress Safe by Configuration

Even when no specific attempt has been made to implement propagation of the ECN field at a tunnel ingress, it ought to be possible for the operator to render a tunnel ingress safe by configuration. The main safety concern is to disable (clear to zero) the ECN capability in the outer IP header at the ingress if the egress of the tunnel does not implement ECN logic to propagate any ECN markings into the packet forwarded beyond the tunnel. Otherwise the non-ECN egress could discard any ECN marking introduced within the tunnel, which would break all the ECN-based control loops that regulate the traffic load over the tunnel.

Therefore this specification updates RFC 6040 by inserting the following text at the end of section 4.3:

"

Whether or not an ingress implementation claims compliance with RFC 6040, RFC 4301 or RFC3168, when the outer tunnel header is IP (v4 or v6), if possible, the operator MUST configure the ingress to zero the outer ECN field in any of the following cases:

- * if it is known that the tunnel egress does not support any of the RFCs that define propagation of the ECN field (RFC 6040, RFC 4301 or the full functionality mode of RFC 3168)
- * or if the behaviour of the egress is not known or an egress with unknown behaviour might be dynamically paired with the ingress.
- * or if an IP header might be encapsulated within a non-IP header that the tunnel ingress is encapsulating, but the ingress does not inspect within the encapsulation.

For the avoidance of doubt, the above only concerns the outer IP header. The ingress MUST NOT alter the ECN field of the arriving IP header that will become the inner IP header.

In order that the network operator can comply with the above safety rules, even if an implementation of a tunnel ingress does not claim to support RFC 6040, RFC 4301 or the full functionality mode of RFC 3168:

- * it MUST NOT treat the former ToS octet (IPv4) or the former Traffic Class octet (IPv6) as a single 8-bit field, as the resulting linkage of ECN and Diffserv field propagation between inner and outer is not consistent with the definition of the 6-bit Diffserv field in [RFC2474] and [RFC3260];
- * it SHOULD be able to be configured to zero the ECN field of the outer header.

"

For instance, if a tunnel ingress with no ECN-specific logic had a configuration capability to refer to the last 2 bits of the old ToS Byte of the outer (e.g. with a 0x3 mask) and set them to zero, while also being able to allow the DSCP to be re-mapped independently, that would be sufficient to satisfy both the above implementation requirements.

There might be concern that the above "MUST NOT" makes compliant implementations non-compliant at a stroke. However, by definition it solely applies to equipment that provides Diffserv configuration. Any such Diffserv equipment that is configuring treatment of the former ToS octet (IPv4) or the former Traffic Class octet (IPv6) as a single 8-bit field must have always been non-compliant with the definition of the 6-bit Diffserv field in [RFC2474] and [RFC3260]. If a tunnel ingress does not have any ECN logic, copying the ECN field as a side-effect of copying the DSCP is a seriously unsafe bug

that risks breaking the feedback loops that regulate load on a tunnel.

Zeroing the outer ECN field of all packets in all circumstances would be safe, but it would not be sufficient to claim compliance with RFC 6040 because it would not meet the aim of introducing ECN support to tunnels (see Section 4.3 of [RFC6040]).

5. ECN Propagation and Fragmentation/Reassembly

The following requirements update RFC6040, which omitted handling of the ECN field during fragmentation or reassembly. These changes might alter how many ECN-marked packets are propagated by a tunnel that fragments packets, but this would not raise any backward compatibility issues:

If a tunnel ingress fragments a packet, it MUST set the outer ECN field of all the fragments to the same value as it would have set if it had not fragmented the packet.

Section 5.3 of [RFC3168] specifies ECN requirements for reassembly of sets of outer fragments [I-D.ietf-intarea-tunnels] into packets. The following two additional requirements apply at a tunnel egress:

- o During reassembly of outer fragments [I-D.ietf-intarea-tunnels], if the ECN fields of the outer headers being reassembled into a single packet consist of a mixture of Not-ECT and other ECN codepoints, the packet MUST be discarded.
- o If there is mix of ECT(0) and ECT(1) fragments, then the reassembled packet MUST be set to either ECT(0) or ECT(1). In this case, reassembly SHOULD take into account that the RFC series has so far ensured that ECT(0) and ECT(1) can either be considered equivalent, or they can provide 2 levels of congestion severity, where the ranking of severity from highest to lowest is CE, ECT(1), ECT(0) [RFC6040].

6. IP-in-IP Tunnels with Tightly Coupled Shim Headers

There follows a list of specifications of encapsulations with tightly coupled shim header(s), in rough chronological order. The list is confined to standards track or widely deployed protocols. The list is not necessarily exhaustive so, for the avoidance of doubt, the scope of RFC 6040 is defined in Section 3 and is not limited to this list.

- o PPTP (Point-to-Point Tunneling Protocol) [RFC2637];

- o L2TP (Layer 2 Tunneling Protocol), specifically L2TPv2 [RFC2661] and L2TPv3 [RFC3931], which not only includes all the L2-specific specializations of L2TP, but also derivatives such as the Keyed IPv6 Tunnel [RFC8159];
- o GRE (Generic Routing Encapsulation) [RFC2784] and NVGRE (Network Virtualization using GRE) [RFC7637];
- o GTP (GPRS Tunneling Protocol), specifically GTPv1 [GTPv1], GTP v1 User Plane [GTPv1-U], GTP v2 Control Plane [GTPv2-C];
- o Teredo [RFC4380];
- o CAPWAP (Control And Provisioning of Wireless Access Points) [RFC5415];
- o LISP (Locator/Identifier Separation Protocol) [RFC6830];
- o AMT (Automatic Multicast Tunneling) [RFC7450];
- o VXLAN (Virtual eXtensible Local Area Network) [RFC7348] and VXLAN-GPE [I-D.ietf-nvo3-vxlan-gpe];
- o The Network Service Header (NSH [RFC8300]) for Service Function Chaining (SFC);
- o Geneve [RFC8926];
- o GUE (Generic UDP Encapsulation) [I-D.ietf-intarea-gue];
- o Direct tunnelling of an IP packet within a UDP/IP datagram (see Section 3.1.11 of [RFC8085]);
- o TCP Encapsulation of IKE and IPsec Packets (see Section 12.5 of [RFC8229]).

Some of the listed protocols enable encapsulation of a variety of network layer protocols as inner and/or outer. This specification applies in the cases where there is an inner and outer IP header as described in Section 3. Otherwise [I-D.ietf-tsvwg-ecn-encap-guidelines] gives guidance on how to design propagation of ECN into other protocols that might encapsulate IP.

Where protocols in the above list need to be updated to specify ECN propagation and they are under IETF change control, update text is given in the following subsections. For those not under IETF control, it is RECOMMENDED that implementations of encapsulation and decapsulation comply with RFC 6040. It is also RECOMMENDED that

their specifications are updated to add a requirement to comply with RFC 6040 (as updated by the present document).

PPTP is not under the change control of the IETF, but it has been documented in an informational RFC [RFC2637]. However, there is no need for the present specification to update PPTP because L2TP has been developed as a standardized replacement.

NVGRE is not under the change control of the IETF, but it has been documented in an informational RFC [RFC7637]. NVGRE is a specific use-case of GRE (it re-purposes the key field from the initial specification of GRE [RFC1701] as a Virtual Subnet ID). Therefore the text that updates GRE in Section 6.1.2 below is also intended to update NVGRE.

Although the definition of the various GTP shim headers is under the control of the 3GPP, it is hard to determine whether the 3GPP or the IETF controls standardization of the `_process_` of adding both a GTP and an IP header to an inner IP header. Nonetheless, the present specification is provided so that the 3GPP can refer to it from any of its own specifications of GTP and IP header processing.

The specification of CAPWAP already specifies RFC 3168 ECN propagation and ECN capability negotiation. Without modification the CAPWAP specification already interworks with the backward compatible updates to RFC 3168 in RFC 6040.

LISP made the ECN propagation procedures in RFC 3168 mandatory from the start. RFC 3168 has since been updated by RFC 6040, but the changes are backwards compatible so there is still no need for LISP tunnel endpoints to negotiate their ECN capabilities.

VXLAN is not under the change control of the IETF but it has been documented in an informational RFC. In contrast, VXLAN-GPE (Generic Protocol Extension) is being documented under IETF change control. It is RECOMMENDED that VXLAN and VXLAN-GPE implementations comply with RFC 6040 when the VXLAN header is inserted between (or removed from between) IP headers. The authors of any future update to these specifications are encouraged to add a requirement to comply with RFC 6040 as updated by the present specification.

The Network Service Header (NSH [RFC8300]) has been defined as a shim-based encapsulation to identify the Service Function Path (SFP) in the Service Function Chaining (SFC) architecture [RFC7665]. A proposal has been made for the processing of ECN when handling transport encapsulation [I-D.ietf-sfc-nsh-ecn-support].

The specifications of Geneve and GUE already refer to RFC 6040 for ECN encapsulation.

Section 3.1.11 of RFC 8085 already explains that a tunnel that encapsulates an IP header within a UDP/IP datagram needs to follow RFC 6040 when propagating the ECN field between inner and outer IP headers. The requirements in Section 4 update RFC 6040, and hence implicitly update the UDP usage guidelines in RFC 8085 to add the important but previously unstated requirement that, if the UDP tunnel egress does not, or might not, support ECN propagation, a UDP tunnel ingress has to clear the outer IP ECN field to 0b00, e.g. by configuration.

Section 12.5 of TCP Encapsulation of IKE and IPsec Packets [RFC8229] already recommends the compatibility mode of RFC 6040 in this case, because there is not a one-to-one mapping between inner and outer packets.

6.1. Specific Updates to Protocols under IETF Change Control

6.1.1. L2TP (v2 and v3) ECN Extension

The L2TP terminology used here is defined in [RFC2661] and [RFC3931].

L2TPv3 [RFC3931] is used as a shim header between any packet-switched network (PSN) header (e.g. IPv4, IPv6, MPLS) and many types of layer 2 (L2) header. The L2TPv3 shim header encapsulates an L2-specific sub-layer then an L2 header that is likely to contain an inner IP header (v4 or v6). Then this whole stack of headers can be encapsulated optionally within an outer UDP header then an outer PSN header that is typically IP (v4 or v6).

L2TPv2 is used as a shim header between any PSN header and a PPP header, which is in turn likely to encapsulate an IP header.

Even though these shims are rather fat (particularly in the case of L2TPv3), they still fit the definition of a tightly coupled shim header over an encapsulating header (Section 3.1), because all the headers encapsulating the L2 header are added (or removed) together. L2TPv2 and L2TPv3 are therefore within the scope of RFC 6040, as updated by Section 3 above.

L2TP maintainers are RECOMMENDED to implement the ECN extension to L2TPv2 and L2TPv3 defined in Section 6.1.1.2 below, in order to provide the benefits of ECN [RFC8087], whenever a node within an L2TP tunnel becomes the bottleneck for an end-to-end traffic flow.

6.1.1.1. Safe Configuration of a 'Non-ECN' Ingress LCCE

The following text is appended to both Section 5.3 of [RFC2661] and Section 4.5 of [RFC3931] as an update to the base L2TPv2 and L2TPv3 specifications:

The operator of an LCCE that does not support the ECN Extension in Section 6.1.1.2 of RFCXXXX MUST follow the configuration requirements in Section 4 of RFCXXXX to ensure it clears the outer IP ECN field to 0b00 when the outer PSN header is IP (v4 or v6). {RFCXXXX refers to the present document so it will need to be inserted by the RFC Editor}

In particular, for an LCCE implementation that does not support the ECN Extension, this means that configuration of how it propagates the ECN field between inner and outer IP headers MUST be independent of any configuration of the Diffserv extension of L2TP [RFC3308].

6.1.1.2. ECN Extension for L2TP (v2 or v3)

When the outer PSN header and the payload inside the L2 header are both IP (v4 or v6), to comply with RFC 6040, an LCCE will follow the rules for propagation of the ECN field at ingress and egress in Section 4 of RFC 6040 [RFC6040].

Before encapsulating any data packets, RFC 6040 requires an ingress LCCE to check that the egress LCCE supports ECN propagation as defined in RFC 6040 or one of its compatible predecessors ([RFC4301] or the full functionality mode of [RFC3168]). If the egress supports ECN propagation, the ingress LCCE can use the normal mode of encapsulation (copying the ECN field from inner to outer). Otherwise, the ingress LCCE has to use compatibility mode [RFC6040] (clearing the outer IP ECN field to 0b00).

An LCCE can determine the remote LCCE's support for ECN either statically (by configuration) or by dynamic discovery during setup of each control connection between the LCCEs, using the Capability AVP defined in Section 6.1.1.2.1 below.

Where the outer PSN header is some protocol other than IP that supports ECN, the appropriate ECN propagation specification will need to be followed, e.g. "Explicit Congestion Marking in MPLS" [RFC5129]. Where no specification exists for ECN propagation by a particular PSN, [I-D.ietf-tsvwg-ecn-encap-guidelines] gives general guidance on how to design ECN propagation into a protocol that encapsulates IP.

6.1.1.2.1. LCCE Capability AVP for ECN Capability Negotiation

The LCCE Capability Attribute-Value Pair (AVP) defined here has Attribute Type ZZ. The Attribute Value field for this AVP is a bit-mask with the following 16-bit format:

```

      0                               1
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|X X X X X X X X X X X X X X X E|
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

```

Figure 1: Value Field for the LCCE Capability Attribute

This AVP MAY be present in the following message types: SCCRQ and SCCRP (Start-Control-Connection-Request and Start-Control-Connection-Reply). This AVP MAY be hidden (the H-bit set to 0 or 1) and is optional (M-bit not set). The length (before hiding) of this AVP MUST be 8 octets. The Vendor ID is the IETF Vendor ID of 0.

Bit 15 of the Value field of the LCCE Capability AVP is defined as the ECN Capability flag (E). When the ECN Capability flag is set to 1, it indicates that the sender supports ECN propagation. When the ECN Capability flag is cleared to zero, or when no LCCE Capability AVP is present, it indicates that the sender does not support ECN propagation. All the other bits are reserved. They MUST be cleared to zero when sent and ignored when received or forwarded.

An LCCE initiating a control connection will send a Start-Control-Connection-Request (SCCRQ) containing an LCCE Capability AVP with the ECN Capability flag set to 1. If the tunnel terminator supports ECN, it will return a Start-Control-Connection-Reply (SCCRP) that also includes an LCCE Capability AVP with the ECN Capability flag set to 1. Then, for any sessions created by that control connection, both ends of the tunnel can use the normal mode of RFC 6040, i.e. it can copy the IP ECN field from inner to outer when encapsulating data packets.

If, on the other hand, the tunnel terminator does not support ECN it will ignore the ECN flag in the LCCE Capability AVP and send an SCCRP to the tunnel initiator without a Capability AVP (or with a Capability AVP but with the ECN Capability flag cleared to zero). The tunnel initiator interprets the absence of the ECN Capability flag in the SCCRP as an indication that the tunnel terminator is incapable of supporting ECN. When encapsulating data packets for any sessions created by that control connection, the tunnel initiator will then use the compatibility mode of RFC 6040 to clear the ECN field of the outer IP header to 0b00.

If the tunnel terminator does not support this ECN extension, the network operator is still expected to configure it to comply with the safety provisions set out in Section 6.1.1.1 above, when it acts as an ingress LCCE.

6.1.2. GRE

The GRE terminology used here is defined in [RFC2784]. GRE is often used as a tightly coupled shim header between IP headers. Sometimes the GRE shim header encapsulates an L2 header, which might in turn encapsulate an IP header. Therefore GRE is within the scope of RFC 6040 as updated by Section 3 above.

GRE tunnel endpoint maintainers are RECOMMENDED to support [RFC6040] as updated by the present specification, in order to provide the benefits of ECN [RFC8087] whenever a node within a GRE tunnel becomes the bottleneck for an end-to-end IP traffic flow tunnelled over GRE using IP as the delivery protocol (outer header).

GRE itself does not support dynamic set-up and configuration of tunnels. However, control plane protocols such as Mobile IPv4 (MIP4) [RFC5944], Mobile IPv6 (MIP6) [RFC6275], Proxy Mobile IP (PMIP) [RFC5845] and IKEv2 [RFC7296] are sometimes used to set up GRE tunnels dynamically.

When these control protocols set up IP-in-IP or IPSec tunnels, it is likely that they propagate the ECN field as defined in RFC 6040 or one of its compatible predecessors (RFC 4301 or the full functionality mode of RFC 3168). However, if they use a GRE encapsulation, this presumption is less sound.

Therefore, If the outer delivery protocol is IP (v4 or v6) the operator is obliged to follow the safe configuration requirements in Section 4 above. Section 6.1.2.1 below updates the base GRE specification with this requirement, to emphasize its importance.

Where the delivery protocol is some protocol other than IP that supports ECN, the appropriate ECN propagation specification will need to be followed, e.g Explicit Congestion Marking in MPLS [RFC5129]. Where no specification exists for ECN propagation by a particular PSN, [I-D.ietf-tsvwg-ecn-encap-guidelines] gives more general guidance on how to propagate ECN to and from protocols that encapsulate IP.

6.1.2.1. Safe Configuration of a 'Non-ECN' GRE Ingress

The following text is appended to Section 3 of [RFC2784] as an update to the base GRE specification:

The operator of a GRE tunnel ingress MUST follow the configuration requirements in Section 4 of RFCXXXX when the outer delivery protocol is IP (v4 or v6). {RFCXXXX refers to the present document so it will need to be inserted by the RFC Editor}

6.1.3. Teredo

Teredo [RFC4380] provides a way to tunnel IPv6 over an IPv4 network, with a UDP-based shim header between the two.

For Teredo tunnel endpoints to provide the benefits of ECN, the Teredo specification would have to be updated to include negotiation of the ECN capability between Teredo tunnel endpoints. Otherwise it would be unsafe for a Teredo tunnel ingress to copy the ECN field to the IPv6 outer.

It is believed that current implementations do not support propagation of ECN, but that they do safely zero the ECN field in the outer IPv6 header. However the specification does not mention anything about this.

To make existing Teredo deployments safe, it would be possible to add ECN capability negotiation to those that are subject to remote OS update. However, for those implementations not subject to remote OS update, it will not be feasible to require them to be configured correctly, because Teredo tunnel endpoints are generally deployed on hosts.

Therefore, until ECN support is added to the specification of Teredo, the only feasible further safety precaution available here is to update the specification of Teredo implementations with the following text, as a new section 5.1.3:

"5.1.3 Safe 'Non-ECN' Teredo Encapsulation

A Teredo tunnel ingress implementation that does not support ECN propagation as defined in RFC 6040 or one of its compatible predecessors (RFC 4301 or the full functionality mode of RFC 3168) MUST zero the ECN field in the outer IPv6 header."

6.1.4. AMT

Automatic Multicast Tunneling (AMT [RFC7450]) is a tightly coupled shim header that encapsulates an IP packet and is itself encapsulated within a UDP/IP datagram. Therefore AMT is within the scope of RFC 6040 as updated by Section 3 above.

AMT tunnel endpoint maintainers are RECOMMENDED to support [RFC6040] as updated by the present specification, in order to provide the benefits of ECN [RFC8087] whenever a node within an AMT tunnel becomes the bottleneck for an IP traffic flow tunnelled over AMT.

To comply with RFC 6040, an AMT relay and gateway will follow the rules for propagation of the ECN field at ingress and egress respectively, as described in Section 4 of RFC 6040 [RFC6040].

Before encapsulating any data packets, RFC 6040 requires an ingress AMT relay to check that the egress AMT gateway supports ECN propagation as defined in RFC 6040 or one of its compatible predecessors (RFC 4301 or the full functionality mode of RFC 3168). If the egress gateway supports ECN, the ingress relay can use the normal mode of encapsulation (copying the IP ECN field from inner to outer). Otherwise, the ingress relay has to use compatibility mode, which means it has to clear the outer ECN field to zero [RFC6040].

An AMT tunnel is created dynamically (not manually), so the relay will need to determine the remote gateway's support for ECN using the ECN capability declaration defined in Section 6.1.4.2 below.

6.1.4.1. Safe Configuration of a 'Non-ECN' Ingress AMT Relay

The following text is appended to Section 4.2.2 of [RFC7450] as an update to the AMT specification:

The operator of an AMT relay that does not support RFC 6040 or one of its compatible predecessors (RFC 4301 or the full functionality mode of RFC 3168) MUST follow the configuration requirements in Section 4 of RFCXXXX to ensure it clears the outer IP ECN field to zero. {RFCXXXX refers to the present document so it will need to be inserted by the RFC Editor}

6.1.4.2. ECN Capability Declaration of an AMT Gateway

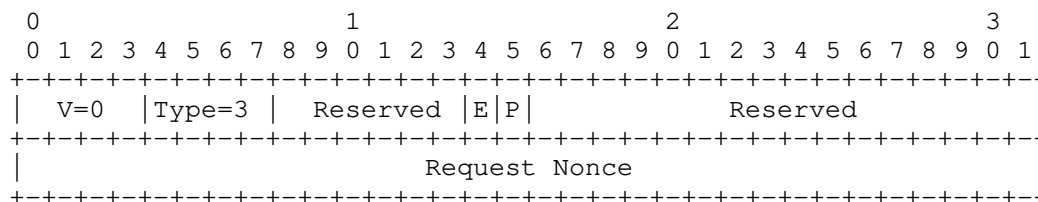


Figure 2: Updated AMT Request Message Format

Bit 14 of the AMT Request Message counting from 0 (or bit 7 of the Reserved field counting from 1) is defined here as the AMT Gateway ECN Capability flag (E), as shown in Figure 2. The definitions of all other fields in the AMT Request Message are unchanged from RFC 7450.

When the E flag is set to 1, it indicates that the sender of the message supports RFC 6040 ECN propagation. When it is cleared to zero, it indicates the sender of the message does not support RFC 6040 ECN propagation. An AMT gateway "that supports RFC 6040 ECN propagation" means one that propagates the ECN field to the forwarded data packet based on the combination of arriving inner and outer ECN fields, as defined in Section 4 of RFC 6040.

The other bits of the Reserved field remain reserved. They will continue to be cleared to zero when sent and ignored when either received or forwarded, as specified in Section 5.1.3.3. of RFC 7450.

An AMT gateway that does not support RFC 6040 MUST NOT set the E flag of its Request Message to 1.

An AMT gateway that supports RFC 6040 ECN propagation MUST set the E flag of its Relay Discovery Message to 1.

The action of the corresponding AMT relay that receives a Request message with the E flag set to 1 depends on whether the relay itself supports RFC 6040 ECN propagation:

- o If the relay supports RFC 6040 ECN propagation, it will store the ECN capability of the gateway along with its address. Then whenever it tunnels datagrams towards this gateway, it MUST use the normal mode of RFC 6040 to propagate the ECN field when encapsulating datagrams (i.e. it copies the IP ECN field from inner to outer).

- o If the discovered AMT relay does not support RFC 6040 ECN propagation, it will ignore the E flag in the Reserved field, as per section 5.1.3.3. of RFC 7450.

If the AMT relay does not support RFC 6040 ECN propagation, the network operator is still expected to configure it to comply with the safety provisions set out in Section 6.1.4.1 above.

7. IANA Considerations

IANA is requested to assign the following L2TP Control Message Attribute Value Pair:

Attribute Type	Description	Reference
ZZ	ECN Capability	RFCXXXX

[TO BE REMOVED: This registration should take place at the following location: <https://www.iana.org/assignments/l2tp-parameters/l2tp-parameters.xhtml>]

8. Security Considerations

The Security Considerations in [RFC6040] and [I-D.ietf-tsvwg-ecn-encap-guidelines] apply equally to the scope defined for the present specification.

9. Comments Solicited

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

10. Acknowledgements

Thanks to Ing-jyh (Inton) Tsang for initial discussions on the need for ECN propagation in L2TP and its applicability. Thanks also to Carlos Pignataro, Tom Herbert, Ignacio Goyret, Alia Atlas, Praveen Balasubramanian, Joe Touch, Mohamed Boucadair, David Black, Jake Holland and Sri Gundavelli for helpful advice and comments. "A Comparison of IPv6-over-IPv4 Tunnel Mechanisms" [RFC7059] helped to identify a number of tunnelling protocols to include within the scope of this document.

Bob Briscoe was part-funded by the Research Council of Norway through the TimeIn project. The views expressed here are solely those of the authors.

11. References

11.1. Normative References

- [I-D.ietf-tsvwg-ecn-encap-guidelines]
Briscoe, B. and J. Kaippallimalil, "Guidelines for Adding Congestion Notification to Protocols that Encapsulate IP", draft-ietf-tsvwg-ecn-encap-guidelines-15 (work in progress), March 2021.
- [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>>.
- [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>>.
- [RFC2661] Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G., and B. Palter, "Layer Two Tunneling Protocol "L2TP"", RFC 2661, DOI 10.17487/RFC2661, August 1999, <<https://www.rfc-editor.org/info/rfc2661>>.
- [RFC2784] Farinacci, D., Li, T., Hanks, S., Meyer, D., and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 2784, DOI 10.17487/RFC2784, March 2000, <<https://www.rfc-editor.org/info/rfc2784>>.
- [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>>.
- [RFC3931] Lau, J., Ed., Townsley, M., Ed., and I. Goyret, Ed., "Layer Two Tunneling Protocol - Version 3 (L2TPv3)", RFC 3931, DOI 10.17487/RFC3931, March 2005, <<https://www.rfc-editor.org/info/rfc3931>>.
- [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>>.

- [RFC4380] Huitema, C., "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)", RFC 4380, DOI 10.17487/RFC4380, February 2006, <<https://www.rfc-editor.org/info/rfc4380>>.
- [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, <<https://www.rfc-editor.org/info/rfc6040>>.

11.2. Informative References

- [GTPv1] 3GPP, "GPRS Tunnelling Protocol (GTP) across the Gn and Gp interface", Technical Specification TS 29.060.
- [GTPv1-U] 3GPP, "General Packet Radio System (GPRS) Tunnelling Protocol User Plane (GTPv1-U)", Technical Specification TS 29.281.
- [GTPv2-C] 3GPP, "Evolved General Packet Radio Service (GPRS) Tunnelling Protocol for Control plane (GTPv2-C)", Technical Specification TS 29.274.
- [I-D.ietf-intarea-gue]
Herbert, T., Yong, L., and O. Zia, "Generic UDP Encapsulation", draft-ietf-intarea-gue-09 (work in progress), October 2019.
- [I-D.ietf-intarea-tunnels]
Touch, J. and M. Townsley, "IP Tunnels in the Internet Architecture", draft-ietf-intarea-tunnels-10 (work in progress), September 2019.
- [I-D.ietf-nvo3-vxlan-gpe]
(Editor), F. M., (editor), L. K., and U. E. (editor), "Generic Protocol Extension for VXLAN (VXLAN-GPE)", draft-ietf-nvo3-vxlan-gpe-11 (work in progress), March 2021.
- [I-D.ietf-sfc-nsh-ecn-support]
Eastlake, D. E., Briscoe, B., Li, Y., Malis, A. G., and X. Wei, "Explicit Congestion Notification (ECN) and Congestion Feedback Using the Network Service Header (NSH) and IPFIX", draft-ietf-sfc-nsh-ecn-support-05 (work in progress), April 2021.

- [RFC1701] Hanks, S., Li, T., Farinacci, D., and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 1701, DOI 10.17487/RFC1701, October 1994, <<https://www.rfc-editor.org/info/rfc1701>>.
- [RFC2637] Hamzeh, K., Pall, G., Verthein, W., Taarud, J., Little, W., and G. Zorn, "Point-to-Point Tunneling Protocol (PPTP)", RFC 2637, DOI 10.17487/RFC2637, July 1999, <<https://www.rfc-editor.org/info/rfc2637>>.
- [RFC2983] Black, D., "Differentiated Services and Tunnels", RFC 2983, DOI 10.17487/RFC2983, October 2000, <<https://www.rfc-editor.org/info/rfc2983>>.
- [RFC3260] Grossman, D., "New Terminology and Clarifications for Diffserv", RFC 3260, DOI 10.17487/RFC3260, April 2002, <<https://www.rfc-editor.org/info/rfc3260>>.
- [RFC3308] Calhoun, P., Luo, W., McPherson, D., and K. Peirce, "Layer Two Tunneling Protocol (L2TP) Differentiated Services Extension", RFC 3308, DOI 10.17487/RFC3308, November 2002, <<https://www.rfc-editor.org/info/rfc3308>>.
- [RFC5415] Calhoun, P., Ed., Montemurro, M., Ed., and D. Stanley, Ed., "Control And Provisioning of Wireless Access Points (CAPWAP) Protocol Specification", RFC 5415, DOI 10.17487/RFC5415, March 2009, <<https://www.rfc-editor.org/info/rfc5415>>.
- [RFC5845] Muhanna, A., Khalil, M., Gundavelli, S., and K. Leung, "Generic Routing Encapsulation (GRE) Key Option for Proxy Mobile IPv6", RFC 5845, DOI 10.17487/RFC5845, June 2010, <<https://www.rfc-editor.org/info/rfc5845>>.
- [RFC5944] Perkins, C., Ed., "IP Mobility Support for IPv4, Revised", RFC 5944, DOI 10.17487/RFC5944, November 2010, <<https://www.rfc-editor.org/info/rfc5944>>.
- [RFC6275] Perkins, C., Ed., Johnson, D., and J. Arkko, "Mobility Support in IPv6", RFC 6275, DOI 10.17487/RFC6275, July 2011, <<https://www.rfc-editor.org/info/rfc6275>>.
- [RFC6830] Farinacci, D., Fuller, V., Meyer, D., and D. Lewis, "The Locator/ID Separation Protocol (LISP)", RFC 6830, DOI 10.17487/RFC6830, January 2013, <<https://www.rfc-editor.org/info/rfc6830>>.

- [RFC7059] Steffann, S., van Beijnum, I., and R. van Rein, "A Comparison of IPv6-over-IPv4 Tunnel Mechanisms", RFC 7059, DOI 10.17487/RFC7059, November 2013, <<https://www.rfc-editor.org/info/rfc7059>>.
- [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>>.
- [RFC7348] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger, L., Sridhar, T., Bursell, M., and C. Wright, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks", RFC 7348, DOI 10.17487/RFC7348, August 2014, <<https://www.rfc-editor.org/info/rfc7348>>.
- [RFC7450] Bumgardner, G., "Automatic Multicast Tunneling", RFC 7450, DOI 10.17487/RFC7450, February 2015, <<https://www.rfc-editor.org/info/rfc7450>>.
- [RFC7637] Garg, P., Ed. and Y. Wang, Ed., "NVGRE: Network Virtualization Using Generic Routing Encapsulation", RFC 7637, DOI 10.17487/RFC7637, September 2015, <<https://www.rfc-editor.org/info/rfc7637>>.
- [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>>.
- [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>>.
- [RFC8159] Konstantynowicz, M., Ed., Heron, G., Ed., Schatzmayr, R., and W. Henderickx, "Keyed IPv6 Tunnel", RFC 8159, DOI 10.17487/RFC8159, May 2017, <<https://www.rfc-editor.org/info/rfc8159>>.
- [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>>.

- [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>>.
- [RFC8926] Gross, J., Ed., Ganga, I., Ed., and T. Sridhar, Ed.,
"Geneve: Generic Network Virtualization Encapsulation",
RFC 8926, DOI 10.17487/RFC8926, November 2020,
<<https://www.rfc-editor.org/info/rfc8926>>.

Author's Address

Bob Briscoe
Independent
UK

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

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 $GF(2^{2^8})$ 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.

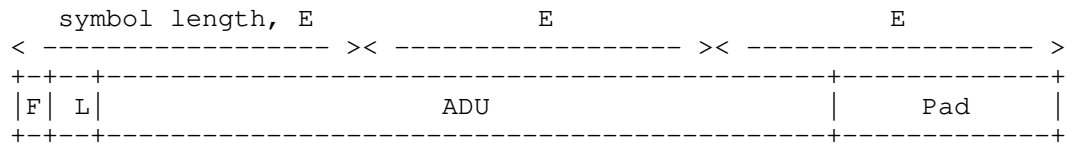


Figure 1: ADUI Creation example (here 3 source symbols are created for this ADUI).

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^{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^{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( $2^{2^8}$ )), otherwise
 *                    a fraction of them will be 0.
 * (in) m              Finite Field GF( $2^{2^m}$ ) 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;
    tinynt32_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 */
            tinynt32_init(&s, repair_key);
            for (i = 0 ; i < cc_nb ; i++) {
                cc_tab[i] = (tinynt32_rand16(&s) <= dt) ? 1 : 0;
            }
        }
        break;

    case 8:
        tinynt32_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) tinynt32_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 (tinynt32_rand16(&s) <= dt) {
                    do {
                        cc_tab[i] = (uint8_t) tinynt32_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.

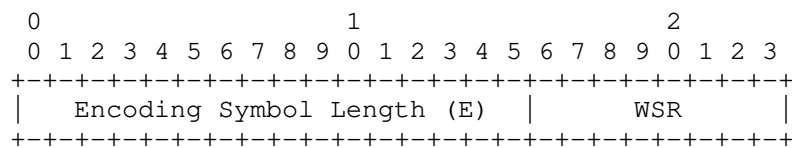


Figure 4: FSSI Encoding Format

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.

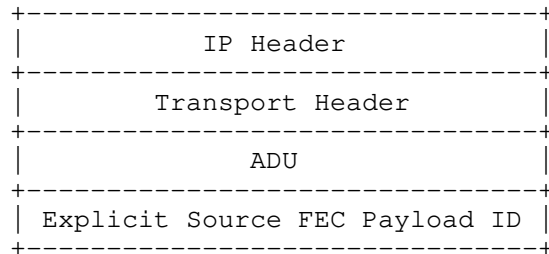


Figure 5: Structure of an FEC Source Packet with the Explicit Source FEC Payload ID

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.

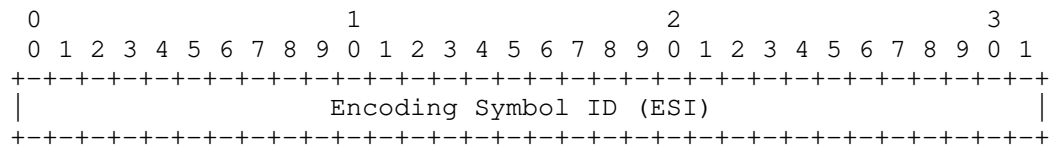


Figure 6: Source FEC Payload ID Encoding Format

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.

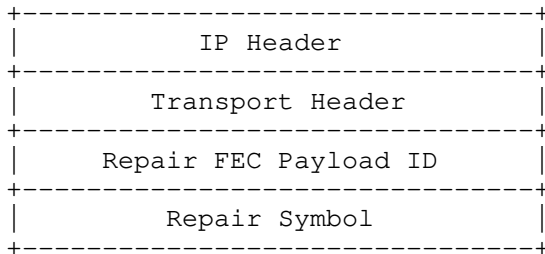


Figure 7: Structure of an FEC Repair Packet with the Repair FEC Payload ID

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;

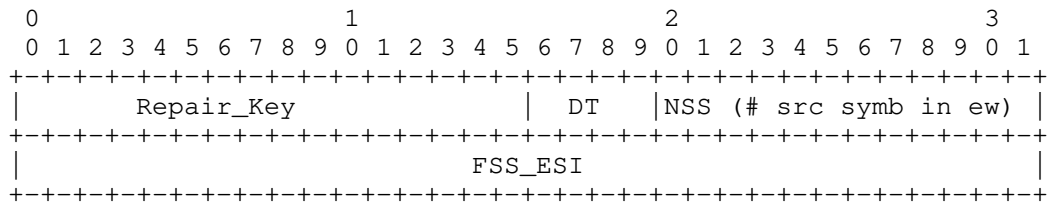


Figure 8: Repair FEC Payload ID Encoding Format

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 $GF(2^{2^8})$), 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.

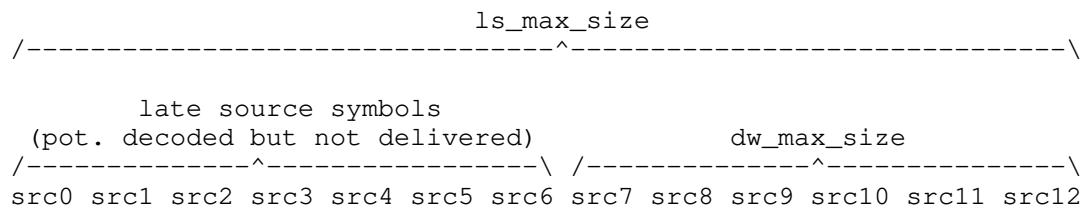


Figure 13: Relationship between parameters to decode beyond maximum latency.

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 28, 2015

M. Tuexen
Muenster Univ. of Appl. Sciences
R. Stewart
Netflix, Inc.
R. Jesup
WorldGate Communications
S. Loreto
Ericsson
January 24, 2015

DTLS Encapsulation of SCTP Packets
draft-ietf-tsvwg-sctp-dtls-encaps-09.txt

Abstract

The Stream Control Transmission Protocol (SCTP) is a transport protocol originally defined to run on top of the network protocols IPv4 or IPv6. This document specifies how SCTP can be used on top of the Datagram Transport Layer Security (DTLS) protocol. Using the encapsulation method described in this document, SCTP is unaware of the protocols being used below DTLS; hence explicit IP addresses cannot be used in the SCTP control chunks. As a consequence, the SCTP associations carried over DTLS can only be single homed.

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 July 28, 2015.

Copyright Notice

Copyright (c) 2015 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. Overview	2
2. Conventions	3
3. Encapsulation and Decapsulation Procedure	3
4. General Considerations	3
5. DTLS Considerations	4
6. SCTP Considerations	5
7. IANA Considerations	6
8. Security Considerations	6
9. Acknowledgments	7
10. References	7
Appendix A. NOTE to the RFC-Editor	9
Authors' Addresses	9

1. Overview

The Stream Control Transmission Protocol (SCTP) as defined in [RFC4960] is a transport protocol running on top of the network protocols IPv4 [RFC0791] or IPv6 [RFC2460]. This document specifies how SCTP is used on top of the Datagram Transport Layer Security (DTLS) protocol. DTLS 1.0 is defined in [RFC4347] and the latest version when this RFC was published, DTLS 1.2, is defined in [RFC6347]. This encapsulation is used for example within the WebRTC protocol suite (see [I-D.ietf-rtcweb-overview] for an overview) for transporting non-SRTP data between browsers. The architecture of this stack is described in [I-D.ietf-rtcweb-data-channel].

[NOTE to RFC-Editor:

Please ensure that the authors double check the above statement about DTLS 1.2 during AUTH48 and then remove this note before publication.

]

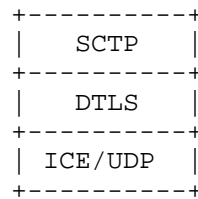


Figure 1: Basic stack diagram

This encapsulation of SCTP over DTLS over UDP or ICE/UDP (see [RFC5245]) can provide a NAT traversal solution in addition to confidentiality, source authentication, and integrity protected transfers. Please note that using ICE does not necessarily imply that a different packet format is used on the wire.

Please note that the procedures defined in [RFC6951] for dealing with the UDP port numbers do not apply here. When using the encapsulation defined in this document, SCTP is unaware about the protocols used below DTLS.

2. Conventions

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

3. Encapsulation and Decapsulation Procedure

When an SCTP packet is provided to the DTLS layer, the complete SCTP packet, consisting of the SCTP common header and a number of SCTP chunks, is handled as the payload of the application layer protocol of DTLS. When the DTLS layer has processed a DTLS record containing a message of the application layer protocol, the payload is passed to the SCTP layer. The SCTP layer expects an SCTP common header followed by a number of SCTP chunks.

4. General Considerations

An implementation of SCTP over DTLS MUST implement and use a path maximum transmission unit (MTU) discovery method that functions without ICMP to provide SCTP/DTLS with an MTU estimate. An implementation of "Packetization Layer Path MTU Discovery" [RFC4821] either in SCTP or DTLS is RECOMMENDED.

The path MTU discovery is performed by SCTP when SCTP over DTLS is used for data channels (see Section 5 of [I-D.ietf-rtcweb-data-channel]).

5. DTLS Considerations

The DTLS implementation MUST support DTLS 1.0 [RFC4347] and SHOULD support the most recently published version of DTLS, which was DTLS 1.2 [RFC6347] when this RFC was published. In the absence of a revision to this document, the latter requirement applies to all future versions of DTLS when they are published as RFCs. This document will only be revised if a revision to DTLS or SCTP makes a revision to the encapsulation necessary.

[NOTE to RFC-Editor:

Please ensure that the authors double check the above statement about DTLS 1.2 during AUTH48 and then remove this note before publication.

]

SCTP performs segmentation and reassembly based on the path MTU. Therefore the DTLS layer MUST NOT use any compression algorithm.

The DTLS MUST support sending messages larger than the current path MTU. This might result in sending IP level fragmented messages.

If path MTU discovery is performed by the DTLS layer, the method described in [RFC4821] MUST be used. For probe packets, the extension defined in [RFC6520] MUST be used.

If path MTU discovery is performed by the SCTP layer and IPv4 is used as the network layer protocol, the DTLS implementation SHOULD allow the DTLS user to enforce that the corresponding IPv4 packet is sent with the Don't Fragment (DF) bit set. If controlling the DF bit is not possible, for example due to implementation restrictions, a safe value for the path MTU has to be used by the SCTP stack. It is RECOMMENDED that the safe value does not exceed 1200 bytes. Please note that [RFC1122] only requires end hosts to be able to reassemble fragmented IP packets up to 576 bytes in length.

The DTLS implementation SHOULD allow the DTLS user to set the Differentiated services code point (DSCP) used for IP packets being sent (see [RFC2474]). This requires the DTLS implementation to pass the value through and the lower layer to allow setting this value. If the lower layer does not support setting the DSCP, then the DTLS user will end up with the default value used by protocol stack. Please note that only a single DSCP value can be used for all packets belonging to the same SCTP association.

Using explicit congestion notifications (ECN) in SCTP requires the DTLS layer to pass the ECN bits through and its lower layer to expose access to them for sent and received packets (see [RFC3168]). The implementation of DTLS and its lower layer have to provide this support. If this is not possible, for example due to implementation restrictions, ECN can't be used by SCTP.

6. SCTP Considerations

This section describes the usage of the base protocol and the applicability of various SCTP extensions.

6.1. Base Protocol

This document uses SCTP [RFC4960] with the following restrictions, which are required to reflect that the lower layer is DTLS instead of IPv4 and IPv6 and that SCTP does not deal with the IP addresses or the transport protocol used below DTLS:

- o A DTLS connection MUST be established before an SCTP association can be set up.
- o Multiple SCTP associations MAY be multiplexed over a single DTLS connection. The SCTP port numbers are used for multiplexing and demultiplexing the SCTP associations carried over a single DTLS connection.
- o All SCTP associations are single-homed, because DTLS does not expose any address management to its upper layer. Therefore it is RECOMMENDED to set the SCTP parameter `path.max.retrans` to `association.max.retrans`.
- o The INIT and INIT-ACK chunk MUST NOT contain any IPv4 Address or IPv6 Address parameters. The INIT chunk MUST NOT contain the Supported Address Types parameter.
- o The implementation MUST NOT rely on processing ICMP or ICMPv6 packets, since the SCTP layer most likely is unable to access the SCTP common header in the plain text of the packet, which triggered the sending of the ICMP or ICMPv6 packet. This applies in particular to path MTU discovery when performed by SCTP.
- o If the SCTP layer is notified about a path change by its lower layers, SCTP SHOULD retest the Path MTU and reset the congestion state to the initial state. The window-based congestion control method specified in [RFC4960], resets the congestion window and slow start threshold to their initial values.

6.2. Padding Extension

When the SCTP layer performs path MTU discovery as specified in [RFC4821], the padding extension defined in [RFC4820] MUST be supported and used for probe packets (HEARTBEAT chunks bundled with PADDING chunks [RFC4820]).

6.3. Dynamic Address Reconfiguration Extension

If the dynamic address reconfiguration extension defined in [RFC5061] is used, ASCONF chunks MUST use wildcard addresses only.

6.4. SCTP Authentication Extension

The SCTP authentication extension defined in [RFC4895] can be used with DTLS encapsulation, but does not provide any additional benefit.

6.5. Partial Reliability Extension

Partial reliability as defined in [RFC3758] can be used in combination with DTLS encapsulation. It is also possible to use additional PR-SCTP policies, for example the ones defined in [I-D.ietf-tsvwg-sctp-prpolicies].

6.6. Stream Reset Extension

The SCTP stream reset extension defined in [RFC6525] can be used with DTLS encapsulation. It is used to reset SCTP streams and add SCTP streams during the lifetime of the SCTP association.

6.7. Interleaving of Large User Messages

SCTP as defined in [RFC4960] does not support the interleaving of large user messages that need to be fragmented and reassembled by the SCTP layer. The protocol extension defined in [I-D.ietf-tsvwg-sctp-ndata] overcomes this limitation and can be used with DTLS encapsulation.

7. IANA Considerations

This document requires no actions from IANA.

8. Security Considerations

Security considerations for DTLS are specified in [RFC4347] and for SCTP in [RFC4960], [RFC3758], and [RFC6525]. The combination of SCTP and DTLS introduces no new security considerations.

SCTP should not process the IP addresses used for the underlying communication since DTLS provides no guarantees about them.

It should be noted that the inability to process ICMP or ICMPv6 messages does not add any security issue. When SCTP is carried over a connection-less lower layer like IPv4, IPv6, or UDP, processing of these messages is required to protect other nodes not supporting SCTP. Since DTLS provides a connection-oriented lower layer, this kind of protection is not necessary.

9. Acknowledgments

The authors wish to thank David Black, Benoit Claise, Spencer Dawkins, Francis Dupont, Gorrry Fairhurst, Stephen Farrell, Christer Holmberg, Barry Leiba, Eric Rescorla, Tom Taylor, Joe Touch and Magnus Westerlund for their invaluable comments.

10. References

10.1. Normative References

- [RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, April 2006.
- [RFC4820] Tuexen, M., Stewart, R., and P. Lei, "Padding Chunk and Parameter for the Stream Control Transmission Protocol (SCTP)", RFC 4820, March 2007.
- [RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.
- [RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.
- [RFC6520] Seggelmann, R., Tuexen, M., and M. Williams, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension", RFC 6520, February 2012.

10.2. Informative References

- [RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.
- [RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.
- [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, December 1998.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.
- [RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, May 2004.
- [RFC4895] Tuexen, M., Stewart, R., Lei, P., and E. Rescorla, "Authenticated Chunks for the Stream Control Transmission Protocol (SCTP)", RFC 4895, August 2007.
- [RFC5061] Stewart, R., Xie, Q., Tuexen, M., Maruyama, S., and M. Kozuka, "Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration", RFC 5061, September 2007.
- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.
- [RFC6525] Stewart, R., Tuexen, M., and P. Lei, "Stream Control Transmission Protocol (SCTP) Stream Reconfiguration", RFC 6525, February 2012.
- [RFC6951] Tuexen, M. and R. Stewart, "UDP Encapsulation of Stream Control Transmission Protocol (SCTP) Packets for End-Host to End-Host Communication", RFC 6951, May 2013.
- [I-D.ietf-rtcweb-overview] Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-13 (work in progress), November 2014.

[I-D.ietf-rtcweb-data-channel]

Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channels", draft-ietf-rtcweb-data-channel-13 (work in progress), January 2015.

[I-D.ietf-tsvwg-sctp-prpolicies]

Tuexen, M., Seggelmann, R., Stewart, R., and S. Loreto, "Additional Policies for the Partial Reliability Extension of the Stream Control Transmission Protocol", draft-ietf-tsvwg-sctp-prpolicies-06 (work in progress), December 2014.

[I-D.ietf-tsvwg-sctp-ndata]

Stewart, R., Tuexen, M., Loreto, S., and R. Seggelmann, "Stream Schedulers and a New Data Chunk for the Stream Control Transmission Protocol", draft-ietf-tsvwg-sctp-ndata-02 (work in progress), January 2015.

Appendix A. NOTE to the RFC-Editor

Although the references to [I-D.ietf-tsvwg-sctp-prpolicies] and [I-D.ietf-tsvwg-sctp-ndata] are informative, put this document in REF-HOLD until these two references have been approved and update these references to the corresponding RFCs.

Authors' Addresses

Michael Tuexen
Muenster University of Applied Sciences
Stegerwaldstrasse 39
48565 Steinfurt
DE

Email: tuexen@fh-muenster.de

Randall R. Stewart
Netflix, Inc.
Chapin, SC 29036
US

Email: randall@lakerest.net

Randell Jesup
WorldGate Communications
3800 Horizon Blvd, Suite #103
Trevose, PA 19053-4947
US

Phone: +1-215-354-5166
Email: randell_ietf@jesup.org

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
FI

Email: Salvatore.Loreto@ericsson.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.

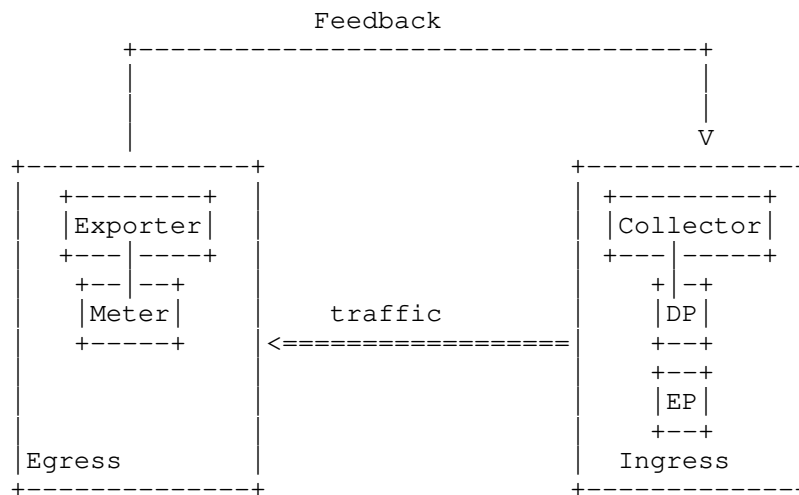


Figure 1: Feedback Model.

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:

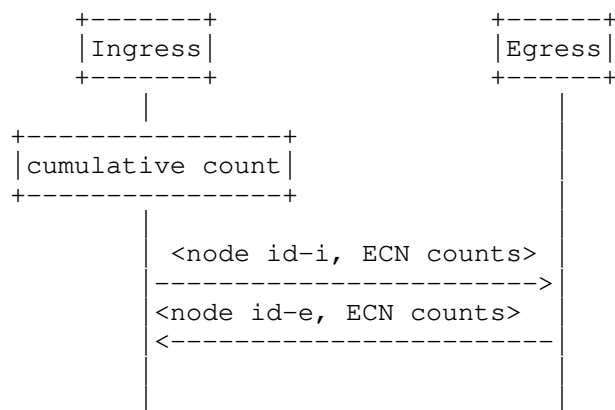


Figure 2: Procedure of Packet Loss Measurement

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.

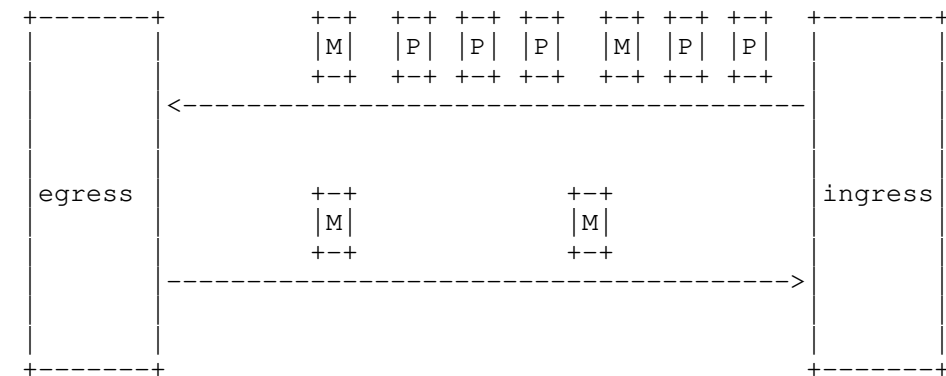
Figure 3 to Figure 6 below show the example procedure between ingress and egress.

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

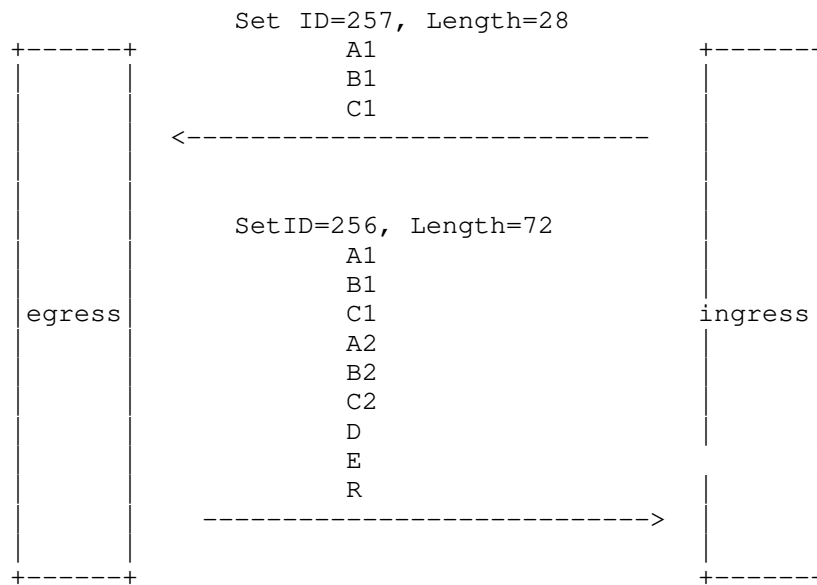


Figure 6: Message Between Ingress and Egress

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:

$$\text{UDP_Length} = \text{IPv4_Total_Length} - \text{IPv4_IHL} * 4$$

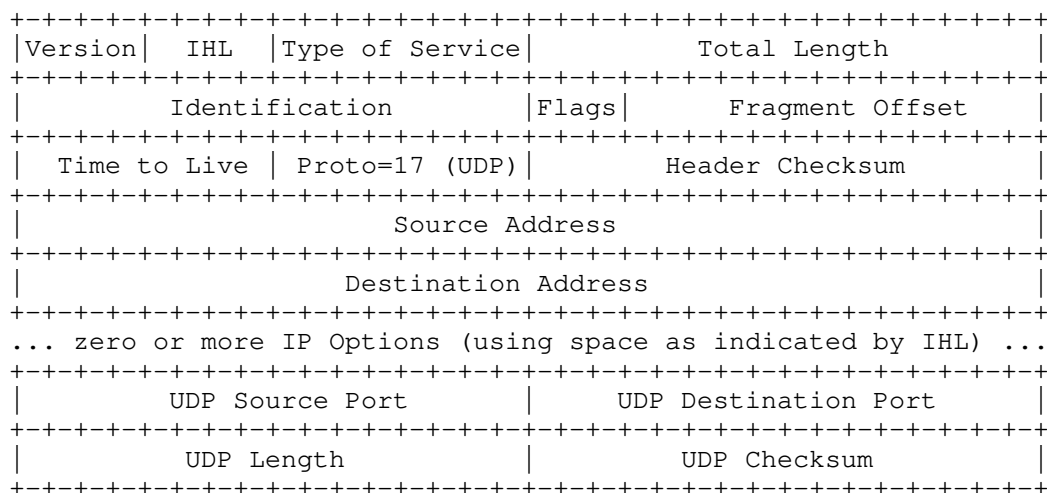


Figure 1 IPv4 datagram with UDP header

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:

$$\text{UDP_Length} = \text{IPv6_Payload_Length} - \text{sum}(\text{extension header lengths})$$

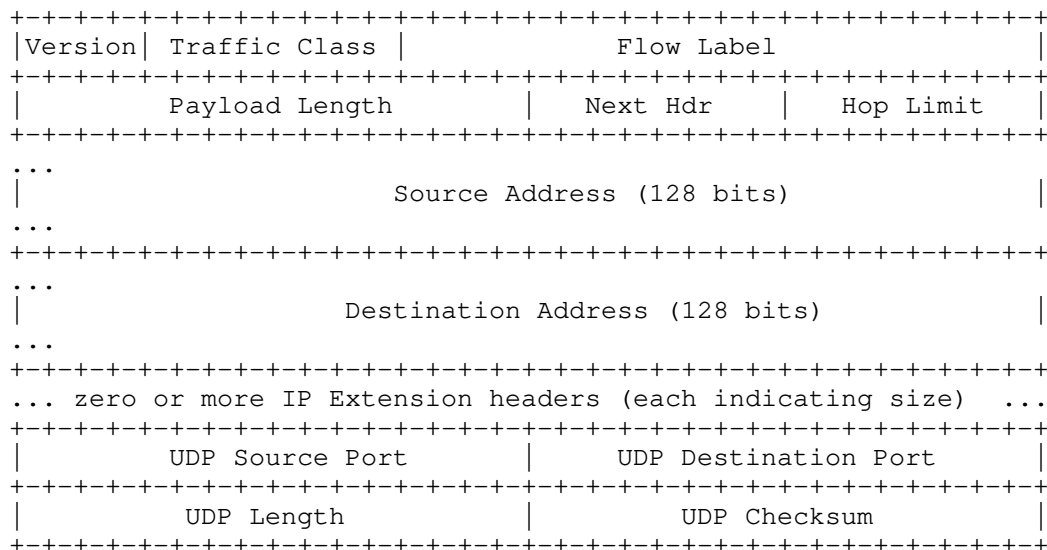


Figure 2 IPv6 datagram with UDP header

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).

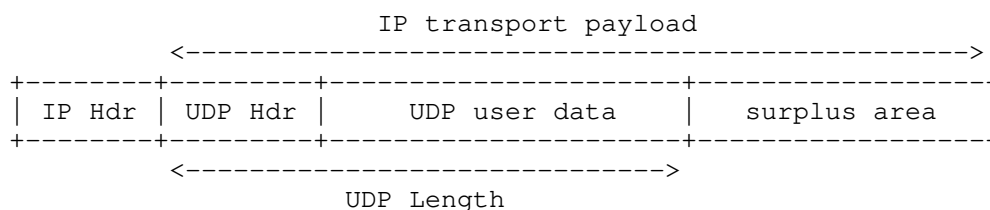


Figure 3 IP transport payload vs. UDP Length

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.

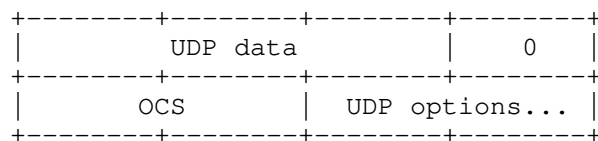


Figure 4 UDP OCS format, here using one zero for alignment

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.

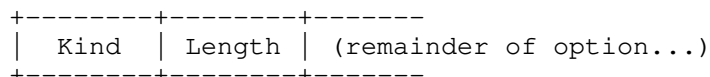


Figure 5 UDP option default format

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.

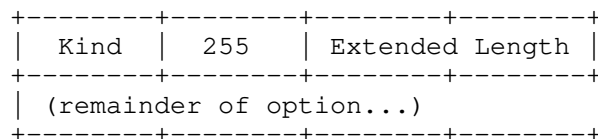


Figure 6 UDP option extended format

>> 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].

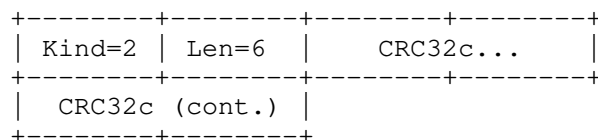


Figure 9 UDP APC option format

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.

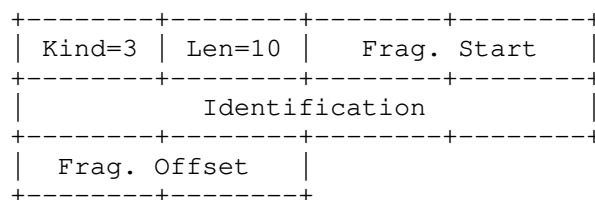


Figure 10 UDP non-terminal FRAG option format

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.

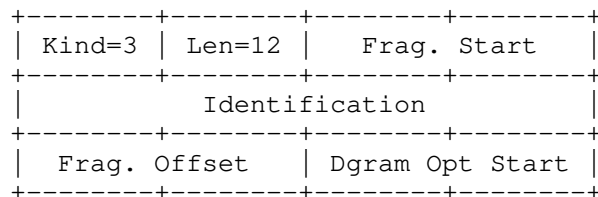


Figure 11 UDP terminal FRAG option format

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.

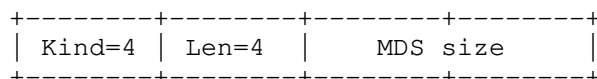


Figure 12 UDP MDS option format

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.

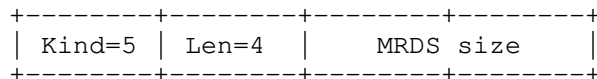


Figure 13 UDP MRDS option format

>> 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.

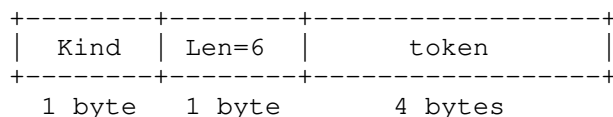


Figure 14 UDP REQ and RES options format

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].

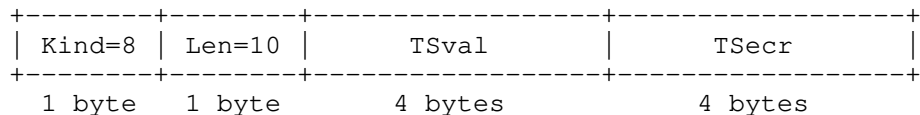


Figure 15 UDP TIME option format

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.

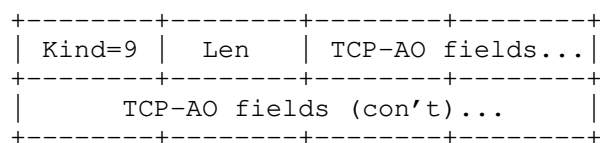


Figure 16 UDP AUTH option format

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

TSVWG
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2018

P. Thubert, Ed.
Cisco
October 30, 2017

A Transport Layer for Deterministic Networks
draft-thubert-tsvwg-detnet-transport-01

Abstract

This document specifies the behavior of a Transport Layer operating over a Deterministic Network and implementing a DetNet Service Layer and a Northbound side of the DetNet User-to-Network Interface.

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 3, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. Terminology	5
3. On Deterministic Networking	5
3.1. Applications and Requirements	5
3.2. The DetNet User-to-Network Interface (UNI)	7
3.3. The DetNet Stack	8
3.4. The DetNet Service Model	8
4. DetTrans Operations	9
4.1. DetTrans Overview	9
4.2. Application Requirements	9
4.2.1. Packet Normalization	9
4.2.2. Packet Streaming	10
4.3. Deterministic Flow Services	10
4.3.1. Deterministic Flows	10
4.3.2. Deterministic Flow Encapsulation and Stitching	11
4.3.2.1. Flow Stitching	11
4.3.2.2. Load Sharing	11
4.3.2.3. Flow Aggregation	12
4.3.3. Deterministic Service Protection	13
4.3.3.1. PRE vs. 1+1 Redundancy	13
4.3.3.2. Network Coding	13
4.3.3.3. Multipath DetTrans Services	13
5. The DetNet-UNI	14
5.1. Local Loop Flow Control	16
5.1.1. Dichotomy of a DetNet End System	16
5.1.2. Local Loop Location	17
5.1.3. Network Pull vs. Rate Based Flow Control	18
5.2. DetNet-UNI Protocol Exchanges	18
5.2.1. the "More" Message	18
5.2.2. the "Time-Correction" Message	19
5.2.3. Loss of a Control Message	19
6. Security Considerations	20
7. IANA Considerations	20
8. Acknowledgments	20
9. Informative References	20
Author's Address	22

1. Introduction

Over last twenty years, voice, data and video networks have converged to digital over IP. Mail delivery has become quasi-immediate and volumes have multiplied; long distance voice is now mostly free and the videophone is finally a reality; TV is available on-demand and games became interactive and massively multi-player. The convergence of highly heterogeneous networks over IP resulted in significant drops in price for the end-user while adding new distinct value to

the related services. Yet, and even though similar benefits can be envisioned when converging new applications over the Internet, there are still many disjoint branches in the networking family tree, many use-cases where mission-specific applications continue to utilize dedicated point-to-point analog and digital technologies for their operations.

Forty years ago, Control Information was first encoded as an analog modulation of current (typically 4 to 20 mA) that can be carried virtually instantly and with no loss over a distance. Then came digitization, which enabled to multiplex data with the control signal and manage the devices, but at the same time introduced latency to industrial processes, the necessary delay to encode a series of bits on a link and transport them along, which in turn may limit the amount of transported information. The need to save cable and simplify wiring lead to the Time Division Multiplexing (TDM) of signals from multiple devices over shared digital buses, each signal being granted access to the medium at a fixed period for a fixed duration; with TDM, came more latency, waiting for the next reserved access time. Statistical multiplexing, with Ethernet and IP, was then introduced to achieve higher speeds at lower cost, and with it came jitter and congestion loss.

A number of Operational Technology (OT) applications are now migrating to Ethernet and IP, but that comes at the expense of additional latency for the flows, to compensate for the degradation of the transport discussed above. This also comes at the expense of additional complexity in particular, applications may need to transport a sense of time, provide some Forward Error Correction (FEC) and include a jitter absorption buffer. for that reason, many applications were never ported and OT networks are still largely operated on point-to-point serial links and TDM buses.

A sense of what Deterministic Networking is has emerged as the capability to make the Application simple again and enable a larger migration of existing applications by absorbing the complexity lower in the stack, at the Transport, Network and Link layers. A Deterministic Network should be capable to emulate point-to-point wires over a packet network, sharing the network resources between deterministic and non-deterministic flows in such a fashion that there can no observable influence whatsoever on a deterministic flow from any other flow, regardless of the load of the network.

The generalization of the needs for more deterministic networks have led to the IEEE 802.1 AVB Task Group becoming the Time-Sensitive Networking (TSN) [IEEE802.1TSNTG] Task Group (TG), with a much-expanded constituency from the industrial and vehicular markets. In order to address the problem at the network layer, the DetNet Working

Group was formed to specify the signaling elements to be used to establish a path and the tagging elements to be used identify the flows that are to be forwarded along that path.

The "Deterministic Networking Use Cases" [I-D.ietf-detnet-use-cases] indicates that beyond the classical case of industrial automation and control systems (IACS), there are in fact multiple industries with strong and yet relatively similar needs for deterministic network services such as latency guarantees and ultra-low packet loss. The "Deterministic Networking Problem Statement" [I-D.ietf-detnet-problem-statement] documents the specific requirements for the use of routed networks to support these applications and the "Deterministic Networking Architecture" [I-D.ietf-detnet-architecture] introduces the model that must be proposed to integrate determinism in IT technology.

A DetNet network will guarantee a bounded latency and a very low packet loss as long as the incoming flows respect a certain Service Level Agreement (SLA), as typically expressed in the form of a maximum packet size, a time window of observation and a maximum number of packets per time window.

Outside the scope of DetNet, the IETF will also need to specify the necessary protocols, or protocol additions, based on relevant IETF technologies, to enable end-to-end deterministic flows. One critical element is the Deterministic Transport Layer (DetTrans) that adapts the flows coming from the Application Layer to the SLA of the DetNet Network and provide end-to-end guarantees such as loss, latency and timeliness.

The DetTrans Layer should in particular ensure that:

- o the Deterministic Network setup matches the needs of the Application
- o the Application flows are presented to the Deterministic Network in accordance to the SLA regardless of the way the data is passed from the application
- o the use of the network is optimized so as to ensure that every byte from the application can effectively be transported
- o the application flow is delivered reliably and with a bounded latency to the other Transport End Point, which may imply a FEC technique such as Network Coding, Packet Replication and Elimination (PRE), or basic 1+1 redundancy.

- o the full of the application flow is served, which may require the use of multiple reservations in parallel, and the reordering of the flows

On the one hand, the Deterministic Network will typically guarantee a constant rate, so the classical Transport feature of flow control will not be needed in a Deterministic Transport. On the other hand, the Application and Transport layers may not reside in the same device as the DetNet Router and/or the IEEE Std. 802.1 TSN Bridge that acts as ingress point to the Deterministic Network. It results that a minimum reliability and flow control must take place over the Local Loop between these devices to ensure that the Deterministic Network is kept optimally fed, meaning that packets are received just in time for their scheduled transmission opportunities.

2. Terminology

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

3. On Deterministic Networking

3.1. Applications and Requirements

The Internet is not the only digital network that has grown dramatically over the last 30-40 years. Video and audio entertainment, and control systems for machinery, manufacturing processes, and vehicles are also ubiquitous, and are now based almost entirely on digital technologies. Over the past 10 years, engineers in these fields have come to realize that significant advantages in both cost and in the ability to accelerate growth can be obtained by basing all of these disparate digital technologies on packet networks.

The goals of Deterministic Networking are to enable the migration of applications that use special-purpose fieldbus technologies (HDMI, CANbus, ProfiBus, etc... even RS-232!) to packet technologies in general, and the Internet Protocol in particular, and to support both these new applications, and existing packet network applications, over the same physical network.

Considerable experience ([ODVA]/[EIP], [AVnu], [Profinet],[HART], [IEC62439], [ISA100.11a] and [WirelessHART], etc...) has shown that these applications need a some or all of a suite of deterministic features.

That suite of deterministic features includes:

1. Time synchronization of all Host and network nodes (Routers and/or Bridges), accurate to something between 10 nanoseconds and 10 microseconds, depending on the application.
2. Support for critical packet flows that:
 - * Can be unicast or multicast;
 - * Need absolute guarantees of minimum and maximum latency end-to-end across the network; sometimes a tight jitter is required as well;
 - * Need a packet loss ratio beyond the classical range for a particular medium, in the range of 10^{-9} to 10^{-12} , or better, on Ethernet, and in the order of 10^{-5} in Wireless Sensor Mesh Networks;
 - * Can, in total, absorb more than half of the network's available bandwidth (that is, massive over-provisioning is ruled out as a solution);
 - * Cannot suffer throttling, flow control, or any other network-imposed latency, for flows that can be meaningfully characterized either by a fixed, repeating transmission schedule, or by a maximum bandwidth and packet size;
3. Multiple methods to schedule, shape, limit, and otherwise control the transmission of critical packets at each hop through the network data plane;
4. Robust defenses against misbehaving Hosts, Routers, or Bridges, both in the data and control planes, with guarantees that a critical flow within its guaranteed resources cannot be affected by other flows whatever the pressures on the network;
5. One or more methods to reserve resources in Bridges and Routers to carry these flows.

Robustness is a common need for networking protocols, but plays a more important part in real-time control networks, where expensive equipment, and even lives, can be lost due to misbehaving equipment. Reserving resources before packet transmission is the one fundamental shift in the behavior of network applications that is impossible to avoid. In the first place, a network cannot deliver finite latency and practically zero packet loss to an arbitrarily high offered load. Secondly, achieving practically zero packet loss for un-throttled (though bandwidth limited) flows means that Bridges and Routers have to dedicate buffer resources to specific flows or to classes of

flows. The requirements of each reservation have to be translated into the parameters that control each Host's, Bridge's, and Router's queuing, shaping, and scheduling functions and delivered to the Hosts, Bridges, and Routers.

3.2. The DetNet User-to-Network Interface (UNI)

The "Deterministic Networking Architecture" [I-D.ietf-detnet-architecture] presents the end-to-end networking model and the DetNet services; in particular, it depicts the DetNet User-to-Network Interfaces (DetNet-UNIs) ("U" in Figure 1) between the Edge nodes (PE) of the Deterministic Network and the End Systems. These UNIs are assumed to be packet-based reference points and provide connectivity over the packet network. The Architecture also mentions internal reference points between the Central Processing Unit (CPU) and the Network Interface Card (NIC) in the End System. The DetNet-UNIs provide congestion protection services and belong to the DetNet Transport Layer.

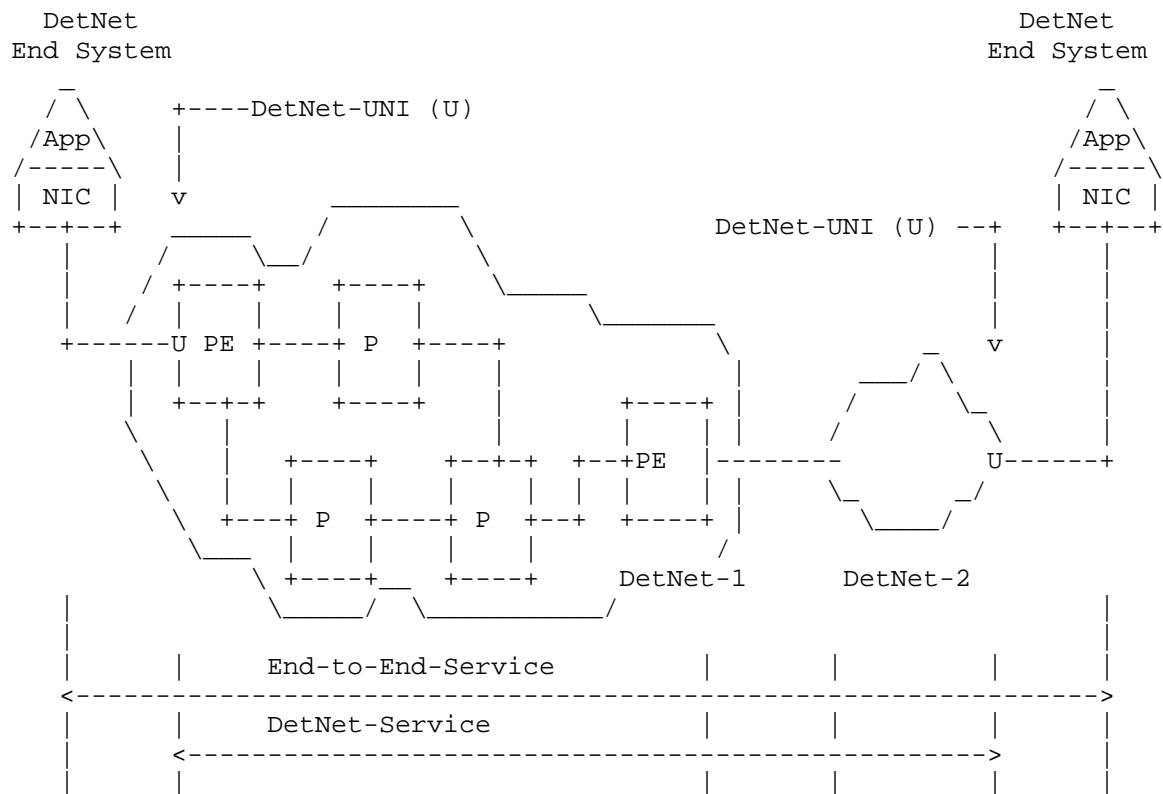


Figure 1: DetNet Service Reference Model (multi-domain)

A specific hardware is necessary for the time-sensitive functions of synchronization, shaping and scheduling. This hardware may or may not be fully available on a NIC inside the Host system. This specification makes a distinction between a fully DetNet-Capable NIC, and a DetNet-Aware NIC that participates to the DetNet-UNI, but is not synchronized and scheduled with the Deterministic Network.

3.3. The DetNet Stack

The "Deterministic Networking Architecture" [I-D.ietf-detnet-architecture] presents a conceptual DetNet data plane layering model. The protocol stack includes a Service Layer and a Transport Layer and is illustrated in Figure 2.

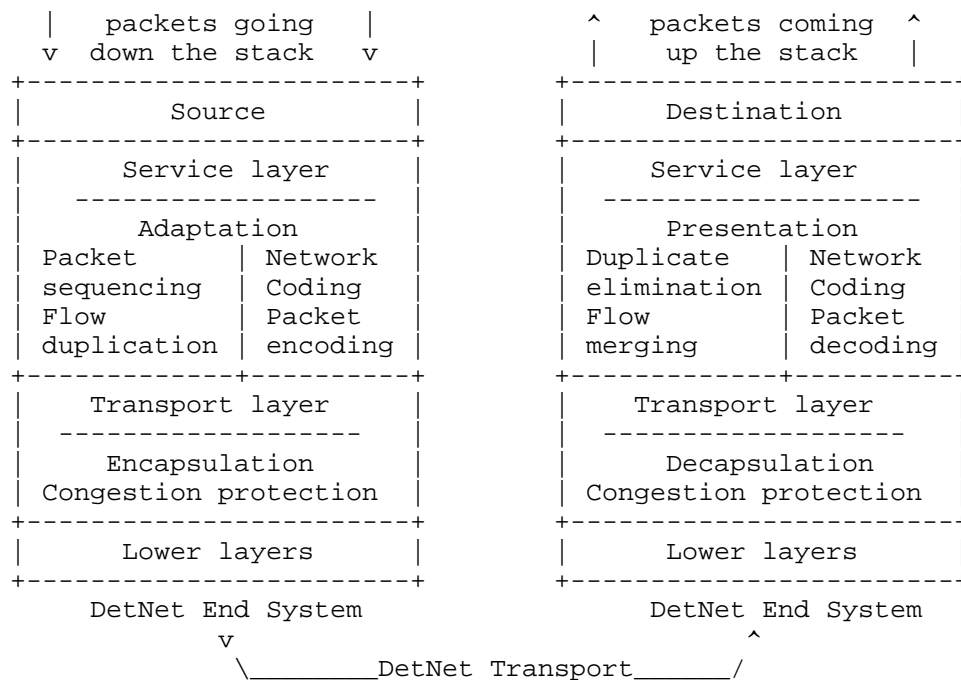


Figure 2: DetNet-Capable End-System Protocol Stack

3.4. The DetNet Service Model

The "DetNet Service Model" [I-D.varga-detnet-service-model] provides more details on the distribution of DetNet awareness and services.

4. DetTrans Operations

4.1. DetTrans Overview

The DetNet Service Layer mostly operates between the end-points, though it is possible that some operations such as Packet Replication and Elimination are also performed in selected intermediate nodes. The DetNet Transport Layer represents the methods that ensure that a packet is deterministically forwarded hop-by-hop from a Detnet Relay to the next. The term "Transport" in the DetNet terminology must not be confused with the function described in this document. This document defines Detrans as a Layer-4 operation and an IETF Transport Layer; DetTrans provides DetNet End-To-End Services for its Applications, as well as intermediate services in selected points.

Following the DetNet Architecture, DetTrans mostly corresponds to the DetNet Service Layer and its interface with the Detnet Transport Layer for congestion protection services through the DetNet_UNI, as well as for encapsulation and decapsulation services. Compared to a traditional IETF Transport Layer, DetTrans performs similar operation of end-to-end reliability, flow control and multipath load sharing, but differs on how those functionalities are achieved.

Architectural variations are also introduced, for instance:

- o Multipath operations are not necessarily end-to-end and a DetTrans function may be present inside the network to relay between N parallel paths and M parallel path, and or perform reliability functionality such as Packet Replication and Elimination.
- o The flow control is only needed between the DetTrans Layer and the first Deterministic Transit or Relay Node, for instance a DetNet Router or an IEEE Std. 802.1 TSN Bridge. From that point on, the flow is strictly controlled by the DetNet operation. Architecturally speaking, the flow control does not belong to the DetNet Service Layer but to the DetNet Transport Layer, which means that this specification also defines a sublayer from the DetNet Transport Layer for DetNet-UNI operations.

4.2. Application Requirements

4.2.1. Packet Normalization

A typical SLA for DetNet must be simple, for instance a maximum packet size, and a maximum number of packets per window of time. Smaller packets will mean wasted bandwidth, and excess packets within a time window will be destroyed by the ingress shaping at the first DetNet Bridge or Router.

The way the application layer feed the DetTrans layer may not necessarily match the SLA with the Deterministic Network and in order to provide the expected service, the DetTrans layer must pack the data in packets that are as close to the maximum packet size as possible, and yet make them available for transmission before scheduled time.

4.2.2. Packet Streaming

The DetTrans Layer operates on its own sense of time which may be loosely connected to the shared sense of time in the Deterministic Network.

The DetTrans layer must shape its transmissions so as to ensure that packets are delivered just in time to be injected along schedule in the Deterministic Network.

4.3. Deterministic Flow Services

4.3.1. Deterministic Flows

Deterministic forwarding can only apply on flows with well-defined characteristics such as periodicity and burstiness. Before a path can be established to serve them, the expression of those characteristics, and how the network can serve them, for instance in shaping and forwarding operations, must be specified.

At the time of this writing, the distinction between application layer flows and lower layer flows is not clearly stated in the "Deterministic Networking Architecture" [I-D.ietf-detnet-architecture]. For the purpose of this document, we use the term Deterministic End-to-End Service Flow (DEESF), or DetTrans Flow, to refer to an end-to-end application flow, and the term Deterministic Service Flow (DSF), or DetNet Flow, to refer to a lower layer deterministic transport. This is illustrated in Figure 3.

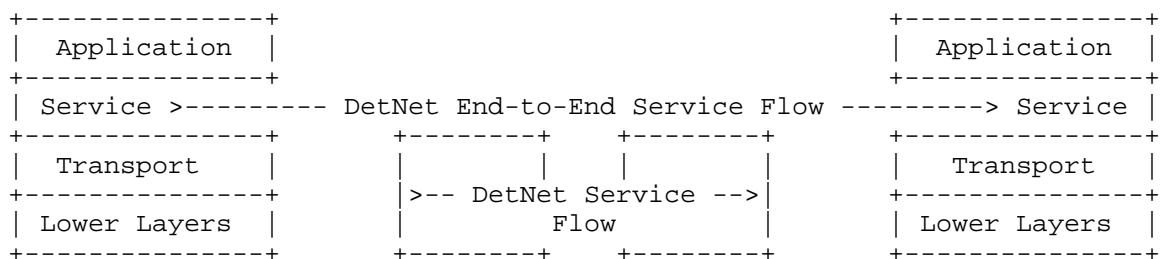


Figure 3: DetTrans vs. DetNet Flows

An application flow is established end-to-end between the DetTrans layers and uses one or more lower-layer deterministic flows either in parallel or in serial modes.

At Application and DetTrans Layers, the characteristics of a flow relate to aggregate properties such as throughput, loss, and traffic shape, and the Traffic Specification (TSPEC) is expressed as a Constant Bit Rate (CBR) or a Variable Bit Rate (VBR), burstiness (e.g. video I-Frames), reliability (e.g. five nines), worst case latency, amount of data to transfer, and expected duration of the flow.

At the DetNet Transport Layer (between Relays), metrics are very different, and relate to immediate actions on a packet as opposed to general characteristics of a flow. DetNet Transport Layer characteristics include time sync precision, time interval between packets, packet size, jitter, and number of packets per window of time. This is how the network SLA is defined, but this is not the native terms for the application and a complex mapping must ensure that the path that is setup and the DetNet Transport Layer effectively matches the requirements from the DetNet Services Layer and above.

4.3.2. Deterministic Flow Encapsulation and Stitching

4.3.2.1. Flow Stitching

The DetNet encapsulation and decapsulation of one-in-one, one-in-many and many-in-one Deterministic flows belongs to the DetNet Transport Layer. Direct one-in-one flow stitching also belongs to the DetNet Transport Layer. This happens when a deterministic flow can be directly bridged into another, resource-to-resource, without the need of an upper layer adaptation such as service protection from the Service Layer. A Detnet End-to-End Service flow may be stitched into one Detnet Service flow, or encapsulated in one or multiple Detnet Service flows.

4.3.2.2. Load Sharing

Load Sharing refers to the encapsulation of a DetNet Flow in more than one DetNet flows, for instance using multiple small and more manageable DetNet Service Flows in parallel to carry a large Deterministic End-to-End Service Flow, in order to avoid the need to periodically defragment the network. Packets are sequenced at the DetTrans Layer and distributed over the DetNet Transports paths in accordance to their relative capacities. In case of inconsistent jitter and Latency characteristics, packets may need to be reordered at the receiving DetTrans Layer based on the DSF Sequence.

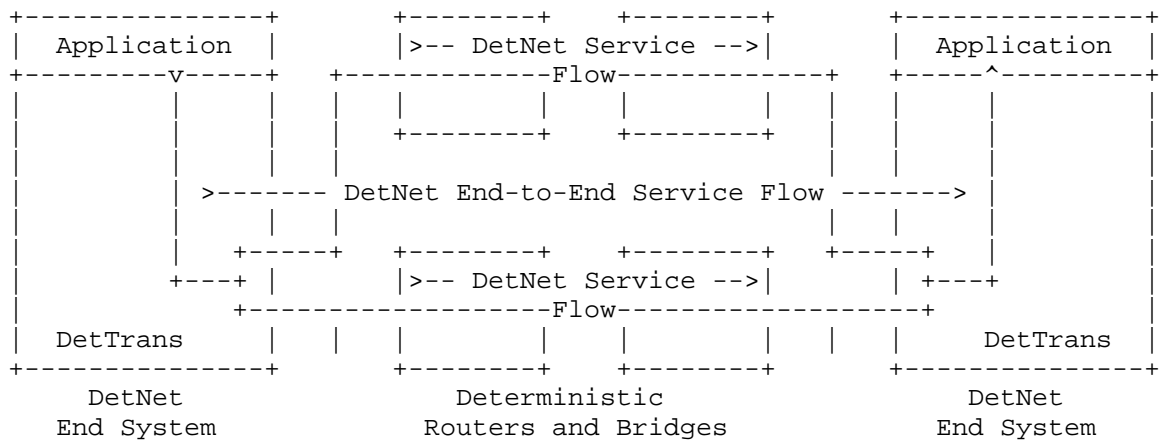


Figure 4: Load Sharing

In order to achieve this function, a Load Distribution function is required at the source and a Re-Ordering Function is required at the destination DetTrans End Point.

4.3.2.3. Flow Aggregation

Flow Aggregation refers to the encapsulation of more than one DetNet flows in one DetNet Flow, for instance using one large and long-lived DetNet Service Flow from a third party provider to carry multiple more dynamic Deterministic End-to-End Service Flows across domains. Packets are sequenced at the DetTrans Layer and distributed over the DetNet Transports paths in accordance to their relative capacities. In case of inconsistent jitter and Latency characteristics, packets may need to be reordered at the receiving DetTrans Layer based on the DSF Sequence.

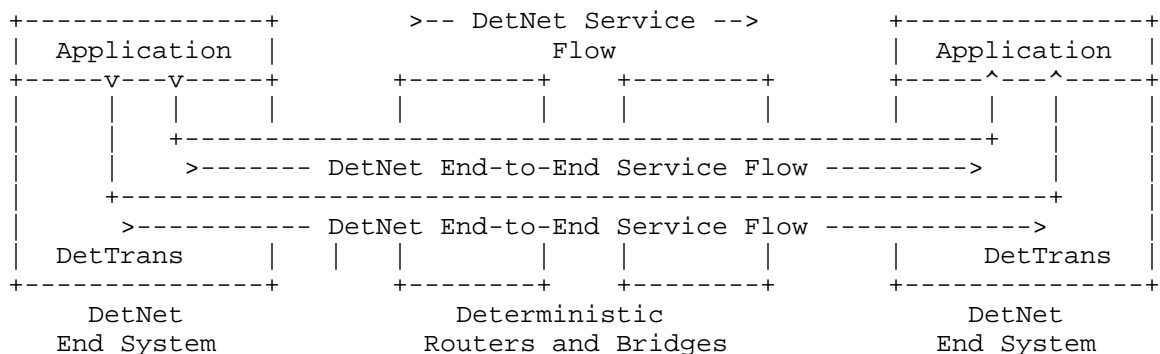


Figure 5: Flow Aggregation

In order to achieve this function, a multiplexing function is required at the source and a demultiplexing function is required at the destination DetTrans End Point.

4.3.3. Deterministic Service Protection

4.3.3.1. PRE vs. 1+1 Redundancy

The DetNet Flows may also be used for Packet Replication and Elimination, in which case an elimination function is required at the DetTrans Termination.

1+1 Redundancy refers to injecting identical copies of a packet at the ingress of two non-congruent paths, and eliminating the excess copy when both are received at the egress of the paths. Packet Replication and Elimination extends the concept by enabling more than 2 paths, and allowing non-end-to-end redundant paths with intermediate Replication and Elimination points.

4.3.3.2. Network Coding

Redundancy and Load Sharing may be combined with the use of Network Coding whereby a coded packet may carry redundancy information for previous data packet and cover the loss of one, in which case the recovery function is required at the other DetTrans End Point. Network Coding provides a Forward Error Correction between multiple packets or multiple fragments of a packet. It may be used at the DSF layer to enable an efficient combination of redundancy and load sharing.

4.3.3.3. Multipath DetTrans Services

A DetTrans Flow may leverage multiple DetNet Flows in parallel in order to achieve its requirements in terms of reliability and Aggregate throughput. The "Deterministic Networking Architecture" [I-D.ietf-detnet-architecture] clearly states that the capability of Replication and Elimination is not limited to the DetNet End Systems. DetNet Relay Nodes that operate DetTrans but then relay the packets are needed when the DetTrans operations are not end-to-end.

It may be that the DetTrans flow may need to traverse different domains where those Services are operated differently, e.g. controlled by different controllers or leveraging different technologies. It may also be that the bandwidth that is required is only available one segment at a time, and that for each segment, a different number of DetNet flows must be setup to transport the full amount of the DetTrans flow.

Figure 6 illustrates an example of the latter case, whereby The DetTrans Flow is distributed over two DetNet Flows, maybe operating PRE, then over three DetNet Flows, for instance operating Network Coding between them but using a smaller bandwidth for each flow, and then two DetNet Flows again.

DetTrans is needed at the interconnection points to adapt the flows, recover losses and reinject the appropriate rates in the next segment.

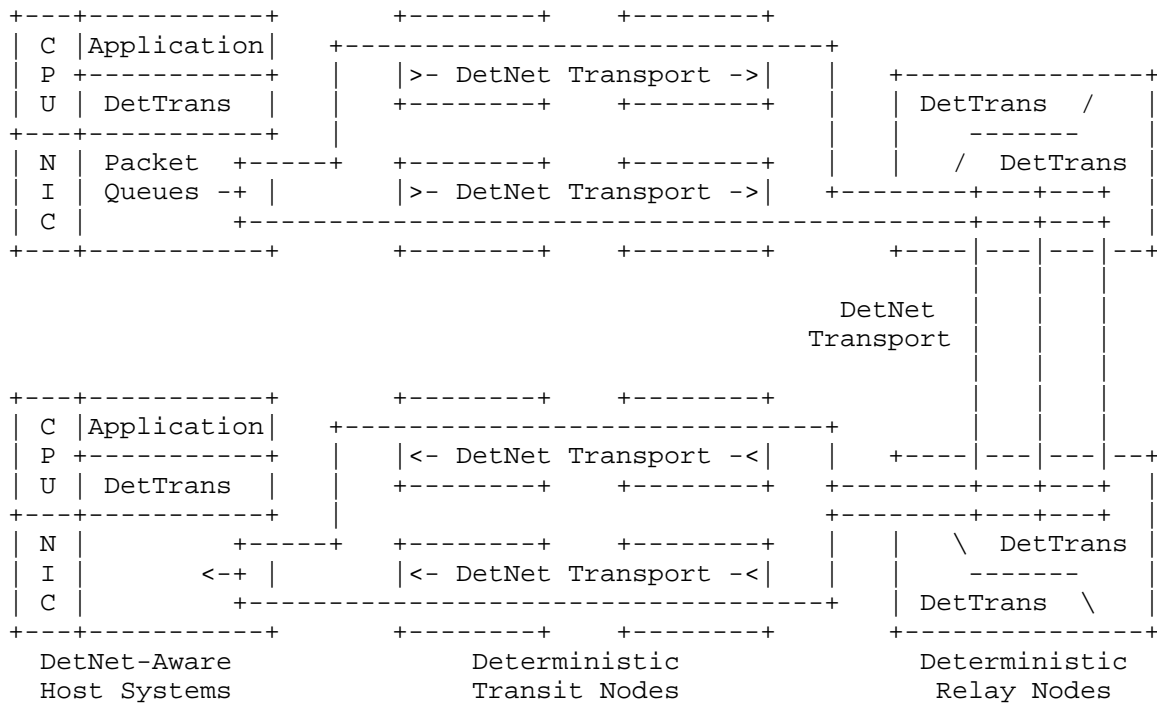


Figure 6: Intermediate Systems

5. The DetNet-UNI

Figure 7 illustrates a simple example of classical networked devices implementing the DetNet architecture. In that example, applications reside on Host systems and run on main CPUs; DetTrans is collocated with its applications and provides them with a Deterministic Service through DetTrans APIs. NICs provides the connectivity to the Deterministic Routers or Bridges acting at DetNet Edge and Relay Nodes - say as an example that they are IEEE Std. 802.1 TSN Bridges.

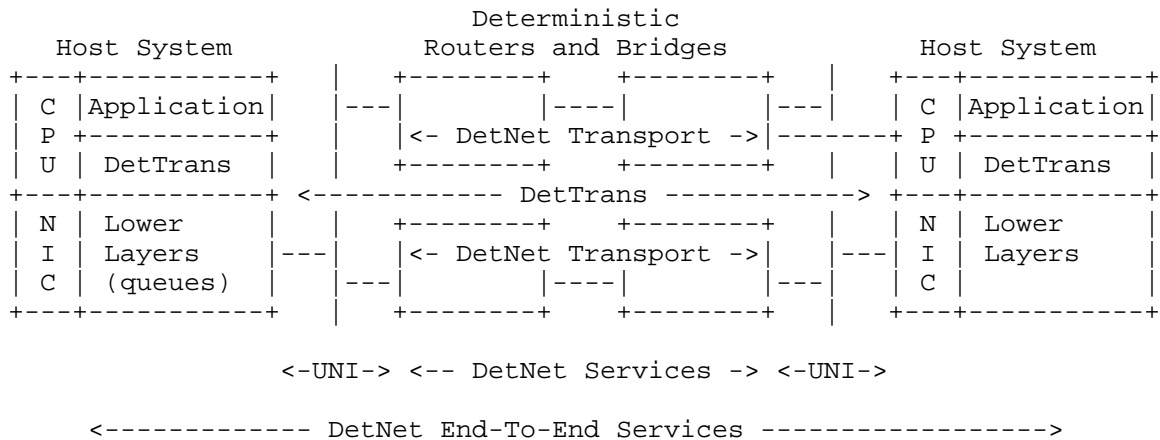


Figure 7: Example Physical Network

The DetTrans Layer aggregates the data coming from the application up to a maximum frame size that is part of the SLA with the DetNet Transport. Packets thus formed can be distributed over any of multiple DetNet Transport sessions that are defined to accept that packet size. Packets formed at the DetTrans Layer are queued and ready to be delivered through the DetNet-UNI either with a Rate-Based or a Network-Pull mechanism.

If the NIC is DetNet-Aware then the queue can be offboarded to the NIC and it can be drained with a time gate (Rate-Base) or a message-driven gate (Network-Pull). Else, the queue is handled by the CPU and hopefully it can be drained within an interrupt, either for a timer (Rate-Base) or for a message (Network-Pull).

The DetNet-UNI protocol enables the DetNet transport ingress point to control when the DetTrans Layer transmits its Data packets. It may happen that the DetTrans Layer has not formed a fully-sized packet when time comes for sending it, in which case the packet will be sent with a size below the maximum.

The DetNet UNI uses ICMPv6 to carry its protocol elements. Data Packets across the UNI are encapsulated in order to carry DetNet-UNI control information to identify the reason of a loss or a delay, and determine the action to be taken in case of a packet lost or delayed over the interface.

5.1. Local Loop Flow Control

5.1.1. Dichotomy of a DetNet End System

The logical DetNet End System depicted in Figure 2 comprises several elements which may implemented in one or separate physical Systems. The example dichotomy in Figure 3 segregates ingress shaping and DetNet Relay functions, which are performed by IEEE Std. 802.1 TSN Bridges, from a DetNet-Aware Host.

Hosts and Edge Bridges are connected over Ethernet and together they form a DetNet End System. As it goes, this example introduces a further dichotomy within the Host, between the CPU and the NIC, across a local bus such as PCI, as illustrated in Figure 8.

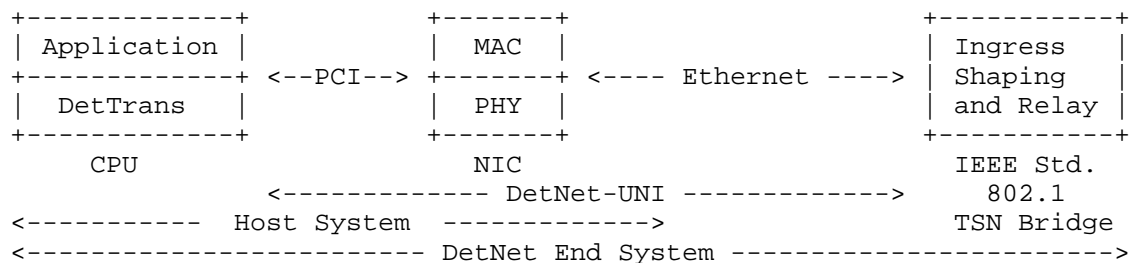


Figure 8: Chained Functions

The NICs in the Host System may not participate to the network time Synchronization and may not be aware of the DetNet protocols running between the Deterministic Routers and Bridges, and the associated scheduling rules. In that situation, the DetNet-UNI operates on a Local Loop to ensure that a packet that leaves the Transport reaches the Router or Bridge just in time for injection into the Deterministic data plane and to provide a flow control that avoids congestion loss at the interface.

It is also possible that the NIC participates to the Deterministic Network but still has asynchronous communication with DetTrans Layer running on the the CPU. Either way, a flow control over a local loop must be implemented to drain the queues from the DetTrans layer and feed the network just in time for the deterministic transmission.

Depending on the level of support by the NIC, the loop may be placed on a different interface but remains functionally the same.

5.1.2. Local Loop Location

If the NIC is not aware at all of DetNet, then it is a plain pipe for the Deterministic Traffic. The Local Loop operates between the Edge TSN Bridge and the CPU as illustrated in Figure 9.

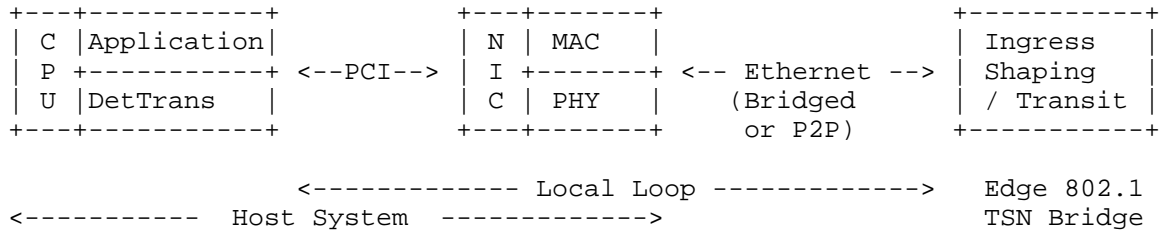


Figure 9: DetNet Unaware NIC

If the NIC is fully DetNet-Capable and participates to the deterministic Network including time synchronization and scheduling, then the local loop operates between the CPU and the NIC as illustrated in Figure 10.

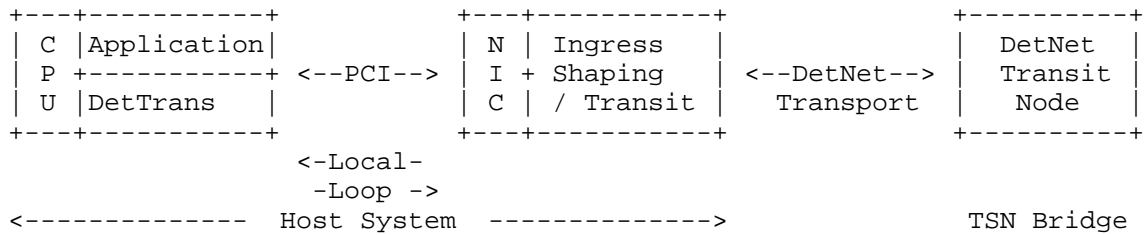


Figure 10: DetNet Capable NIC

If the NIC is DetNet-Aware and does not participate to the deterministic Network including time synchronization and scheduling, then there are two local loops, one that operates between the CPU and the NIC and one that operates between the NIC and the Edge TSN Bridge as illustrated in Figure 11.

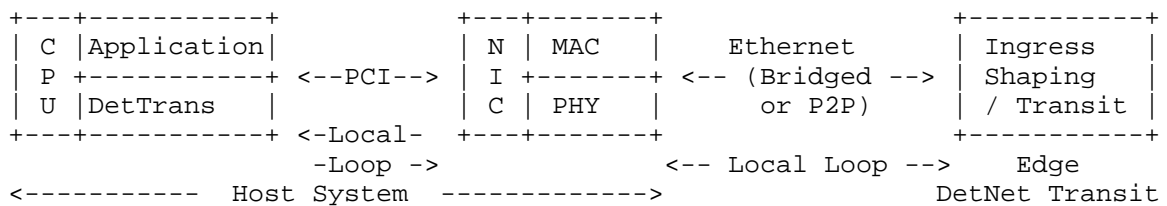


Figure 11: DetNet Capable NIC

5.1.3. Network Pull vs. Rate Based Flow Control

The flow control at the DetNet-UNI can take any of two forms:

Network Pull In that Model, the DetNet Edge node drains the DetTrans queue by sending a DetNet-UNI "More" command some estimated amount of time ahead of the scheduled time of transmission for each packet; in case of load sharing, multiple DetNet Edge nodes may drain a queue at their own rates; in case of a high jitter on the UNI Local Loop (e.g. there is a non-deterministic Bridge in between, or the NIC is not DetNet-Aware and the flows suffer from the more erratic response time of the CPU), the DetNet Edge node may need to pull a window of packets to maintain its own transmission queues fed at all times

Rate Based In that model, the NIC is aware of the rate of the deterministic transmission and is drained by its internal timers. Since the NIC is not synchronized with the Deterministic Network, the Bridge uses a DetNet-UNI "Time-Correction" command asynchronously to move forward or backward the next timeout of the NIC for that flow, in order to keep the Rate-Based transmission by the NIC in rough alignment with the scheduled transmission over the DetNet network.

if the NIC is DetNet-Aware, it is expected that it maintains the DetTrans queues in order to provide a deterministic response to the DetNet-UNI, and in that case another control loop between the NIC and the CPU is needed to ensure that the queue in the NIC is always fed in time by the DetTrans Layer; this second loop may be of a different nature than the DetNet-UNI one and may for instance be operated within an interrupt to limit the asynchronism related to message queueing.

5.2. DetNet-UNI Protocol Exchanges

5.2.1. the "More" Message

The "More" message enables a DetNet Transport Edge to pull one packet from the DetTrans Layer in Network-Pull mode. The message is associated with a future transmission opportunity for a packet. The "More" messages are indexed by a wrapping More Sequence Counter (MSC). The Transport Edge also maintains wrapping counters of Successful Packet Transmissions (SPT) and Missed Transmit Opportunities (MTO). The current value of these counters is placed in the "More" message.

Upon reception of a "More" message, the DetTrans Layer, or the NIC on behalf of the DetTrans Layer, sends the next available packet for

that session. The packet is encapsulated and the encapsulation indicates the MSC. This enables the DetNet Transport Edge to correlate the packet with the transmission opportunity and drop packets that are overly delayed.

5.2.2. the "Time-Correction" Message

The "Time-Correction" message enables a DetNet Transport Edge to adjust the timer associated to the DetNet-UNI session in Rate-Based mode. In that mode, the DetTrans Layer sends a packet and restarts a timer at a period that corresponds to the transmission opportunity of the DetNet Transport Edge. If the clock in the CPU drifts, the DetNet Transport Edge will start receiving packets increasingly ahead of expected time or behind expected time. It is expected that the DetNet Transport Edge is protected against a minimum drift by a guard time, but if the drift becomes too important, then the DetNet Transport Edge issues a "Time-Correction" message indicating a number of time units (e.g. microseconds) by which the DetTrans Layer should advance or delay its next time out.

5.2.3. Loss of a Control Message

The loss of a packet between the DetTrans Layer and the DetNet Transport Edge will correspond to a missed Transmission Opportunity but this does not mean that packets are piling up at the DetTrans Layer. OTOH, if a "More" message is lost, then one packet will not be dequeued and the DetTrans queue might grow, increasingly augmenting the latency. It is thus important to differentiate these situations, and in the latter case, discard an extraneous packet to restore the normal level in the DetTrans queue for that session.

In order to do so, the DetTrans Layer maintains the record of the Number of Packets Sent (NPS), that it compares with the variation of the MTO and SPT counters in the "More" message. Upon a "More" message, the DetTrans Layer computes the variation of NPS ($dNPS = NPS2 - NPS1$) and the variation of SPT ($dSPT = SPT2 - SPT1$) since the previous "More" Message. $dNPS$ is typically 1 if the transport always has data to send. Packets in flight when the "More" message is sent are considered lost since they will be received after their scheduled transmission opportunity, so the Number of Packets Losses (NPL) is ($NPL = dNPS - dSPT$). The DetTrans Layer also computes the variation of MTO since the previous "More" Message ($dMTO = MTO2 - MTO1$). Since a packet loss implies a missed transmission opportunity, there cannot be more packets losses than missed opportunities, so we have $dMTO \geq NPL$. $dMTO - NPL$ represents the number of missed opportunities that are not due to a packet lost or late arrival, thus this is the sub-count of MTOs due to the loss of a "More" message.

For each loss of a "More" message, a packet in the DetTrans queue should be discarded. In order to simplify that operation and outboard it to the NIC, the Transports marks some packets as "Discard Eligible" (DE). A packet can be marked DE if there are enough alternate transmissions of non-DE packets to recover this. For instance, in case of Packet Replication and Elimination only one copy can be marked DE, and the marking should alternate between the sessions to cover a loss on either one rapidly.

6. Security Considerations

The generic threats against Deterministic Networking are discussed in the "Deterministic Networking Security" [I-D.ietf-detnet-security] document.

Security in the context of Deterministic Networking has an added dimension; the time of delivery of a packet can be just as important as the contents of the packet, itself. A man-in-the-middle attack, for example, can impose, and then systematically adjust, additional delays into a link, and thus disrupt or subvert a real-time application without having to crack any encryption methods employed. See [RFC7384] for an exploration of this issue in a related context.

Packet Replication and Elimination if done right can prevent a man-in-the-middle attack on one leg to actually impact the flow beyond the loss of an individual packet for lack of redundancy. This specification expects that PRE is performed at the transport level and provides specific means to protect one leg against misuse of the other.

7. IANA Considerations

This document does not require an action from IANA.

8. Acknowledgments

The authors wish to thank Patrick Wetterwald, Leon Turkevitch, Balazs Varga and Janos Farkas for their various contributions to this work. Special thanks to Norm Finn for being a (if not the) major thought leader to the whole deterministic effort, and for some text that is inlined here from other IETF documents, for the convenience of the reader.

9. Informative References

- [AVnu] <http://www.avnu.org/>, "The AVnu Alliance tests and certifies devices for interoperability, providing a simple and reliable networking solution for AV network implementation based on the IEEE Audio Video Bridging (AVB) and Time-Sensitive Networking (TSN) standards."
- [EIP] <http://www.odva.org/>, "EtherNet/IP provides users with the network tools to deploy standard Ethernet technology (IEEE 802.3 combined with the TCP/IP Suite) for industrial automation applications while enabling Internet and enterprise connectivity data anytime, anywhere.", <http://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00138R3_CIP_Adv_Tech_Series_EtherNetIP.pdf>.
- [HART] www.hartcomm.org, "Highway Addressable Remote Transducer, a group of specifications for industrial process and control devices administered by the HART Foundation".
- [I-D.ietf-detnet-architecture]
Finn, N., Thubert, P., Varga, B., and J. Farkas, "Deterministic Networking Architecture", draft-ietf-detnet-architecture-03 (work in progress), August 2017.
- [I-D.ietf-detnet-problem-statement]
Finn, N. and P. Thubert, "Deterministic Networking Problem Statement", draft-ietf-detnet-problem-statement-02 (work in progress), September 2017.
- [I-D.ietf-detnet-security]
Mizrahi, T., Grossman, E., Hacker, A., Das, S., Dowdell, J., Austad, H., Stanton, K., and N. Finn, "Deterministic Networking (DetNet) Security Considerations", draft-ietf-detnet-security-00 (work in progress), October 2017.
- [I-D.ietf-detnet-use-cases]
Grossman, E., Gunther, C., Thubert, P., Wetterwald, P., Raymond, J., Korhonen, J., Kaneko, Y., Das, S., Zha, Y., Varga, B., Farkas, J., Goetz, F., Schmitt, J., Vilajosana, X., Mahmoodi, T., Spirou, S., Vizarrata, P., Huang, D., Geng, X., Dujovne, D., and M. Seewald, "Deterministic Networking Use Cases", draft-ietf-detnet-use-cases-13 (work in progress), September 2017.
- [I-D.varga-detnet-service-model]
Varga, B. and J. Farkas, "DetNet Service Model", draft-varga-detnet-service-model-02 (work in progress), May 2017.

- [IEC62439] IEC, "Industrial communication networks - High availability automation networks - Part 3: Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy (HSR) - IEC62439-3", 2012, <<https://webstore.iec.ch/publication/7018>>.
- [IEEE802.1TSNTG] IEEE Standards Association, "IEEE 802.1 Time-Sensitive Networks Task Group", 2013, <<http://www.ieee802.org/1/pages/avBridges.html>>.
- [ISA100.11a] ISA/IEC, "ISA100.11a, Wireless Systems for Automation, also IEC 62734", 2011, < <http://www.isa100wci.org/en-US/Documents/PDF/3405-ISA100-WirelessSystems-Future-broch-WEB-ETSI.aspx>>.
- [ODVA] <http://www.odva.org/>, "The organization that supports network technologies built on the Common Industrial Protocol (CIP) including EtherNet/IP.".
- [Profinet] <http://us.profinet.com/technology/profinet/>, "PROFINET is a standard for industrial networking in automation.", <<http://us.profinet.com/technology/profinet/>>.
- [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>>.
- [RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, DOI 10.17487/RFC7384, October 2014, <<https://www.rfc-editor.org/info/rfc7384>>.
- [WirelessHART] www.hartcomm.org, "Industrial Communication Networks - Wireless Communication Network and Communication Profiles - WirelessHART - IEC 62591", 2010.

Author's Address

Pascal Thubert (editor)
Cisco Systems, Inc
Building D (Regus) 45 Allee des Ormes
MOUGINS - Sophia Antipolis
FRANCE

Phone: +33 4 97 23 26 34
Email: pthubert@cisco.com

Network Working Group	M. Tüxen
Internet-Draft	Münster Univ. of Appl. Sciences
Updates: 6951 (if approved)	R. R. Stewart
Intended status: Standards Track	Netflix, Inc.
Expires: 31 August 2022	27 February 2022

Additional Considerations for UDP Encapsulation of Stream Control
Transmission Protocol (SCTP) Packets
draft-tuexen-tsvwg-sctp-udp-encaps-cons-05

Abstract

RFC 6951 specifies the UDP encapsulation of SCTP packets. The described handling of received packets requires the check of the verification tag. However, RFC 6951 misses a specification of the handling of received packets for which this check is not possible.

This document updates RFC 6951 by specifying the handling of received packets for which the verification tag can not be checked.

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 31 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.

Table of Contents

1. Introduction	2
2. Conventions	2
3. Handling of Out of the Blue Packets	3
4. Handling of SCTP Packets Containing an INIT Chunk Matching an Existing Associations	3
5. Middlebox Considerations	5
6. IANA Considerations	5
7. Security Considerations	6
8. Acknowledgments	6
9. Normative References	6
Authors' Addresses	7

1. Introduction

[RFC6951] specifies the UDP encapsulation of SCTP packets. To be able to adopt automatically to changes of the remote UDP encapsulation port number, it is updated when processing received packets. This includes automatic enabling and disabling of UDP encapsulation.

Section 5.4 of [RFC6951] describes the processing of received packets and requires the check of the verification tag before updating the remote UDP encapsulation port and the possible enabling or disabling of UDP encapsulation.

[RFC6951] basically misses a description of the handling of received packets where checking the verification tag is not possible. This includes packets for which no association can be found and packets containing an INIT chunk, since the verification tag of these packets is 0.

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. Handling of Out of the Blue Packets

If the processing of an out of the blue packet requires the sending of a packet in response according to the rules specified in Section 8.4 of [RFC4960], the following rules apply:

1. If the received packet was encapsulated in UDP, the response packets MUST also be encapsulated in UDP. The UDP source port and UDP destination port used for sending the response packet are the UDP destination port and UDP source port of the received packet.
2. If the received packet was not encapsulated in UDP, the response packet MUST NOT be encapsulated in UDP.

Please note that in these cases a check of the verification tag is not possible.

4. Handling of SCTP Packets Containing an INIT Chunk Matching an Existing Associations

SCTP packets containing an INIT chunk have the verification tag 0 in the common header. Therefore the verification tag can't be checked.

The following rules apply when processing the received packet:

1. The remote UDP encapsulation port for the source address of the received SCTP packet MUST NOT be updated if the encapsulation of outgoing packets is enabled and the received SCTP packet is encapsulated.
2. The UDP encapsulation for outgoing packets towards the source address of the received SCTP packet MUST NOT be enabled, if it is disabled and the received SCTP packet is encapsulated.
3. The UDP encapsulation for outgoing packets towards the source address of the received SCTP packet MUST NOT be disabled, if it is enabled and the received SCTP packet is not encapsulated.

4. If the UDP encapsulation for outgoing packets towards the source address of the received SCTP packet is disabled and the received SCTP packet is encapsulated, an SCTP packet containing an ABORT chunk MUST be sent. The ABORT chunk MAY include the error cause defined below indicating an "Restart of an Association with New Encapsulation Port". This packet containing the ABORT chunk MUST be encapsulated in UDP. The UDP source port and UDP destination port used for sending the packet containing the ABORT chunk are the UDP destination port and UDP source port of the received packet containing the INIT chunk.
5. If the UDP encapsulation for outgoing packets towards the source address of the received SCTP packet is disabled and the received SCTP packet is not encapsulated, the processing defined in [RFC4960] MUST be performed. If a packet is sent in response, it MUST NOT be encapsulated.
6. If the UDP encapsulation for outgoing packets towards the source address of the received SCTP packet is enabled and the received SCTP packet is not encapsulated, an SCTP packet containing an ABORT chunk MUST be sent. The ABORT chunk MAY include the error cause defined below indicating an "Restart of an Association with New Encapsulation Port". This packet containing the ABORT chunk MUST NOT be encapsulated in UDP.
7. If the UDP encapsulation for outgoing packets towards the source address of the received SCTP packet is enabled and the received SCTP packet is encapsulated, but the UDP source port of the received SCTP packet is not equal to the remote UDP encapsulation port for the source address of the received SCTP packet, an SCTP packet containing an ABORT chunk MUST be sent. The ABORT chunk MAY include the error cause defined below indicating an "Restart of an Association with New Encapsulation Port". This packet containing the ABORT chunk MUST be encapsulated in UDP. The UDP source port and UDP destination port used for sending the packet containing the ABORT chunk are the UDP destination port and UDP source port of the received packet containing the INIT chunk.
8. If the UDP encapsulation for outgoing packets towards the source address of the received SCTP packet is enabled and the received SCTP packet is encapsulated and the UDP source port of the received SCTP packet is equal to the remote UDP encapsulation port for the source address of the received SCTP packet, the processing defined in [RFC4960] MUST be performed. If a packet is sent in response, it MUST be encapsulated. The UDP source port and UDP destination port used for sending the packet containing the ABORT chunk are the UDP destination port and UDP source port of the received packet containing the INIT chunk.

The error cause indicating an "Restart of an Association with New Encapsulation Port" is defined by the following figure.

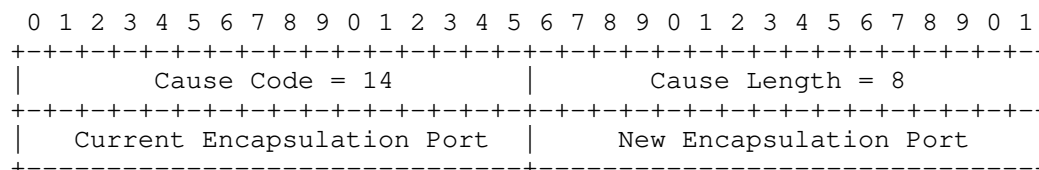


Figure 1: Restart of an Association with New Encapsulation Port error cause

Cause Code: 2 bytes (unsigned integer)

This field holds the IANA defined cause code for the "Restart of an Association with New Encapsulation Port" error cause. IANA is requested to assign the value 14 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 8.

Current Encapsulation Port: 2 bytes (unsigned integer)

This field holds the remote encapsulation port currently being used for the destination address the received packet containing the INIT chunk was sent from. If the UDP encapsulation for destination address is currently disabled, 0 is used.

New Encapsulation Port: 2 bytes (unsigned integer)

If the received SCTP packet containing the INIT chunk is encapsulated in UDP, this field holds the UDP source port number of the UDP packet. If the received SCTP packet is not encapsulated in UDP, this field is 0.

All transported integer numbers are in "network byte order" a.k.a., Big Endian.

5. Middlebox Considerations

Middleboxes often use different timeouts for UDP based flows than for other flows. Therefore the HEARTBEAT.Interval parameter SHOULD be lowered to 15 seconds when UDP encapsulation is used.

6. 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 cause code are tentative and to be confirmed by IANA.]

This document (RFCXXXX) is the reference for the registration described in this section.

A new error cause code has to be assigned by IANA. This requires an additional line in the "Error Cause Codes" registry for SCTP:

Value	Cause Code	Reference
14	Restart of an Association with New Encapsulation Port	[RFCXXXX]

Table 1: New entry in Error Cause Codes registry

7. Security Considerations

This document does not change the considerations given in [RFC6951].

However, not following the procedures given in this document might allow an attacker to take over SCTP associations. The attacker needs only to share the IP address of an existing SCTP association.

It should also be noted that if firewalls will be applied at the SCTP association level they have to take the UDP encapsulation into account.

8. Acknowledgments

The authors wish to thank Georgios Papastergiou for the initial problem report.

The authors wish to thank Irene Rüngeler and Felix Weinrank for their invaluable comments.

This work has received funding from 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).

9. 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>>.
- [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>>.

Authors' Addresses

Michael Tüxen
Münster University of Applied Sciences
Stegerwaldstrasse 39
48565 Steinfurt
Germany
Email: tuexen@fh-muenster.de

Randall R. Stewart
Netflix, Inc.
2455 Heritage Green Ave
Davenport, FL 33837
United States
Email: randall@lakerest.net