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 functionally 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

Table of Contents

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 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:

- 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.

- Path validation: The PLPMTUD method MUST be robust to path changes that could have occurred since the path characteristics were last confirmed.

- 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.

- 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 appication 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 appication 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 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.

- 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.

- 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. Verication 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.

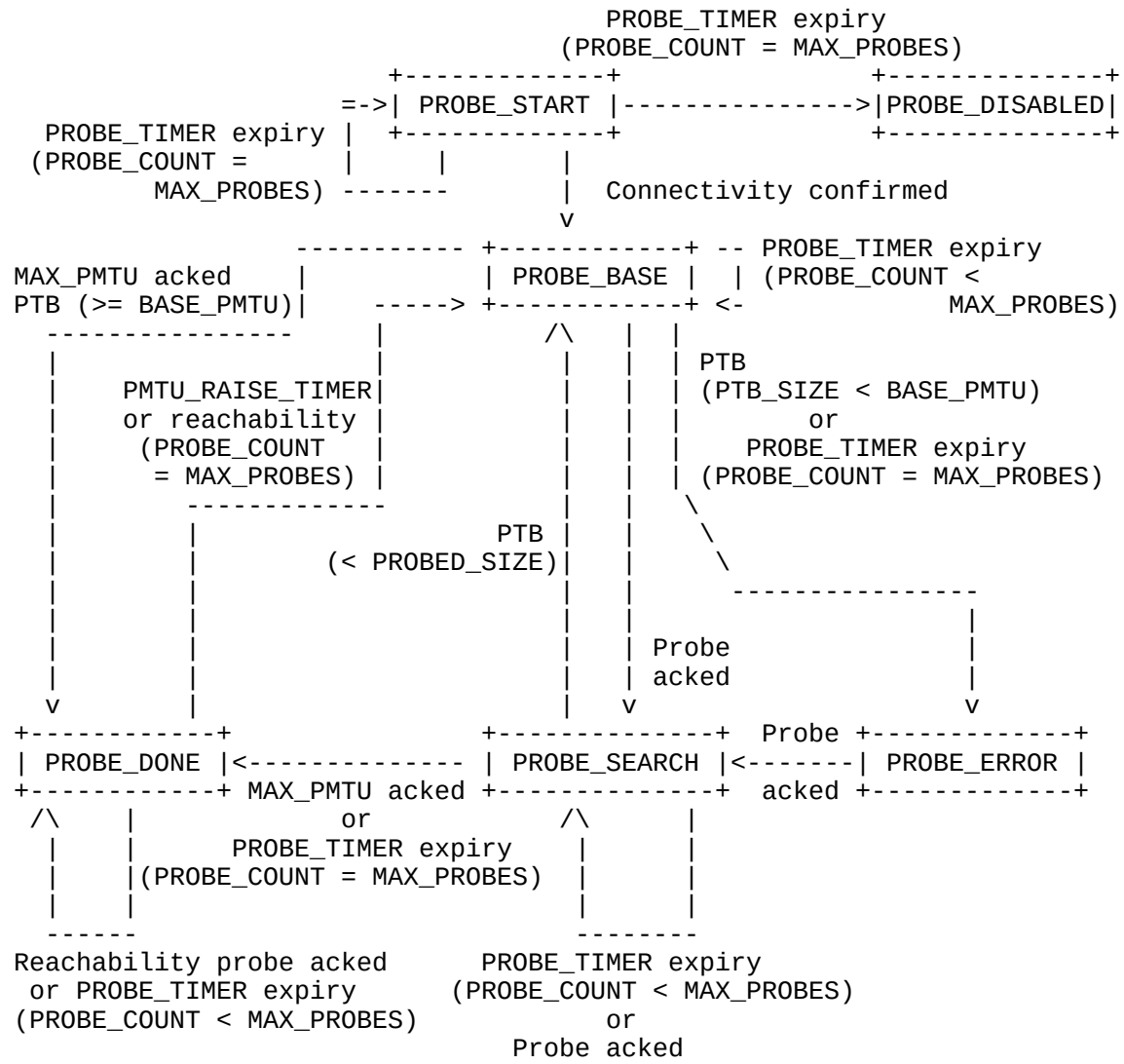
PROBED_SIZE: The PROBED_SIZE is the size of the current probe packet. This is a tentative value for the effective PMTU, which is awaiting confirmation by an acknowledgment.

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

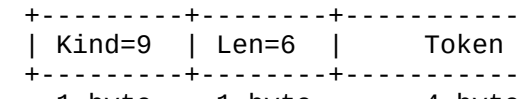
This subsection proposes two new UDP-Options that add support for requesting a datagram response be sent and to mark this datagram as a response to a request.

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.

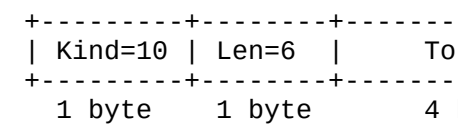
The Echo Request carries a four byte token set by the sender. This token can be set to a value that is likely to be known only to the sender (and becomes known to nodes along the end-to-end path). The sender can then check the value returned in the response to provide additional protection from off-path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



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 MUST 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 mesage 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>.

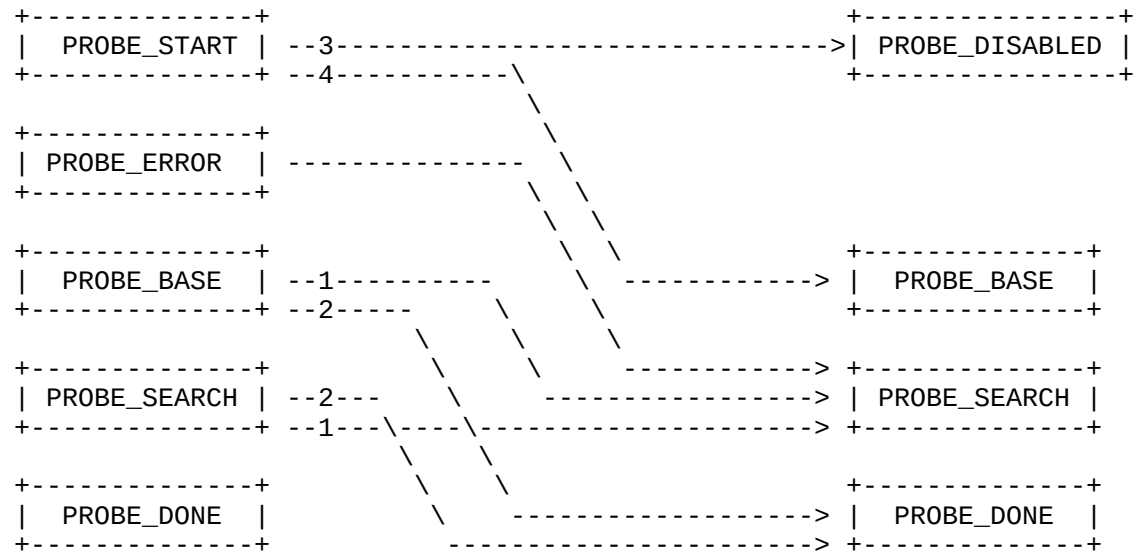
[RFC4890] Davies, E. and J. Mohacsi, "Recommendations for Filtering ICMPv6 Messages in Firewalls", RFC 4890, DOI 10.17487/RFC4890, May 2007, <http://www.rfc-editor.org/info/rfc4890>.

Appendix A. Event-driven state changes

This appendix contains an informative description of key events:

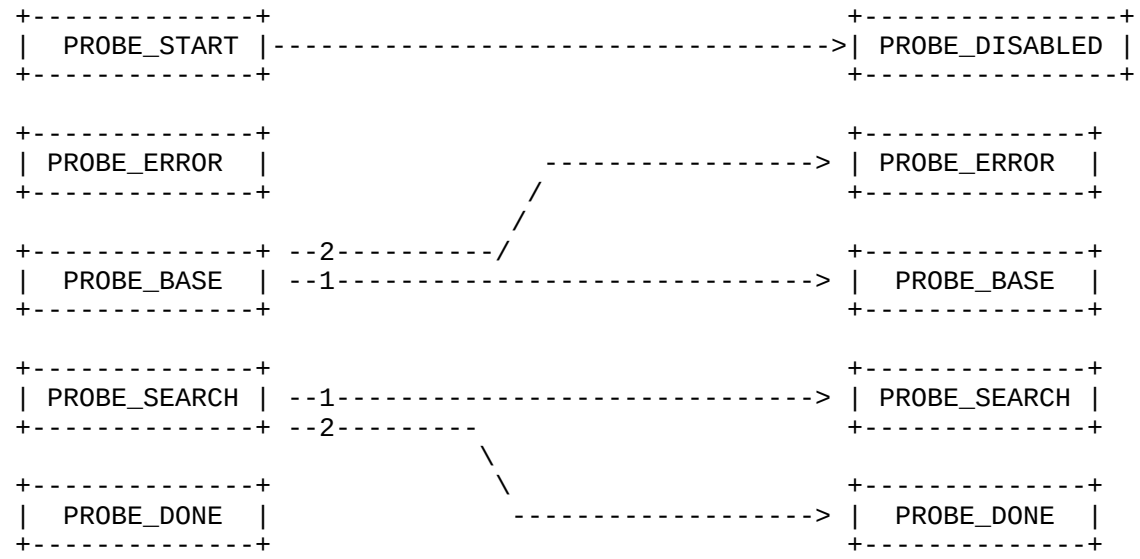
Path Setup: When a new path is initiated, the state is set to PROBE_START. As soon as the path is confirmed, the state changes to PROBE_BASE and the probing mechanism for this path is started. A probe packet with the size of the BASE_PMTU is sent.

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. Conition 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 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:

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

Individual draft -01:

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

Individual draft -02:

- Contains updated representation of the algorithm, and textual corrections.
- The text describing when to set the effective PMTU has not yet been verified by the authors
- 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
- No text currently describes how to handle inconsistent results from arbitrary re-routing along different parallel paths

- 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
Stein fart, 48565
DE

Email: tuexen@fh-muenster.de

Irene Ruengeler
Muenster University of Applied Sciences
Stegerwaldstrasse 39
Stein fart, 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.

Table of Contents

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:

- Understanding Feature Interactions: An appropriate vantage point, coupled with timing information about traffic flows, provides a valuable tool for benchmarking equipment, functions, and/or configurations, and to understand complex feature interactions. An inability to observe transport protocol information can limit the ability to diagnose and explore interactions between features at different protocol layers, a side-effect of not allowing a choice of vantage point from which this information is observed.

- Supporting Common Specifications: Transmission Control Protocol (TCP) is currently the predominant transport protocol used over Internet paths. Its many variants have broadly consistent approaches to avoiding congestion collapse, and to ensuring the stability of the Internet. Increased use of transport layer encryption can overcome ossification, allowing deployment of new transports and different types of congestion control. This flexibility can be beneficial, but it can come at the cost of fragmenting the ecosystem. There is little doubt that developers will try to produce high quality transports for their intended target uses, but it is not clear there are sufficient incentives

to ensure good practice that benefits the wide diversity of requirements for the Internet community as a whole. Increased diversity, and the ability to innovate without public scrutiny, risks point solutions that optimise for specific needs, but accidentally disrupt operations of/in different parts of the network. The social contract that maintains the stability of the Internet relies on accepting common specifications, and on the ability to verify that others also conform.

- 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.

- Restricting research and development: Hiding transport information can impede independent research into new mechanisms, measurement of behaviour, and development initiatives. Experience shows that transport protocols are complicated to design and complex to deploy, and that individual mechanisms need to be evaluated while considering other mechanisms, across a broad range of network topologies and with attention to the impact on traffic sharing the capacity. If this results in reduced availability of open data, it could eliminate the independent self-checks to the standardisation process that have previously been in place from research and academic contributors (e.g., the role of the IRTF ICCRG, and research publications in reviewing new transport mechanisms and assessing the impact of their experimental deployment)

In summary, there are trade offs. On the one hand, protocol designers have often ignored the implications of whether the information in transport header fields can or will be used by in-network devices, and the implications this places on protocol evolution. This motivates a design that provides confidentiality of the header information. On the other hand, it can be expected that a lack of visibility of transport header information can impact the ways that protocols are deployed, standardised, and their operational support. The choice of whether future transport protocols encrypt their protocol headers therefore needs to be taken based not solely on security and privacy considerations, but also taking into account the impact on operations, standards, and research. Any new Internet transport need to provide appropriate transport mechanisms and operational support to assure the resulting traffic can not result in

persistent congestion collapse [RFC2914]. This document suggests that the balance between information exposed and concealed should be carefully considered when specifying new protocols.

3. Current uses of Transport Headers within the Network

Despite transport headers having end-to-end meaning, some of these transport headers have come to be used in various ways within the Internet. In response to pervasive monitoring [RFC7624] revelations and the IETF consensus that "Pervasive Monitoring is an Attack" [RFC7258], efforts are underway to increase encryption of Internet traffic,. Applying confidentiality to transport header fields would affect how protocol information is used [RFC8404]. To understand these implications, it is first necessary to understand how transport layer headers are currently observed and/or modified by middleboxes within the network.

Transport protocols can be designed to encrypt or authenticate transport header fields. Authentication at the transport layer can be used to detect any changes to an immutable header field that were made by a network device along a path. The intentional modification of transport headers by middleboxes (such as Network Address Translation, NAT, or Firewalls) is not considered. Common issues concerning IP address sharing are described in [RFC6269].

3.1. Observing Transport Information in the Network

If in-network observation of transport protocol headers is needed, this requires knowledge of the format of the transport header:

- Flows need to be identified at the level required to perform the observation;
- The protocol and version of the header need to be visible. As protocols evolve over time and there may be a need to introduce new transport headers. This may require interpretation of protocol version information or connection setup information;
- The location and syntax of any observed transport headers needs to be known. IETF transport protocols can specify this information.

The following subsections describe various ways that observable transport information has been utilised.

3.1.1. Flow Identification

Transport protocol header information (together with information in the network header), has been used to identify a flow and the connection state of the flow, together with the protocol options being used. In some usages, a low-numbered (well-known) transport port number has been used to identify a protocol (although port information alone is not sufficient to guarantee identification of a protocol, since applications can use arbitrary ports, multiple sessions can be multiplexed on a single port, and ports can be re-used by subsequent sessions).

Transport protocols, such as TCP and Stream Control Transport Protocol (SCTP) specify a standard base header that includes sequence number information and other data, with the possibility to negotiate additional headers at connection setup, identified by an option number in the transport header. UDP-based protocols can use, but sometimes do not use, well-known port numbers. Some flows can instead be identified by signalling protocols or through the use of magic numbers placed in the first byte(s) of the datagram payload.

Flow identification is a common function. For example, performed by measurement activities, QoS classification, firewalls, Denial of Service, DOS, prevention. It becomes more complex and less easily achieved when multiplexing is used at or above the transport layer.

3.1.2. Metrics derived from Transport Layer Headers

Some actors manage their portion of the Internet by characterizing the performance of link/network segments. Passive monitoring uses observed traffic to makes inferences from transport headers to derive these measurements. A variety of open source and commercial tools have been deployed that utilise this information. The following metrics can be derived from transport header information:

Traffic Rate and Volume: Header information e.g., (sequence number, length) allows derivation of volume measures per-application, to characterise the traffic that uses a network segment or the pattern of network usage. This may be measured per endpoint or for an aggregate of endpoints (e.g., by an operator to assess subscriber usage). It can also be used to trigger measurement-based traffic shaping and to implement QoS support within the network and lower layers. Volume measures can be valuable for capacity planning (providing detail of trends rather than the volume per subscriber).

Loss Rate and Loss Pattern: Flow loss rate may be derived (e.g., from sequence number) and has been used as a metric for

performance assessment and to characterise transport behaviour. Understanding the root cause of loss can help an operator determine whether this requires corrective action. Network operators have used the variation in patterns of loss as a key performance metric, utilising this to detect changes in the offered service.

There are various causes of loss, including: corruption of link frames (e.g., interference on a radio link), buffer overflow (e.g., due to congestion), policing (traffic management), buffer management (e.g., Active Queue Management, AQM [RFC7567]), inadequate provision of traffic preemption. Understanding flow loss rate requires either maintaining per flow packet counters or by observing sequence numbers in transport headers. Loss can be monitored at the interface level by devices in the network. It is often important to understand the conditions under which packet loss occurs. This usually requires relating loss to the traffic flowing on the network node/segment at the time of loss.

Observation of transport feedback information (observing loss reports, e.g., RTP Control Protocol (RTCP) [RFC3550], TCP SACK) can increase understanding of the impact of loss and help identify cases where loss may have been wrongly identified, or the transport did not require the lost packet. It is sometimes more important to understand the pattern of loss, than the loss rate, because losses can often occur as bursts, rather than randomly-timed events.

Throughput and Goodput: The throughput achieved by a flow can be determined even when a flow is encrypted, providing the individual flow can be identified. Goodput [RFC7928] is a measure of useful data exchanged (the ratio of useful/total volume of traffic sent by a flow). This requires ability to differentiate loss and retransmission of packets (e.g., by observing packet sequence numbers in the TCP or the Real Time Protocol, RTP, headers [RFC3550]).

Latency: Latency is a key performance metric that impacts application response time and user-perceived response time. It often indirectly impacts throughput and flow completion time. Latency determines the reaction time of the transport protocol itself, impacting flow setup, congestion control, loss recovery, and other transport mechanisms. The observed latency can have many components [Latency]. Of these, unnecessary/unwanted queuing in network buffers has often been observed as a significant factor. Once the cause of unwanted latency has been identified, this can often be eliminated.

To measure latency across a part of a path, an observation point can measure the experienced round trip time (RTT) using packet sequence numbers, and acknowledgements, or by observing header timestamp information. Such information allows an observation point in the network to determine not only the path RTT, but also to measure the upstream and downstream contribution to the RTT. This has been used to locate a source of latency, e.g., by observing cases where the ratio of median to minimum RTT is large for a part of a path.

The service offered by operators can benefit from latency information to understand the impact of deployment and tune deployed services. Latency metrics are key to evaluating and deploying AQM [RFC7567], DiffServ [RFC2474], and Explicit Congestion Notification (ECN) [RFC3168] [RFC8087]. Measurements could identify excessively large buffers, indicating where to deploy or configure AQM. An AQM method is often deployed in combination with other techniques, such as scheduling [RFC7567] [RFC8290] and although parameter-less methods are desired [RFC7567], current methods [RFC8290] [RFC8289] [RFC8033] often cannot scale across all possible deployment scenarios.

Variation in delay: Some network applications are sensitive to small changes in packet timing. To assess the performance of such applications, it can be necessary to measure the variation in delay observed along a portion of the path [RFC3393] [RFC5481]. The requirements resemble those for the measurement of latency.

Flow Reordering: Significant flow reordering can impact time-critical applications and can be interpreted as loss by reliable transports. Many transport protocol techniques are impacted by reordering (e.g., triggering TCP retransmission, or re-buffering of real-time applications). Packet reordering can occur for many reasons (from equipment design to misconfiguration of forwarding rules). Since this impacts transport performance, network tools are needed to detect and measure unwanted/excessive reordering.

There have been initiatives in the IETF transport area to reduce the impact of reordering within a transport flow, possibly leading to a reduction in the requirements for preserving ordering. These have promise to simplify network equipment design as well as the potential to improve robustness of the transport service. Measurements of reordering can help understand the present level of reordering within deployed infrastructure, and inform decisions about how to progress such mechanisms.

Operational tools to detect mis-ordered packet flows and quantify the degree or reordering. Key performance indicators are retransmission

rate, packet drop rate, sector utilisation level, a measure of reordering, peak rate, the ECN congestion experienced (CE) marking rate, etc.

Metrics have been defined that evaluate whether a network has maintained packet order on a packet-by-packet basis [RFC4737] and [RFC5236].

Techniques for measuring reordering typically observe packet sequence numbers. Some protocols provide in-built monitoring and reporting functions. Transport fields in the RTP header [RFC3550] [RFC4585] can be observed to derive traffic volume measurements and provide information on the progress and quality of a session using RTP. As with other measurement, metadata is often important to understand the context under which the data was collected, including the time, observation point, and way in which metrics were accumulated. The RTCP protocol directly reports some of this information in a form that can be directly visible in the network. A user of summary measurement data needs to trust the source of this data and the method used to generate the summary information.

3.1.3. Metrics derived from Network Layer Headers

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

Use of IPv6 Network-Layer Flow Label: Endpoints are encouraged 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,s with direct impact on the available capacity, and potentially loss/reordering of packets. The impact of the pattern of loss and congestion, differs for different traffic types, correlation with propagation and interference can all have significant impact on the cost and performance of a provided service. The need for this type of information is expected to increase as operators bring together heterogeneous types of network equipment and seek to deploy opportunistic methods to access radio spectrum.

3.4. Observing Headers to Implement Network Policy

Information from the transport protocol can be used by a multi-field classifier as a part of policy framework. Policies are commonly used for management of the QoS or Quality of Experience (QoE) in resource-constrained networks and by firewalls that use the information to implement access rules (see also section 2.2.2 of [RFC8404]). Traffic that cannot be classified, will typically receive a default treatment.

4. Encryption and Authentication of Transport Headers

End-to-end encryption can be applied at various protocol layers. It can be applied above the transport to encrypt the transport payload. Encryption methods can hide information from an eavesdropper in the network. Encryption can also help protect the privacy of a user, by

hiding data relating to user/device identity or location. Neither an integrity check nor encryption methods prevent traffic analysis, and usage needs to reflect that profiling of users, identification of location and fingerprinting of behaviour can take place even on encrypted traffic flows.

There are several motivations:

- One motive to use encryption is a response to perceptions that the network has become ossified by over-reliance on middleboxes that prevent new protocols and mechanisms from being deployed. This has lead to a perception that there is too much "manipulation" of protocol headers within the network, and that designing to deploy in such networks is preventing transport evolution. In the light of this, a method that authenticates transport headers may help improve the pace of transport development, by eliminating the need to always consider deployed middleboxes [I-D.trammell-plus-abstract-mech], or potentially to only explicitly enable middlebox use for particular paths with particular middleboxes that are deliberately deployed to realise a useful function for the network and/or users[RFC3135].

- Another motivation stems from increased concerns about privacy and surveillance. Some Internet users have valued the ability to protect identity, user location, and defend against traffic analysis, and have used methods such as IPsec Encapsulated Security Payload (ESP), Virtual Private Networks (VPNs) and other encrypted tunnel technologies. Revelations about the use of pervasive surveillance [RFC7624] have, to some extent, eroded trust in the service offered by network operators, and following the Snowden revelation in the USA in 2013 has led to an increased desire for people to employ encryption to avoid unwanted "eavesdropping" on their communications. Concerns have also been voiced about the addition of information to packets by third parties to provide analytics, customization, advertising, cross-site tracking of users, to bill the customer, or to selectively allow or block content. Whatever the reasons, there are now activities in the IETF to design new protocols that may include some form of transport header encryption (e.g., QUIC [I-D.ietf-quic-transport]).

Authentication methods (that provide integrity checks of protocols fields) have also been specified at the network layer, and this also protects transport header fields. The network layer itself carries protocol header fields that are increasingly used to help forwarding decisions reflect the need of transport protocols, such as the IPv6 Flow Label [RFC6437], the DSCP and ECN.

The use of transport layer authentication and encryption exposes a tussle between middlebox vendors, operators, applications developers and users.

- On the one hand, future Internet protocols that enable large-scale encryption assist in the restoration of the end-to-end nature of the Internet by returning complex processing to the endpoints, since middleboxes cannot modify what they cannot see.

- On the other hand, encryption of transport layer header information has implications for people who are responsible for operating networks and researchers and analysts seeking to understand the dynamics of protocols and traffic patterns.

Whatever the motives, a decision to use pervasive of transport header encryption will have implications on the way in which design and evaluation is performed, and which can in turn impact the direction of evolution of the TCP/IP stack. While the IETF can specify protocols, the success in actual deployment is often determined by many factors [RFC5218] that are not always clear at the time when protocols are being defined.

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 observes these fields. An integrity check can not prevent in-network modification, but can avoid a receiving accepting changes and avoid impact on the transport protocol operation.

An example transport authentication mechanism is TCP-Authentication (TCP-AO) [RFC5925]. This TCP option authenticates the IP pseudo header, TCP header, and TCP data. TCP-AO protects the transport layer, preventing attacks from disabling the TCP connection itself and provides replay protection. TCP-AO may interact with middleboxes, depending on their behaviour [RFC3234].

The IPsec Authentication Header (AH) [RFC4302] was designed to work at the network layer and authenticate the IP payload. This approach authenticates all transport headers, and verifies their integrity at the receiver, preventing in-network modification.

4.2. Encrypting the Transport Payload

The transport layer payload can be encrypted to protect the content of transport segments. This leaves transport protocol header information in the clear. The integrity of immutable transport header fields could be protected by combining this with an integrity check (Section 4.1).

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

4.3. Encrypting the Transport Header

The network layer payload could be encrypted (including the entire transport header and the payload). This method provides confidentiality of the entire transport packet. It therefore does not expose any transport information to devices in the network, which also prevents modification along a network path.

One example of encryption at the network layer is use of IPsec Encapsulating Security Payload (ESP) [RFC4303] in tunnel mode. This encrypts and authenticates all transport headers, preventing visibility of the transport headers by in-network devices. Some Virtual Private Network (VPN) methods also encrypt these headers.

4.4. Authenticating Transport Information and Selectively Encrypting the Transport Header

A transport protocol design can encrypt selected header fields, while also choosing to authenticate fields in the transport header. This allows specific transport header fields to be made observable by network devices. End-to end integrity checks can prevent an endpoint from undetected modification of the immutable transport headers.

Mutable fields in the transport header provide opportunities for middleboxes to modify the transport behaviour (e.g., the extended headers described in [I-D.trammell-plus-abstract-mech]). This considers only immutable fields in the transport headers, that is, fields that may be authenticated End-to-End across a path.

An example of a method that encrypts some, but not all, transport information is GRE-in-UDP [RFC8086] when used with GRE encryption.

4.5. Optional Encryption of Header Information

There are implications to the use of optional header encryption in the design of a transport protocol, where support of optional mechanisms can increase the complexity of the protocol and its implementation and in the management decisions that are required to use variable format fields. Instead, fields of a specific type ought to always be sent with the same level of confidentiality or integrity protection.

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

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

Some measurements may be made by adding additional protocol headers carrying operations, administration and management (OAM) information to packets at the ingress to a maintenance domain (e.g., an Ethernet protocol header with timestamps and sequence number information using a method such as 802.11ag or in-situ OAM [I-D.ietf-ippm-ioam-data]) and removing the additional header at the egress of the maintenance domain. This approach enables some types of measurements, but does not cover the entire range of measurements described in this document. In some cases, it can be difficult to position measurement tools at the required segments/nodes and there can be challenges in correlating the downsream/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 caries header fields that can be used to observe timestamps and packet sequence numbers. This information could be authenticated by receiving transport endpoints when the information is added at the sender and visible at the receiving endpoint, although methods to do this have not currently been proposed. This method needs to be explicitly enabled at the sender.

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 asses congestion and requests for retransmission) provide an incentive for the sending endpoint to provide correct information, increasing confidence that the observer understands the transport interaction with the network. This becomes important when considering changes to transport protocols, changes in network infrastructure, or the emergence of new traffic patterns.

6.2. Characterising "Unknown" Network Traffic

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

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

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

6.3. Accountability and Internet Transport Protocols

Information provided by tools observing transport headers can be used to classify traffic, and to limit the network capacity used by certain flows. Operators can potentially use this information to prioritise or de-prioritise certain flows or classes of flow, with potential implications for network neutrality, or to rate limit malicious or otherwise undesirable flows (e.g., for Distributed Denial of Service, DDOS, protection, or to ensure compliance with a traffic profile Section 3.2.4). Equally, operators could use analysis of transport headers and transport flow state to demonstrate that they are not providing differential treatment to certain flows. Obfuscating or hiding this information using encryption is expected to lead operators and maintainers of middleboxes (firewalls, etc.) to seek other methods to classify, and potentially other mechanisms to condition, network traffic.

A lack of data reduces the level of precision with which flows can be classified and conditioning mechanisms are applied (e.g., rate limiting, circuit breaker techniques [RFC8084], or blocking of uncharacterised traffic), and this needs to be considered when evaluating the impact of designs for transport encryption [RFC5218].

6.4. Impact on Research, Development and Deployment

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

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

Integrity checks can protect an endpoint from undetected modification of protocol fields by network devices, whereas encryption and obfuscation can further prevent these headers being utilised by network devices. Hiding headers can therefore provide the opportunity for greater freedom to update the protocols and can ease experimentation with new techniques and their final deployment in endpoints.

Hiding headers can limit the ability to measure and characterise traffic. Measurement data is increasingly being used to inform design decisions in networking research, during development of new mechanisms and protocols and in standardisation. Measurement has a critical role in the design of transport protocol mechanisms and their acceptance by the wider community (e.g., as a method to judge the safety for Internet deployment). Observation of pathologies are also important in understanding the interactions between cooperating protocols and network mechanism, the implications of sharing capacity with other traffic and the impact of different patterns of usage.

Evolution and the ability to understand (measure) the impact need to proceed hand-in-hand. Attention needs to be paid to the expected scale of deployment of new protocols and protocol mechanisms. Whatever the mechanism, experience has shown that it is often difficult to correctly implement combination of mechanisms [RFC8085]. These mechanisms therefore typically evolve as a protocol matures, or in response to changes in network conditions, changes in network traffic or changes to application usage.

New transport protocol formats are expected to facilitate an increased pace of transport evolution, and with it the possibility to experiment with and deploy a wide range of protocol mechanisms.

There has been recent interest in a wide range of new transport methods, e.g., Larger Initial Window, Proportional Rate Reduction (PRR), congestion control methods based on measuring bottleneck bandwidth and round-trip propagation time, the introduction of AQM techniques and new forms of ECN response (e.g., Data Centre TCP, DCTP, and methods proposed for L4S).The growth and diversity of applications and protocols using the Internet also continues to expand. For each new method or application it is desirable to build a body of data reflecting its behaviour under a wide range of deployment scenarios, traffic load, and interactions with other deployed/candidate methods.

Open standards motivate a desire for this evaluation to include independent observation and evaluation of performance data, which in turn suggests control over where and when measurement samples are collected. This requires consideration of the appropriate balance between encrypting all and no transport information.

7. Conclusions

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

Confidentiality and strong integrity checks have properties that are being incorporated into new protocols and which have important benefits. The pace of development of transports using the WebRTC data channel and the rapid deployment of QUIC prototype transports can both be attributed to using a combination of UDP transport and confidentiality of the UDP payload.

The traffic that can be observed by on-path network devices is a function of transport protocol design/options, network use, applications and user characteristics. In general, when only a small proportion of the traffic has a specific (different) characteristic. Such traffic seldom leads to an operational issue although the ability to measure and monitor it is less. The desire to understand the traffic and protocol interactions typically grows as the proportion of traffic increases in volume. The challenges increase when multiple instances of an evolving protocol contribute to the traffic that share network capacity.

An increased pace of evolution therefore needs to be accompanied by methods that can be successfully deployed and used across operational networks. This leads to a need for network operators (at various level (ISPs, enterprises, firewall maintainer, etc) to identify appropriate operational support functions and procedures.

Protocols that change their transport header format (wire format) or their behaviour (e.g., algorithms that are needed to classify and characterise the protocol), will require new tooling needs to be developed to catch-up with the changes. If the currently deployed tools and methods are no longer relevant and performance may not be correctly measured. This can increase the response-time after faults, and can impact the ability to manage the network resulting in traffic causing traffic to be treated inappropriately (e.g., rate limiting because of being incorrectly classified/monitored). There are benefits in exposing consistent information to the network that avoids traffic being mis-classified and then receiving a default treatment by the network.

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

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

- An alternative scenario adopts different design goals, with a different outcome. A protocol that encrypts all header information forces network operators to act independently from

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

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

8. Security Considerations

This document is about design and deployment considerations for transport protocols. Issues relating to security are discussed in the various sections of the document.

Authentication, confidentiality protection, and integrity protection are identified as Transport Features by [RFC8095]. As currently deployed in the Internet, these features are generally provided by a protocol or layer on top of the transport protocol [I-D.ietf-taps-transport-security].

Confidentiality and strong integrity checks have properties that can also be incorporated into the deisgn 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 yo 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 obfusticate 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@csperkins.org
URI: https://csperkins.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

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.

Table of Contents

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].

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

1.3. Priority Switching Scheduler in a nutshell

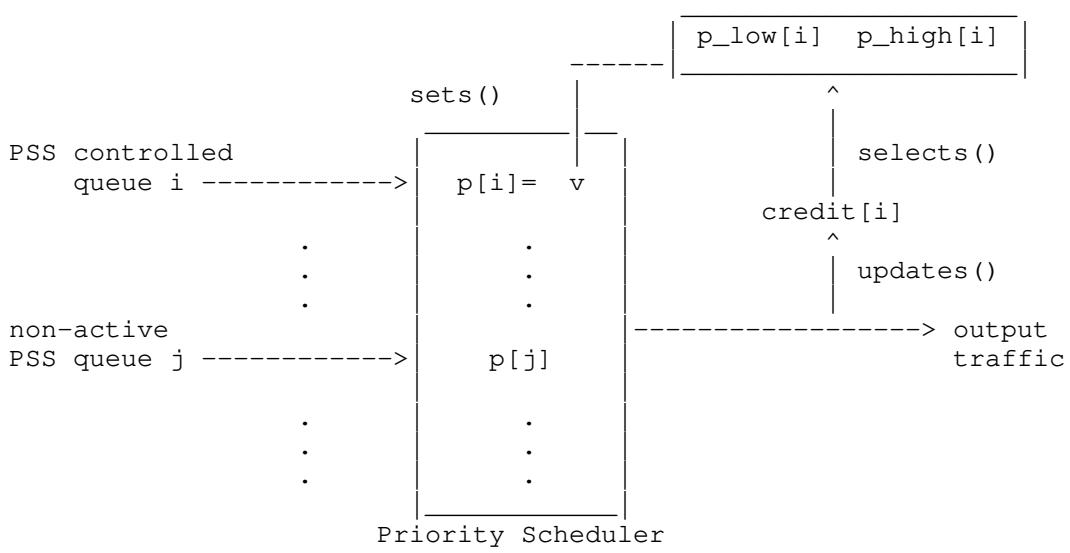


Figure 1: PSS in a nutshell

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 takes 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 transfered 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.

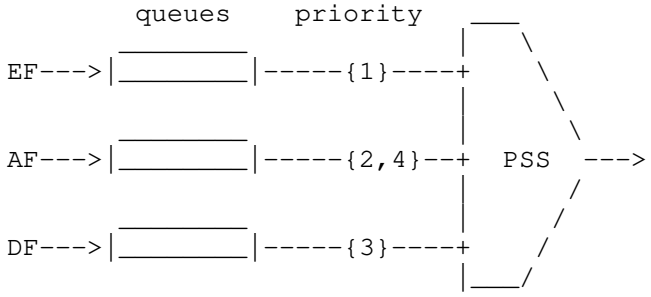


Figure 2: PSS with three traffic classes

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:

- a minimum level: 0;
- a maximum level: LM;
- a resume level: LR such as 0 <= LR < LR;
- a reserved capacity: BW;
- an idle slope: I_idle = C * BW, where C is the link output capacity;

- a sending slope: I_send = C - I_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 Lmax * BW, with Lmax 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:

- initially, the credit counter starts at 0;
- the change of priority p[i] of controlled queue i occurs in two cases:
  - if p[i] is currently set to p_high[i] and credit[i] reaches LM;
  - if p[i] is currently set to p_low[i] and credit[i] reaches LR;

- 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;

- when the credit reaches LM, it remains at this level until the end of the transmission of the current packet (if any);

- when the credit reaches LR and the transmission of the current packet is finished, in the abscence 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_high[i] is available, this packet is sent before the traffic of queue i. Secondly, when no traffic with a priority lower than p_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_high[i] (resp. p[i] = p_low[i]) does not necessarily mean that traffic of queue i is being sent (resp. not being sent).

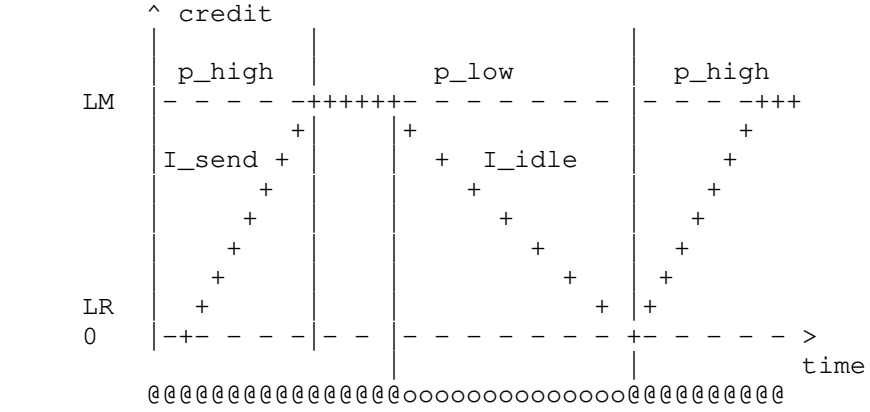


Figure 3: First example of queue credit evolution and priority switching.

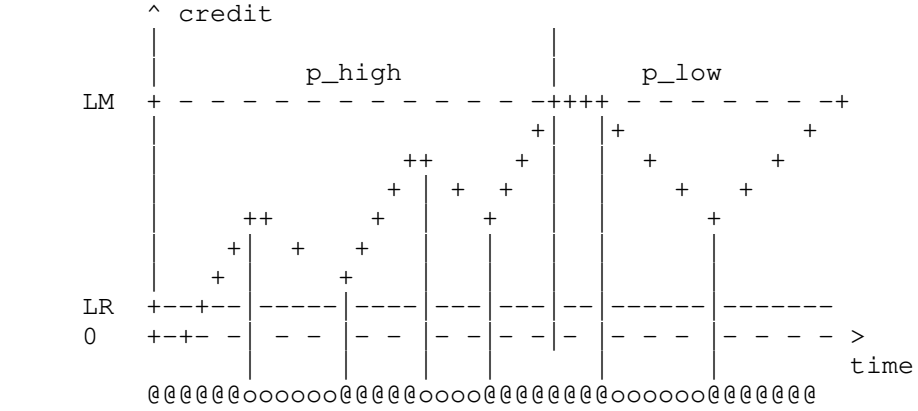


Figure 4: Second example of queue credit evolution and priority switching.

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 =
                min(LM, credit + size(head(queue[i])) * (1 - BW))
17            timerDQ = currentTime + size(head(queue[i]))/C
18            if credit >= LM and p = p_high then:
19               p = p_low
20            end if
21         end if
22         dequeue(head(queue[i]))
23         break
24      end if
25   end for
```

Figure 5: PSS algorithm

PSS algorithm implements the following functions:

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

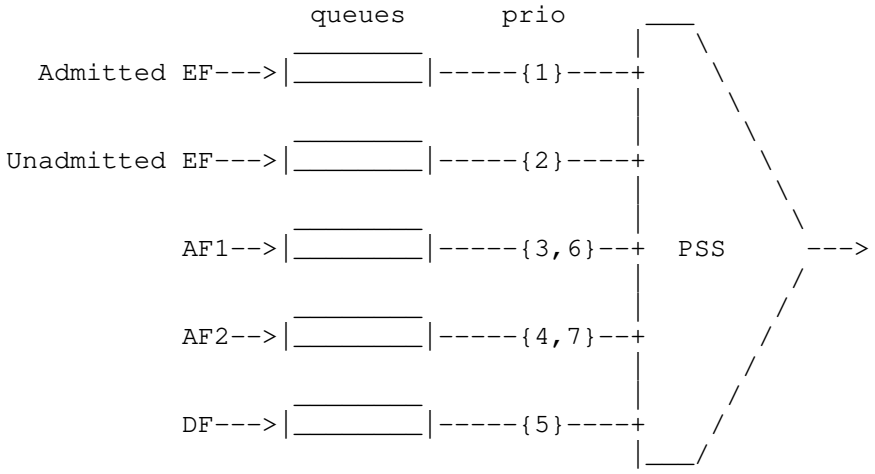


Figure 6: PSS with three traffic classes

```
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:

- 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 refered 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.

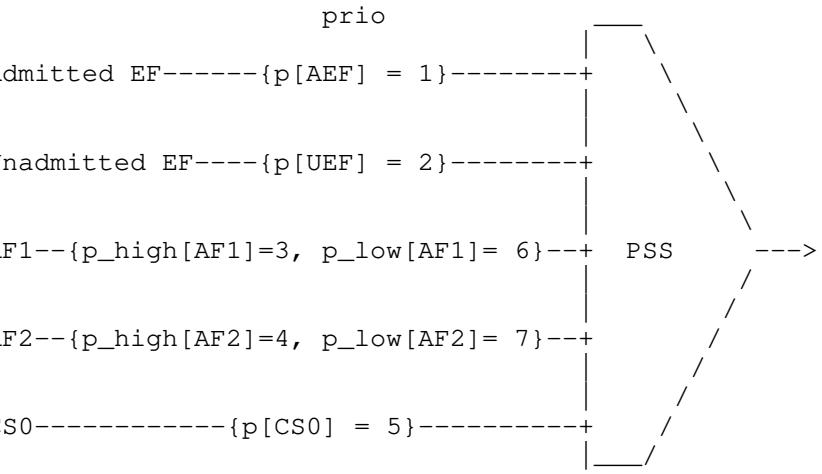


Figure 8: PSS applied to Data Plane Priority (we borrow the syntax from RCF5865)

3.2. New service offered

The new service we seek to obtain is:

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

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

4. Security Considerations

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

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

5. Acknowledgements

This document was the result of collaboration and discussion among a large number of people. In particular the authors wish to thank 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-supaero.fr

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

Email: ahlem.mifdaoui@isae-supaero.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.

Table of Contents

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-iccrg-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.

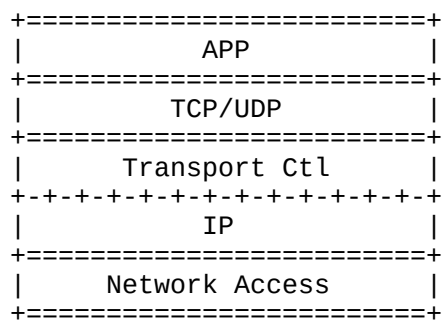


Figure 1: The new stack with a sub-layer in Layer 3

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 reprogramed 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,

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

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

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

   * Retrieve the QoS parameters to program the Hardware, it includes: FL, Time, Bandwidth, Latency, Burst

   * Update the field in the EH, it includes: Hop_number, Total_latency, and possibly 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-iccrg-arvr-transport-problem]. Based on this characteristics, application can request a relatively small CIR for the time of P-frame and P-frame, and request a big CIR for the time of I-frame.

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

- Setup the QoS channel for a TCP when the TCP session is establishing.

- 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:

- 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.

- 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:

- IP path properties: Total hop number that is HbH-EH-aware node; The IP address of each HbH-EH-aware node.

- 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.

- 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:

- IP path dynamic properties: Total end to end latency

- Dynamic properties at ach HbH-EH-aware node: Queue size; Remained bandwidth; Dropped packet number by different reasons.

- 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 harsh 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 securely 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-banding signaling is processed in network processor without punting to controller CPU for help

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

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

4. The 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 muts 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 be discovered or 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:

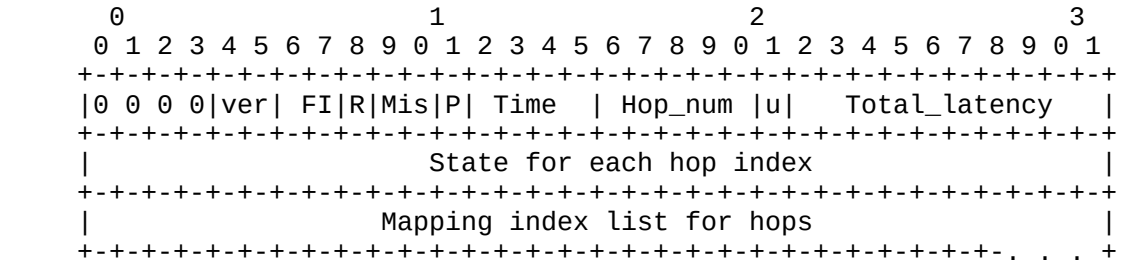
1. Proxy can check if a in-band signaling from end user for the SLA compliance, security and DOS attack prevention.

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



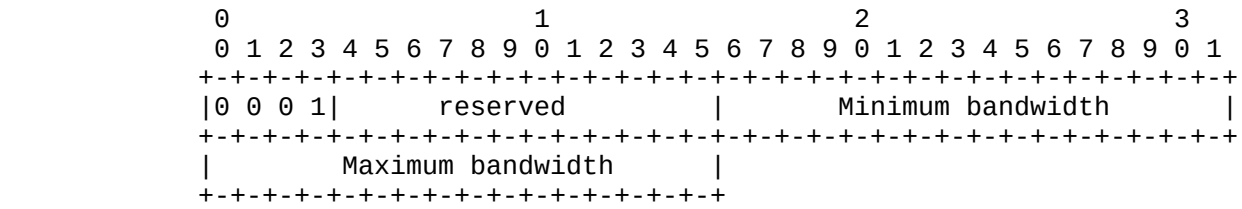
Type = 0, Setup state;
Version: The version of the protocol for the QoS
FI: Flow identification method,
0: 5 tuples; 1: src,dst,TCP; 2: src,dst,UDP; 3: src,dst
R: If the destination host report the received Setup state to the src address by Destination EH. 0: dont report; 1: report
Mis: Mapping index 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 hader.

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.

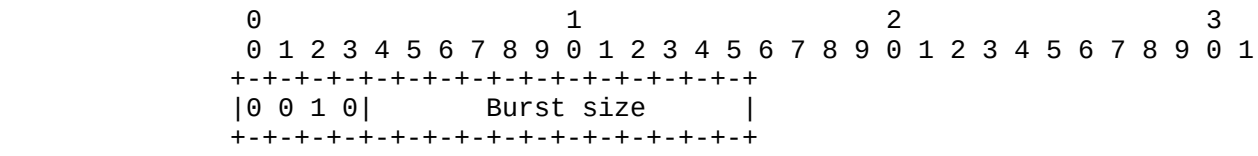
6.2. Bandwidth Msg



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

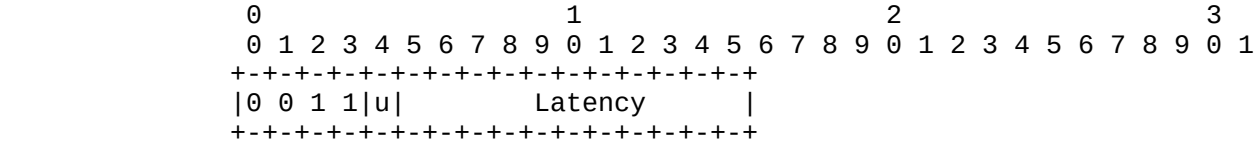
6.3. Burst Msg



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

Figure 4: The burst message

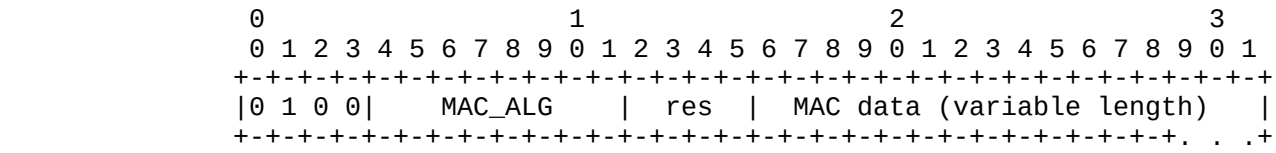
6.4. Latency Msg



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

Figure 5: The Latency message

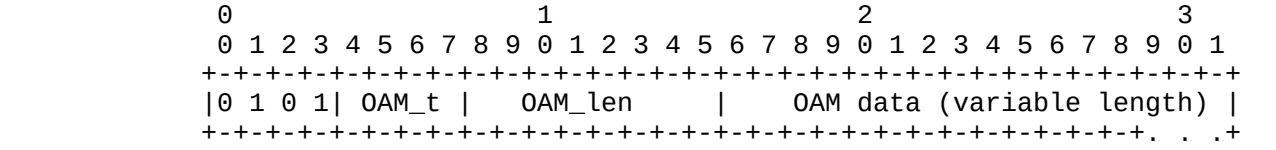
6.5. Authentication Msg



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

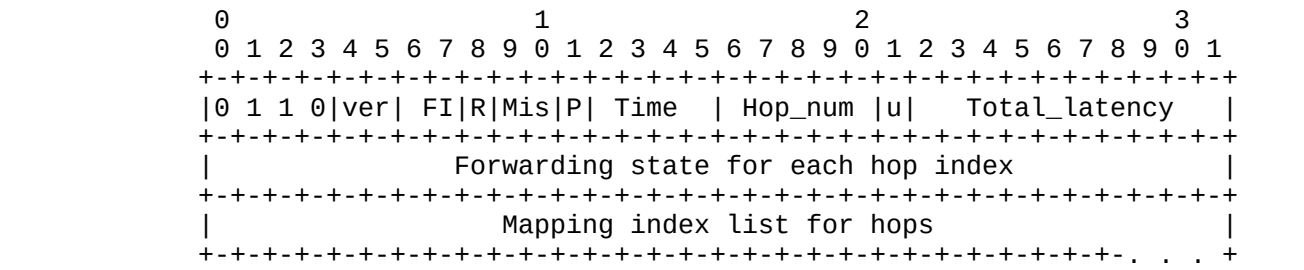
6.6. OAM Msg



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

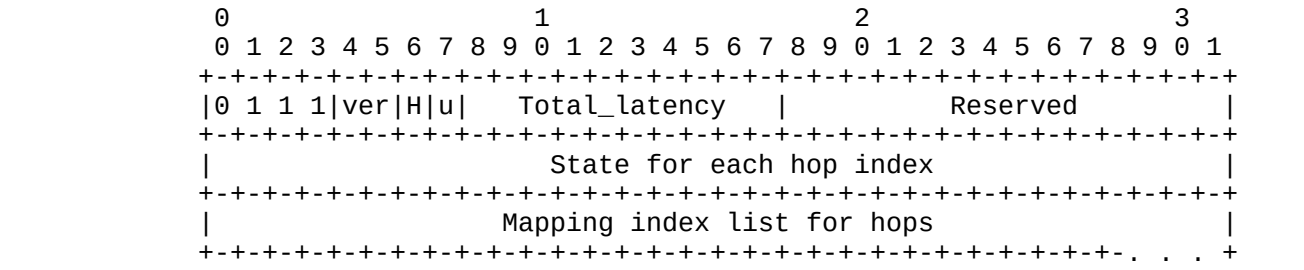
6.7. Forwarding State Msg



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

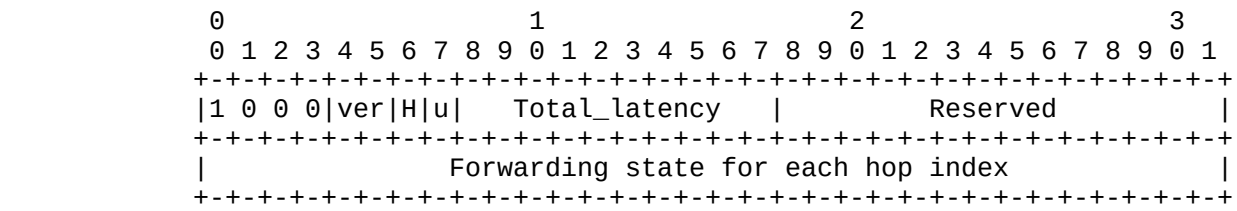
6.8. Setup State Report Msg



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-iccrg-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-iccrg-arvr-transport-problem]
Han, L. and K. Smith, "Problem Statement: Transport Support for Augmented and Virtual Reality Applications", draft-han-iccrg-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., McKenney, 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>.

[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>.

[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>.

[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: May 8, 2019

K. De Schepper
Nokia Bell Labs
B. Briscoe, Ed.
CableLabs
O. Bondarenko
Simula Research Lab
I. Tsang
Nokia

November 04, 2018

DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)
draft-ietf-tsvwg-aqm-dualq-coupled-08

Abstract

The Low Latency Low Loss Scalable Throughput (L4S) architecture allows data flows over the public Internet to predictably achieve ultra-low queuing latency, generally zero congestion loss and scaling of per-flow throughput without the problems of traditional TCP. To achieve this, L4S data flows use a 'scalable' congestion control similar to Data Centre TCP (DCTCP) and a form of Explicit Congestion Notification (ECN) with modified behaviour. However, until now, scalable congestion controls did not co-exist with existing TCP Reno/ Cubic traffic---scalable controls are so aggressive that 'Classic' TCP algorithms drive themselves to starvation. Therefore, until now, L4S controls could only be deployed where a clean-slate environment could be arranged, such as in private data centres (hence the name DCTCP). This specification defines 'DualQ Coupled Active Queue Management (AQM)', which enables these scalable congestion controls to safely co-exist with Classic Internet traffic.

The Coupled AQM ensures that a flow runs at about the same rate whether it uses DCTCP or TCP Reno/Cubic. It achieves this indirectly, without having to inspect transport layer flow identifiers, When tested in a residential broadband setting, DCTCP also achieves sub-millisecond average queuing delay and zero congestion loss under a wide range of mixes of DCTCP and 'Classic' broadband Internet traffic, without compromising the performance of the Classic traffic. The solution also reduces network complexity and eliminates network configuration.

Table of Contents

1. Introduction

1.1. Problem and Scope

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

The Diffserv architecture provides Expedited Forwarding [RFC3246], so that low latency traffic can jump the queue of other traffic. However, on access links dedicated to individual sites (homes, small enterprises or mobile devices), often all traffic at any one time will be latency-sensitive and, if all the traffic on a link is marked as EF, Diffserv cannot reduce the delay of any of it. In contrast, the Low Latency Low Loss Scalable throughput (L4S) approach removes the causes of any unnecessary queuing delay.

The bufferbloat project has shown that excessively-large buffering ('bufferbloat') has been introducing significantly more delay than the underlying propagation time. These delays appear only intermittently--only when a capacity-seeking (e.g. TCP) flow is long enough for the queue to fill the buffer, making every packet in other flows sharing the buffer sit through the queue.

Active queue management (AQM) was originally developed to solve this problem (and others). Unlike Diffserv, which gives low latency to

some traffic at the expense of others, AQM controls latency for _all_ traffic in a class. In general, AQMs introduce an increasing level of discard from the buffer the longer the queue persists above a shallow threshold. This gives sufficient signals to capacity-seeking (aka. greedy) flows to keep the buffer empty for its intended purpose: absorbing bursts. However, RED [RFC2309] and other algorithms from the 1990s were sensitive to their configuration and hard to set correctly. So, AQM was not widely deployed in the 1990s.

More recent state-of-the-art AQMs, e.g. fq_CoDel [RFC8290], PIE [RFC8033], Adaptive RED [ARED01], are easier to configure, because they define the queuing threshold in time not bytes, so it is invariant for different link rates. However, no matter how good the AQM, the sawtoothing rate of TCP will either cause queuing delay to vary or cause the link to be under-utilized. Even with a perfectly tuned AQM, the additional queuing delay will be of the same order as the underlying speed-of-light delay across the network. Flow-queuing can isolate one flow from another, but it cannot isolate a TCP flow from the delay variations it inflicts on itself, and it has other problems - it overrides the flow rate decisions of variable rate video applications, it does not recognise the flows within IPSec VPN tunnels and it is relatively expensive to implement.

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

1. the 'sawtooth' varying rate of TCP itself;

2. the smoothing delay deliberately introduced into AQMs to permit bursts without triggering losses.

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

Latency is not our only concern:

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

Given regular broadband bit-rates over WAN distances are already [RFC3649] beyond the scaling range of 'classic' TCP Reno, 'less unscalable' Cubic [RFC8312] and Compound [I-D.sridharan-tcpm-ctcp] variants of TCP have been

successfully deployed. However, these are now approaching their scaling limits. Unfortunately, fully scalable TCPs such as DCTCP cause 'classic' TCP to starve itself, which is why they have been confined to private data centres or research testbeds (until now).

This document specifies a 'DualQ Coupled AQM' extension that solves the problem of coexistence between scalable and classic flows, without having to inspect flow identifiers. The AQM is not like flow-queuing approaches [RFC8290] that classify packets by flow identifier into numerous separate queues in order to isolate sparse flows from the higher latency in the queues assigned to heavier flows. In contrast, the AQM exploits the behaviour of scalable congestion controls like DCTCP so that every packet in every flow sharing the queue for DCTCP-like traffic can be served with very low latency.

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

For the public Internet, nearly all the benefit will typically be achieved by deploying the Coupled AQM into either end of the access link between a 'site' and the Internet, which is invariably the bottleneck. Here, the term 'site' is used loosely to mean a home, an office, a campus or mobile user equipment.

The overall L4S architecture [I-D.ietf-tsvwg-l4s-arch] gives more detail, including on wider deployment aspects such as coexistence in bottlenecks where a DualQ Coupled AQM has not been deployed. The supporting papers [PI2] and [DCttH15] give the full rationale for the AQM's design, both discursively and in more precise mathematical form.

1.2. 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] when, and only when, they appear in all capitals, as shown here.

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

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

Low-Latency, Low-Loss and Scalable (L4S, denoted by subscript L): The 'L4S' service is intended for a set of congestion controls with scalable properties (e.g. DCTCP [RFC8257], Relentless TCP [Mathis09], the L4S variant of SCREAM for real-time media {ToDo: ref}). For the public Internet a scalable control has to comply with the requirements in [I-D.ietf-tsvwg-ecn-l4s-id] (aka. the 'TCP Prague requirements').

Either service can cope with a proportion of unresponsive or less-responsive traffic as well, as long (e.g. DNS, VoIP, game sync datagrams, etc), just as a single queue AQM can if this traffic makes minimal contribution to queuing. The DualQ Coupled AQM behaviour below is defined to be similar to a single FIFO queue with respect to unresponsive and overload traffic.

1.3. Features

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

Subjective testing was also conducted by multiple people all simultaneously using very demanding high bandwidth low latency applications over a single shared access link [L4Sdemo16]. In one application, each user could use finger gestures to pan or zoom their own high definition (HD) sub-window of a larger video scene generated on the fly in 'the cloud' from a football match. Another user wearing VR goggles was remotely receiving a feed from a 360-degree camera in a racing car, again with the sub-window in their field of

vision generated on the fly in 'the cloud' dependent on their head movements. Even though other users were also downloading large amounts of L4S and Classic data, playing a gaming benchmark and watchings videos over the same 40Mb/s downstream broadband link, latency was so low that the football picture appeared to stick to the user's finger on the touchpad and the experience fed from the remote camera did not noticeably lag head movements. All the L4S data (even including the downloads) achieved the same ultra-low latency. With an alternative AQM, the video noticeably lagged behind the finger gestures and head movements.

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

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

2. DualQ Coupled AQM

There are two main aspects to the approach:

- the Coupled AQM that addresses throughput equivalence between Classic (e.g. Reno, Cubic) flows and L4S flows (that satisfy the TCP Prague requirements).

- the Dual Queue structure that provides latency separation for L4S flows to isolate them from the typically large Classic queue.

2.1. Coupled AQM

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

We focus on Reno as the worst case, because if we do not harm Reno, we will not harm Cubic. Nonetheless, TCP Cubic implements a Reno-compatibility mode, which is the only relevant mode for typical RTTs under 20ms as long as the throughput of a single flow is less than about 500Mb/s. Therefore it can be assumed that Cubic traffic behaves similarly to Reno (but with a slightly different constant of proportionality). The term 'Classic' will be used for the collection of Reno-friendly traffic including Cubic in Reno mode.

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

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

Also, to remain stable, Classic sources need the network to smooth p_C so it changes relatively slowly. In contrast, L4S avoids smoothing in the network, because it delays all signals for a worst-case RTT. So instead, L4S sources smooth the ECN marking probability themselves, so they expect the network to generate ECN marks with a probability p_L that tracks the instantaneous unsmoothed queue.

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

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

$$p_C = (p_CL / k)^2 \quad (1)$$

where k is the constant of proportionality, which we shall call the coupling factor.

2.2. Dual Queue

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

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

2.3. Traffic Classification

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

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

In addition, other identifiers could be used to classify certain additional packet types into the L queue, that are deemed not to risk harming the L4S service. For instance addresses of specific applications or hosts (see [I-D.ietf-tsvwg-ecn-l4s-id]), specific

Diffserv codepoints such as EF (Expedited Forwarding) and Voice-Admit service classes (see [I-D.briscoe-tsvwg-l4s-diffserv]) or certain protocols (e.g. ARP, DNS).

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

2.4. Overall DualQ Coupled AQM Structure

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

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

$$p\_C = (p')^2. \qquad (2)$$

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:

$$p\_CL = k*p', \qquad (3)$$

where k is the constant coupling factor (see Appendix C).

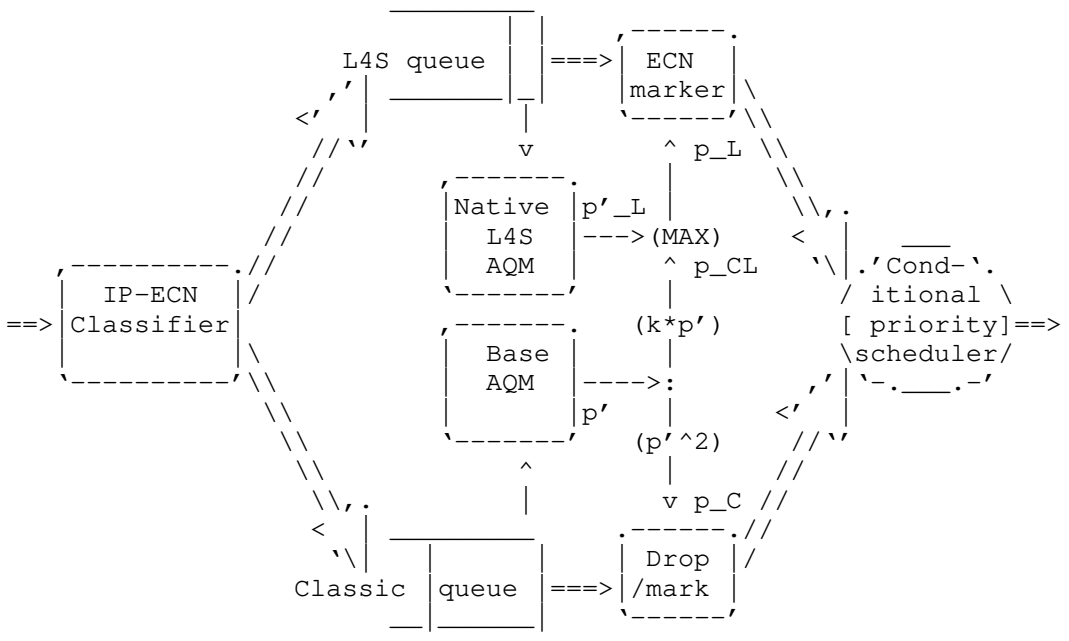
It can be seen that these two transformations of p' implement the required coupling given in equation (1) earlier.

The actual probability p_L that we apply to the L queue needs to track the immediate L queue delay, as well as track p_CL under stationary conditions. So we use a native AQM in the L queue that

calculates a probability p'_L as a function of the instantaneous L queue. And, given the L queue has conditional strict priority over the C queue, whenever the L queue grows, we should apply marking probability p'_L, but p_L should not fall below p_CL. This suggests:

p_L = max(p'_L, p_CL), (4)

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



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

Figure 1: DualQ Coupled AQM Schematic

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

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

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

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

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

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

2.5. Normative Requirements for a DualQ Coupled AQM

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

2.5.1. Functional Requirements

A Dual Queue Coupled AQM implementation MUST utilize two queues, each with an AQM algorithm. The two queues can be part of a larger queuing hierarchy [I-D.briscoe-tsvwg-l4s-diffserv].

The AQM algorithm for the low latency (L) queue MUST apply ECN marking.

The scheduler draining the two queues MUST give L4S packets priority over Classic, although priority MUST be bounded in order not to starve Classic traffic.

[I-D.ietf-tsvwg-ecn-l4s-id] defines the meaning of an ECN marking on L4S traffic, relative to drop of Classic traffic. In order to prevent starvation of Classic traffic by scalable L4S traffic, it says, "The likelihood that an AQM drops a Not-ECT Classic packet

(p_C) MUST be roughly proportional to the square of the likelihood that it would have marked it if it had been an L4S packet (p_L)." The term 'likelihood' is used to allow for marking and dropping to be either probabilistic or deterministic.

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

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

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

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

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

2.5.1.1. Requirements in Unexpected Cases

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

decides whether to apply congestion notification to this particular packet, as follows:

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

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

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

2.5.2. Management Requirements

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

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

An experimental DualQ Coupled AQM SHOULD allow the operator to monitor 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 arriving, enqueued and dequeued to distinguish tail discard from proactive AQM discard;
- ECN packets marked, non-ECN packets dropped, ECN packets dropped, from which marking and dropping probabilities can be calculated;
- 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.

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.

[RFC5706] suggests that deployment, coexistence and scaling should also be covered as management requirements. The raison d'etre of the DualQ Couple AQM is to enable deployment and coexistence of scalable congestion controls - as incremental replacements for today's TCP-friendly controls that do not scale with bandwidth-delay product. Therefore, these motivating issues are explained in the Introduction and detailed in the L4S architecture [I-D.ietf-tsvwg-l4s-arch]. Also, the descriptions of specific DualQ Coupled AQM algorithms in the appendices cover scaling of their configuration parameters, e.g. with respect to RTT and sampling frequency.

3. IANA Considerations

This specification contains no IANA considerations.

4. Security Considerations

4.1. Overload Handling

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

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

Under overload the higher priority L4S service will have to sacrifice some aspect of its performance. Alternative solutions are provided below that each relax a different factor: e.g. throughput, delay, drop. These choices need to be made either by the developer or by operator policy, rather than by the IETF.

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

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

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

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

Sacrifice L4S Delay: To control milder overload of responsive traffic, particularly when close to the maximum congestion signal, the operator could choose to control overload of the Classic queue by allowing some delay to 'leak' across to the L4S queue. The

scheduler can be made to behave like a single First-In First-Out (FIFO) queue with different service times by implementing a very simple conditional priority scheduler that could be called a "time-shifted FIFO" (see the Modifier Earliest Deadline First (MEDF) scheduler of [MEDF]). This scheduler adds tshift to the queue delay of the next L4S packet, before comparing it with the queue delay of the next Classic packet, then it selects the packet with the greater adjusted queue delay. Under regular conditions, this time-shifted FIFO scheduler behaves just like a strict priority scheduler. But under moderate or high overload it prevents starvation of the Classic queue, because the time-shift (tshift) defines the maximum extra queuing delay of Classic packets relative to L4S.

The example implementation in Appendix A can implement either policy.

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

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

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

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

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

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

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

4.1.3. Protecting against Unresponsive ECN-Capable Traffic

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

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

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

5. Acknowledgements

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

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

6. References

6.1. Normative References

[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

[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>.

[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, July 2015, <http://www.bobbriscoe.net/projects/latency/credi_tr.pdf>.

[DCttH15] De Schepper, K., Bondarenko, O., Briscoe, B., and I. Tsang, "'Data Centre to the Home': Ultra-Low Latency for All", 2015, <http://www.bobbriscoe.net/projects/latency/dctth_preprint.pdf>.

(Under submission)

[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.

[I-D.briscoe-tsvwg-l4s-diffserv] Briscoe, B., "Interactions between Low Latency, Low Loss, Scalable Throughput (L4S) and Differentiated Services", draft-briscoe-tsvwg-l4s-diffserv-00 (work in progress), March 2018.

[I-D.ietf-tsvwg-ecn-l4s-id]
Schepper, K., Briscoe, B., and I. Tsang, "Identifying Modified Explicit Congestion Notification (ECN) Semantics for Ultra-Low Queuing Delay", draft-ietf-tsvwg-ecn-l4s-id-02 (work in progress), March 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.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.

[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)>.

[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.

[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>.

(To appear)

[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>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <https://www.rfc-editor.org/info/rfc3168>.

[RFC3246] Davie, B., Charny, A., Bennet, J., Benson, K., Le Boudec, J., Courtney, W., Davari, S., Firoiu, V., and D. Stiliadis, "An Expedited Forwarding PHB (Per-Hop Behavior)", RFC 3246, DOI 10.17487/RFC3246, March 2002, <https://www.rfc-editor.org/info/rfc3246>.

[RFC3649] Floyd, S., "HighSpeed TCP for Large Congestion Windows", RFC 3649, DOI 10.17487/RFC3649, December 2003, <https://www.rfc-editor.org/info/rfc3649>.

[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., 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>.

[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>.

[TCP-CA] Jacobson, V. and M. Karels, "Congestion Avoidance and Control", Laurence Berkeley Labs Technical Report , November 1988, <http://ee.lbl.gov/papers/congavoid.pdf>.

Appendix A. Example DualQ Coupled PI2 Algorithm

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

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

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

A.1. Pass #1: Core Concepts

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

- initialization code (Figure 2) that sets parameter defaults (the API for setting non-default values is omitted for brevity)
- enqueue code (Figure 3)
- dequeue code (Figure 4)
- a ramp function (Figure 5) used to calculate the ECN-marking probability for the L4S queue
- code to regularly update the base probability (p) used in the dequeue code (Figure 6).

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

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

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

In our experiments so far (building on experiments with PIE) on broadband access links ranging from 4 Mb/s to 200 Mb/s with base RTTs from 5 ms to 100 ms, DualPI2 achieves good results with the default parameters in Figure 2. The parameters are categorised by whether they relate to the Base PI2 AQM, the L4S AQM or the framework coupling them together. Variables derived from these parameters are also included at the end of each category. Each parameter is explained as it is encountered in the walk-through of the pseudocode below.

```
1:  dualpi2_params_init(...) {         % Set input parameter defaults
2:    % PI2 AQM parameters
3:    target = 15 ms              % PI AQM Classic queue delay target
4:    Tupdate = 16 ms            % PI Classic queue sampling interval
5:    alpha = 10 Hz^2                              % PI integral gain
6:    beta = 100 Hz^2                          % PI proportional gain
7:    p_Cmax = 1/4                       % Max Classic drop/mark prob
8:    % Constants derived from PI2 AQM parameters
9:    alpha_U = alpha *Tupdate % PI integral gain per update interval
10:   beta_U = beta * Tupdate  % PI prop'nal gain per update interval
11:
12:   % DualQ Coupled framework parameters
13:   k = 2                                          % Coupling factor
14:   % scheduler weight or equival't parameter (scheduler-dependent)
15:   limit = MAX_LINK_RATE * 250 ms               % Dual buffer size
16:
17:   % L4S ramp AQM parameters
18:   minTh = 475 us        % L4S min marking threshold in time units
19:   range = 525 us               % Range of L4S ramp in time units
20:   Th_len = 2 * MTU           % Min L4S marking threshold in bytes
21:   % Constants derived from L4S AQM parameters
22:   p_Lmax = min(k*sqrt(p_Cmax), 1)          % Max L4S marking prob
23:   floor = Th_len * 8 / MIN_LINK_RATE   % MIN_LINK_RATE is in Mb/s
24:   if (minTh < floor) {
25:     % Adjust ramp to exceed serialization time of 2 MTU
26:     range = max(range - (floor-minTh), 1)   % 1us avoids /0 error
27:     minTh = floor
28:   }
29:   maxTh = minTh+range   % L4S min marking threshold in time units
30: }
```

Figure 2: Example Header Pseudocode for DualQ Coupled PI2 AQM

For brevity the pseudocode shows some parameters in units of microseconds (us), but a real implementation would probably use nanoseconds.

The overall goal of the code is to maintain the base probability (p), which is an internal variable from which the marking and dropping probabilities for L4S and Classic traffic (p_L and p_C) are derived. The variable named p in the pseudocode and in this walk-through is the same as p' (p-prime) in Section 2.4. The probabilities p_L and p_C are derived in lines 3, 4 and 5 of the dualpi2_update() function (Figure 6) then used in the dualpi2_dequeue() function (Figure 4). The code walk-through below builds up to explaining that part of the code eventually, but it starts from packet arrival.

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

Figure 3: Example Enqueue Pseudocode for DualQ Coupled PI2 AQM

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

Figure 4: Example Dequeue Pseudocode for DualQ Coupled PI2 AQM

When packets arrive, first a common queue limit is checked as shown in line 2 of the enqueuing pseudocode in Figure 3. Note that the limit is deliberately tested before enqueue to avoid any bias against larger packets (so depending whether the implementation stores a packet while testing whether to drop it from the tail, it might be necessary for the actual buffer memory to be one MTU larger than limit).

Line 2 assumes an implementation where lq and cq share common buffer memory. An alternative implementation could use separate buffers for

each queue, in which case the arriving packet would have to be classified first to determine which buffer to check for available space. The choice is a trade off; a shared buffer can use less memory whereas separate buffers isolate the L4S queue from tail-drop due to large bursts of Classic traffic (e.g. a Classic TCP during slow-start over a long RTT).

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

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

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

- If an L4S packet is scheduled, lines 7 and 8 ECN-mark the packet if a random marking decision is drawn according to p_L. 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 (default 16ms) (see Figure 2).

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

The Native L4S AQM algorithm (Figure 5) is a ramp function, similar to the RED algorithm, but simpler due to the following differences:

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

- It uses instantaneous queueing delay to remove smoothing delay (L4S senders smooth incoming ECN feedback when necessary).

- The ramp rises linearly directly from 0 to 1, not to a 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 being experimented with instead of randomness; to reduce the delay necessary to smooth out the noise of randomness from the signal. In this case, for each packet, the algorithm would accumulate p_L in a counter and mark the packet that took the counter over 1, then subtract 1 from the counter and continue.

This 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 the parameter initialization code in Figure 2. A minimum marking threshold parameter (Th_len) in transmission units (default 2 MTU) is also necessary to ensure that the ramp does not trigger excessive marking on slow links. The code in lines 23-28 of Figure 2 converts 2 MTU into time units and adjusts the ramp thresholds to be no shallower than this floor.

An operator can effectively turn the ramp into a step function, as used by DCTCP, by setting the range to its minimum value (e.g. 1 ns). Then the condition for the ramp calculation will hardly ever arise. There is 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 seems to be 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.

```
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 would use a bit-shift
6:    else
7:      return 0
8:  }
```

Figure 5: Example Pseudocode for the Native L4S AQM

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

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

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 and implicitly takes the current queuing delay as 0 if the queue is empty.

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

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

alpha_U and beta_U are derived from the input parameters alpha and beta (see lines 5 and 6 of Figure 2). These recommended values of alpha and beta come from the stability analysis in [PI2] so that the

AQM can change p as fast as possible in response to changes in load without over-compensating and therefore causing oscillations in the queue.

alpha and beta determine how much p ought to change if it was updated every second. It is best to update p as frequently as possible, but the update interval (Tupdate) will probably be constrained by hardware performance. For link rates from 4 - 200 Mb/s, we found Tupdate=16ms (as recommended in [RFC8033]) is sufficient. However small the chosen value of Tupdate, p should change by the same amount per second, but in finer more frequent steps. So the gain factors used for updating p in Figure 6 need to be scaled by (Tupdate/1s), which is done in lines 9 and 10 of Figure 2). The suffix '_U' represents 'per update time' (Tupdate).

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

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

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

{ToDo: Scaling beta with Tupdate and scaling both alpha & beta with RTT}

A.2. Pass #2: Overload Details

Figure 7 repeats the dequeue function of Figure 4, but with overload details added. Similarly Figure 8 repeats the core PI algorithm of Figure 6 with overload details added. The initialization, enqueue and L4S AQM functions are unchanged.

In line 7 of the initialization function (Figure 2), the default maximum Classic drop probability p_Cmax = 1/4 or 25%. This is the point at which it is deemed that the Classic queue has become persistently overloaded, so it switches to using solely drop, even

for ECN-capable packets. This protects the queue against any unresponsive traffic that falsely claims that it is responsive to ECN marking, as required by [RFC3168] and [RFC7567].

Line 22 of the initialization function translates this into a maximum L4S marking probability (p_Lmax) by rearranging Equation (1). With a coupling factor of k=2 (the default) or greater, this translates to a maximum L4S marking probability of 1 (or 100%). This is intended to ensure that the L4S queue starts to introduce dropping once marking saturates and can rise no further. The 'TCP Prague' requirements [I-D.ietf-tsvwg-ecn-l4s-id] state that, when an L4S congestion control detects a drop, it falls back to a response that coexists with 'Classic' TCP. So it is correct that the L4S queue drops packets proportional to p^2, as if they are Classic packets.

Both these switch-overs are triggered by the tests for overload introduced in lines 4b and 12b of the dequeue function (Figure 7). Lines 8c to 8g drop L4S packets with probability p^2. Lines 8h to 8i mark the remaining packets with probability p_CL. If p_Lmax = 1, which is the suggested default configuration, all remaining packets will be marked because, to have reached the else block at line 8b, p_CL >= 1.

Lines 2c to 2d in the core PI algorithm (Figure 8) deal with overload of the L4S queue when there is no Classic traffic. This is necessary, because the core PI algorithm maintains the appropriate drop probability to regulate overload, but it depends on the length of the Classic queue. If there is no Classic queue the naive algorithm in Figure 6 drops nothing, even if the L4S queue is overloaded - so tail drop would have to take over (lines 3 and 4 of Figure 3).

If the test at line 2a finds that the Classic queue is empty, line 2d measures the current queue delay using the L4S queue instead. While the L4S queue is not overloaded, its delay will always be tiny compared to the target Classic queue delay. So p_L will be driven to zero, and the L4S queue will naturally be governed solely by threshold marking (lines 5 and 6 of the dequeue algorithm in Figure 7). But, if unresponsive L4S source(s) cause overload, the DualQ transitions smoothly to L4S marking based on the PI algorithm. And as overload increases, it naturally transitions from marking to dropping by the switch-over mechanism already described.

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

Figure 7: Example Dequeue Pseudocode for DualQ Coupled PI2 AQM (Including Integer Arithmetic and Overload Code)

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

Figure 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 4.1).

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

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

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

Appendix B. Example DualQ Coupled Curvy RED Algorithm

As another example of a DualQ Coupled AQM algorithm, the pseudocode below gives the Curvy RED based algorithm we used and tested. Although we designed the AQM to be efficient in integer arithmetic, to aid understanding it is first given using real-number arithmetic. Then, one possible optimization for integer arithmetic is given, also in pseudocode. To aid comparison, the line numbers are kept in step between the two by using letter suffixes where the longer code needs extra lines.

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

16: maxrand(u) {                % return the max of u random numbers
17:     maxr=0
18:     while (u-- > 0)
19:         maxr = max(maxr, rand())              % 0 <= rand() < 1
20:     return(maxr)
21: }
```

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

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

At the outer level, the structure of dualq_dequeue() implements strict priority scheduling. The code is written assuming the AQM is applied on dequeue (Note 1) . Every time dualq_dequeue() is called, the if-block in lines 2-6 determines whether there is an L4S packet to dequeue by calling lq.dequeue(pkt), and otherwise the while-block in lines 7-13 determines whether there is a Classic packet to dequeue, by calling cq.dequeue(pkt). (Note 2)

In the lower priority Classic queue, a while loop is used so that, if the AQM determines that a classic packet should be dropped, it continues to test for classic packets deciding whether to drop each until it actually forwards one. Thus, every call to dualq_dequeue() returns one packet if at least one is present in either queue, otherwise it returns NULL at line 14. (Note 3)

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

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

Specifically, in line 3a the marking probability p_L is set to the Classic queueing time qc.sec() in seconds divided by the L4S scaling parameter 2^S_L, which represents the queuing time (in seconds) at which marking probability would hit 100%. Then in line 3d (if U=1) the result is compared with a uniformly distributed random number between 0 and 1, which ensures that marking probability will linearly increase with queueing time. The scaling parameter is expressed as a power of 2 so that division can be implemented as a right bit-shift (>>) in line 3 of the integer variant of the pseudocode (Figure 10).

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

the division needed to weight the moving average can be implemented by a right bit-shift (>> f_C).

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

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

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

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

Classic:

S_C : The scaling factor of the dropping function scales Classic queuing times in the range [0, 2^(S_C)] seconds into a dropping probability in the range [0,1]. To make division efficient, it is constrained to be an integer power of two;

f_C : To smooth the queuing time of the Classic queue and make multiplication efficient, we use a negative integer power of two for the dimensionless EWMA constant, which we define as alpha = 2^(-f_C).

L4S :

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

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

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

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

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

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

U=2, L4S would be a square function and Classic would be quartic. And so on.

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

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

Figure 10: Optimised Example Dequeue Pseudocode for Coupled DualQ AQM using Integer Arithmetic

Notes:

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

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

3. If packets can be enqueued while processing dequeue code, an implementer might prefer to place the while loop around both queues so that it goes back to test again whether any L4S packets arrived while it was dropping a Classic packet.

4. In order not to change too many factors at once, for now, we keep the marking function for DCTCP-only traffic as similar as possible to DCTCP. However, unlike DCTCP, all processing is at dequeue, so we determine whether to mark a packet at the head of the queue by the byte-length of the queue _behind_ it. We plan to test whether using queuing time will work in all circumstances, and if we find that the step can cause oscillations, we will investigate replacing it with a steep random marking curve.

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

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

7. In the integer variant of the pseudocode (Figure 10) real numbers are all represented as integers scaled up by 2^32. In lines 3 & 9 the function maxrand() is arranged to return an integer in the range 0 <= maxrand() < 2^32. Queuing times are also scaled up by 2^32, but in two stages: i) In lines 3 and 8 queuing times cq.ns() and pkt.ns() are returned in integer nanoseconds, making the values about 2^30 times larger than when the units were seconds, ii) then in lines 3 and 9 an adjustment of -2 to the right bit-shift multiplies the result by 2^2, to complete the scaling by 2^32.

Appendix C. Guidance on Controlling Throughput Equivalence

RTT_C / RTT_L	Reno	Cubic
1	k'=1	k'=0
2	k'=2	k'=1
3	k'=2	k'=2
4	k'=3	k'=2
5	k'=3	k'=3

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

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

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

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

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

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

$$2^{k'} = 1.64 \ (RTT_{reno} / RTT_{dc}) \quad (2)$$

$$2^{k'} = 1.19 \ (RTT_{cubic} / RTT_{dc}) \quad (3)$$

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

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

Appendix D. Open Issues

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

- PI2 appendix: scaling of alpha & beta, esp. dependence of beta_U on Tupdate

- Curvy RED appendix: complete the unfinished parts

Authors' Addresses

Koen De Schepper
Nokia Bell Labs
Antwerp
Belgium

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

Bob Briscoe (editor)
CableLabs
UK

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

Olga Bondarenko
Simula Research Lab
Lysaker
Norway

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

Ing-jyh Tsang
Nokia
Antwerp
Belgium

Email: ing-jyh.tsang@nokia.com

Transport Area Working Group
Internet-Draft
Updates: 3819 (if approved)
Intended status: Best Current Practice
Expires: May 16, 2019

B. Briscoe
Independent
J. Kaippallimalil
Huawei
P. Thaler
Broadcom Corporation
November 12, 2018

Guidelines for Adding Congestion Notification to Protocols that Encapsulate IP
draft-ietf-tsvwg-ecn-encap-guidelines-11

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 between new lower layer congestion notification mechanisms, whether specified by the IETF or other standards bodies.

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 16, 2019.

Copyright Notice

Table of Contents

1. Introduction

The benefits of Explicit Congestion Notification (ECN) described in [RFC8087] and summarized below can only be fully realised 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 a lower layer marks a packet with ECN, the marking needs to be explicitly propagated up the layers. The same is true if a buffer 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 an egress at any layer 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 equipment can signal congestion explicitly and it will propagate consistently into encapsulated (higher layer) headers, otherwise the signals will not reach their ultimate destination.

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

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

- It avoids the delay when recovering from congestion losses, which particularly benefits small flows or real-time flows, making their delivery time predictably short [RFC2884];

- As ECN is used more widely by end-systems, it will gradually remove the need to configure a degree of delay into buffers before they start to notify congestion (the cause of bufferbloat). This is because drop involves a trade-off between sending a timely signal and trying to avoid impairment, whereas ECN is solely a signal not an impairment, so there is no harm triggering it earlier.

Some lower layer technologies (e.g. MPLS, Ethernet) are used to form subnetworks with IP-aware nodes only at the edges. These networks are often sized so that it is rare for interior queues to overflow. However, until recently this was more due to the inability of TCP to saturate the links. For many years, fixes such as window scaling [RFC1323] (now [RFC7323]) proved hard to deploy. And the Reno

variant of TCP has remained in widespread use despite its inability to scale to high flow rates. However, now that modern operating systems are finally capable of saturating interior links, even the buffers of well-provisioned interior switches will need to signal episodes of queuing.

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

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

Incremental deployment is the most delicate aspect when adding support for ECN. The original ECN protocol in IP [RFC3168] was carefully designed so that a congested buffer would not mark a packet (rather than drop it) unless both source and destination hosts were ECN-capable. Otherwise its congestion markings would never be detected and congestion would just build up further. However, to support congestion marking below the IP layer or within tunnels, it is not sufficient to only check that the two layer 4 transport end-points 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 capitalised 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 to contradict 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.

This document updates the advice to subnetwork designers about ECN in Section 13 of [RFC3819].

1.1. 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, for example:

- by 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];
- or by 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 [RFC8311] [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. 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 descapsulation, as in the pipe model of [RFC2983], the special semantics of the ECN field will 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 [IEEE802.1Qah]) 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 it would be so complicated that it is hoped no-one would attempt such an abomination).

2. Terminology

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

Further terminology used within this document:

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

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

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

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

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

Outer header: The header added to encapsulate a PDU.

Inner header: The header encapsulated by the outer header.

Outgoing header: The header forwarded by the decapsulator.

CE: Congestion Experienced [RFC3168]

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

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

Load Regulator: For each flow of PDUs, the transport function that is capable of controlling the data rate. Typically located at the data source, but in-path nodes can regulate load in some congestion control arrangements (e.g. admission control, policing nodes or transport circuit-breakers [RFC8084]). Note the term "a function capable of controlling the load" deliberately includes a transport that doesn't 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 that is part of a feedback loop within which all the nodes that need to propagate explicit congestion notifications back to the Load Regulator are ECN-capable. 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 any layer, not just IP.

Not-ECN-PDU: A PDU that is part of a feedback-loop within which some nodes necessary to propagate explicit congestion notifications back to the load regulator are not ECN-capable.

Congestion Baseline: The location of the function on the path that initialised the values of all congestion notification fields in a sequence of packets, before any are set to the congestion experienced (CE) codepoint if they experience congestion further downstream. Typically the original data source at layer-4.

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 ECN field in the higher layer (e.g. 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. It will then also be necessary to define how the egress of the lower layer subnet propagates this explicit signal into the forwarded upper layer (IP) header. It can then continue 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).

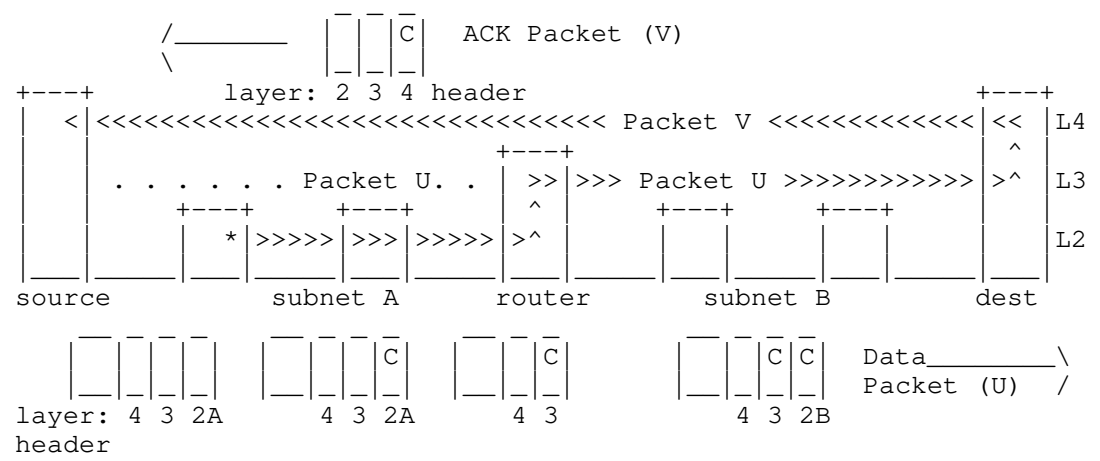


Figure 1: Feed-Forward-and-Up Mode

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) 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 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 [Buck00].

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 bury 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 doesn't use IP addresses, and it doesn't decrement the TTL field in the IP header.

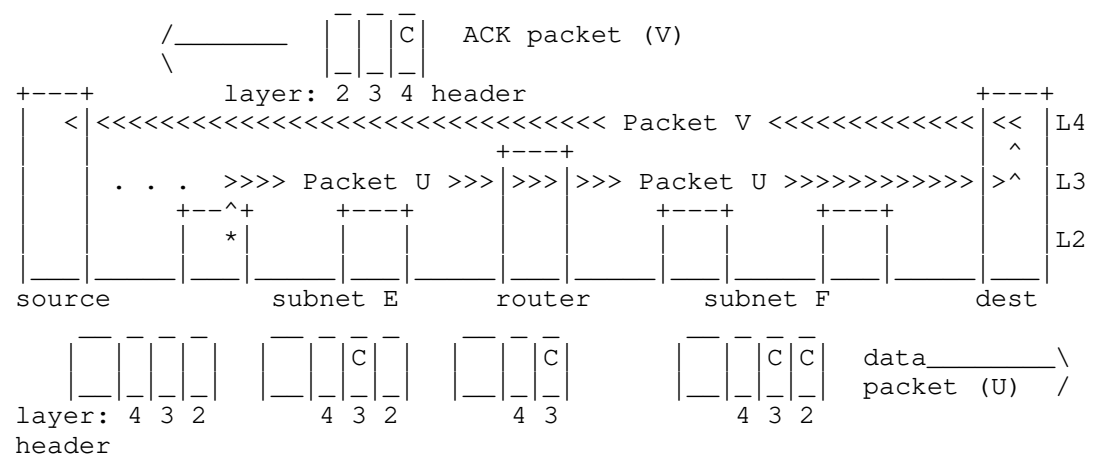


Figure 2: Feed-Up-and-Forward Mode

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.

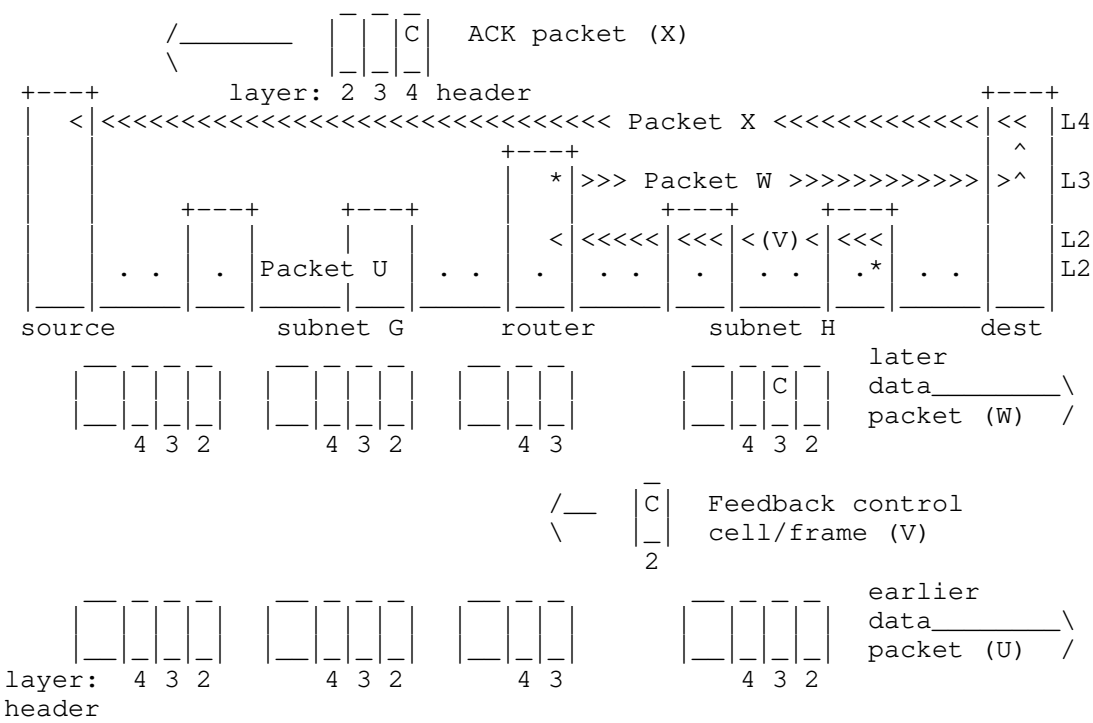


Figure 3: Feed-Backward Mode

ATM's feed-backward approach doesn't 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 doesn't 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 doesn't reach the original data source directly because IP doesn't 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.

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. 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.

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].

The rest of this section is structured as follows:

- Section 4.1 addresses the most straightforward cases, where [RFC6040] can be applied directly to add ECN to tunnels that are effectively IP-in-IP tunnels, but with shim header(s) between the IP headers.

- The subsequent sections give guidelines for adding ECN to a subnet technology that uses feed-forward-and-up mode like IP, but it is not so similar to IP that [RFC6040] rules can be applied directly. Specifically:

* Sections 4.2, 4.3 and 4.4 respectively address how to add ECN support to the wire protocol and to the encapsulators and decapsulators at the ingress and egress of the subnet.

* Section 4.5 deals with the special, but common, case of sequences of tunnels or subnets that all use the same technology

* Section 4.6 deals with the question of reframing when IP packets do not map 1:1 into lower layer frames.

4.1. IP-in-IP Tunnels with Shim Headers

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

However, it is not always feasible or necessary to propagate ECN between IP headers when separated by a shim. For instance, it might be too costly to dig to arbitrary depths to find an inner IP header, there may be little or no congestion within the tunnel by design (see null mode in Section 3.4 above), or a legacy implementation might not support ECN. In cases where a tunnel does not support ECN, it is important that the ingress does not copy the ECN field from an inner IP header to an outer. Therefore section 4 of [I-D.ietf-tsvwg-rfc6040update-shim] requires network operators to configure the ingress of a non-ECN tunnel 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 header and propagate ECN between that and the outer IP header. This can be thought of as a special case of the feed-up-and-forward mode (Section 3.2), so the guidelines for this mode apply (Section 5).

Numerous shim protocols have been defined for IP tunnelling. More recent ones e.g. Generic UDP Encapsulation (GUE) [I-D.ietf-intarea-gue] and Geneve [I-D.ietf-nvo3-geneve] cite and

follow RFC 6040. And some earlier ones, e.g. CAPWAP [RFC5415] and LISP [RFC6830], cite RFC 3168, which is compatible with RFC 6040.

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

All these IP-based encapsulations can be updated in one shot by simple reference to RFC 6040. However, it would not be appropriate to update all these protocols from within the present guidance document. Instead a companion specification [I-D.ietf-tsvwg-rfc6040update-shim] has been prepared that has sufficient 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 both 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 IEEE 802.1ah Provider Backbone Bridges (PBB).

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

4.3. Encapsulation Guidelines

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

1. Egress Capability Check: A subnet ingress needs to be sure that the corresponding egress of a subnet will propagate any congestion notification added to the outer header across the subnet. This is necessary in addition to checking that an incoming PDU indicates an ECN-capable (L4) transport. Examples of how this guarantee might be provided include:

* by configuration (e.g. if any label switches in a domain support ECN marking, [RFC5129] requires all egress nodes to have been configured to propagate ECN)

* by the ingress explicitly checking that the egress propagates ECN (e.g. TRILL uses IS-IS to check path capabilities before using critical options [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 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 when it forwards the PDU at the lower layer. 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 all outer headers reflect 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-initialise the level of CE markings in the outer to zero.

   This guideline is intended to ensure that any bulk congestion monitoring of outer headers (e.g. by a network management node monitoring ECN in passing frames) is most meaningful. For instance, if an operator measures CE in 0.4% of passing outer headers, this information is only useful if the operator knows where the proportion of CE markings was last initialised to 0% (the Congestion Baseline). Such monitoring information will not be useful if some subnet ingress nodes reset all outer CE markings while others copy incoming CE markings into the outer.

   Most information can be extracted if the Congestion Baseline is standardised at the node that is regulating the load (the Load Regulator--typically the data source). Then the operator can

measure both congestion since the Load Regulator, and congestion since the subnet ingress. The latter might be measurable by subtracting the level of CE markings on inner headers from that on outer headers (see Appendix C of [RFC6040]).

4.4. Decapsulation Guidelines

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

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

1. If the arriving inner header is a Not-ECN-PDU it implies the L4 transport will not understand explicit congestion markings. Then:

   * If the outer header carries an explicit congestion marking, drop is the only indication of congestion that the L4 transport will understand. If the congestion marking is the most severe possible, the packet MUST be dropped. However, if congestion can be marked with multiple levels severity and the packet's marking is not the most severe, the packet MAY be forwarded, but it SHOULD be dropped.

   * If the outer is an ECN-PDU that carries no indication of congestion or a Not-ECN-PDU the PDU SHOULD be forwarded, but still as a Not-ECN-PDU.

2. If the outer header does not support explicit congestion notification (a Not-ECN-PDU), but the inner header does (an ECN-PDU), the inner header SHOULD be forwarded unchanged.

3. In some lower layer protocols congestion may be signalled as a numerical level, such as in the control frames of quantised congestion notification [IEEE802.1Qau]. If such a multi-bit encoding encapsulates an ECN-capable IP data packet, a function will be needed to convert the quantised 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 framing boundaries are different between two layers, congestion indications SHOULD be propagated on the basis that a congestion indication on a PDU applies to all the octets in the PDU. On average, an encapsulator or decapsulator SHOULD approximately preserve the number of marked octets arriving and leaving (counting the size of inner headers, but not added encapsulating headers).

The next departing frame SHOULD be immediately marked even if only enough incoming marked octets have arrived for part of the departing frame. This ensures that any outstanding congestion marked octets are propagated immediately, rather than held back waiting for a frame no bigger than the outstanding marked octets--which might involve a long wait.

For instance, an algorithm for marking departing frames could maintain a counter representing the balance of arriving marked octets minus departing marked octets. It adds the size of every marked frame that arrives and if the counter is positive it marks the next frame to depart and subtracts its size from the counter. This will often leave a negative remainder in the counter, which is deliberate.

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

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

- are encapsulated in Ethernet headers, which have no support for ECN;

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

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

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

- IPv4 and IPv6 are not the only layer-3 protocols that might be encapsulated by lower layer protocols

- Link-layer encryption might be in use, making the layer-2 payload inaccessible

- Many Ethernet switches do not have 'layer-3 switch' capabilities so they cannot read or modify an IP payload

- It might be costly to find an IP header (v4 or v6) when it may be encapsulated by more than one lower layer header, e.g. Ethernet MAC in MAC [IEEE802.1Qah].

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 unrecognised 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 minimise 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.

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

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

This memo includes no request to IANA.

8. Security Considerations

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

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

- Congestion exposure (ConEx) for networks to audit that their congestion signals are not being suppressed by other networks or by receivers, and for networks to police that senders are responding sufficiently to the signals, irrespective of the L4 transport protocol used [RFC7713].

- A test for a sender to detect whether a network or the receiver is suppressing congestion signals (for example see 2nd para of Section 20.2 of [RFC3168]).

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

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

9. Conclusions

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

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

- A wide range of subnet technologies, particularly those that work in the same 'feed-forward-and-up' mode that is used to support ECN in IP and MPLS.

Guidelines have been defined for supporting propagation of ECN between Ethernet and IP on so-called Layer-3 Ethernet switches, using a 'feed-up-an-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 for extensive reviews. Thanks also to the following reviewers: Richard Scheffenegger, Ingemar Johansson, Piers O'Hanlon and Michael Welzl, who pointed out that lower layer congestion notification signals may have different semantics to those in IP. Thanks are also due to the tsvwg chairs, TSV ADs and IETF liaison people such as Eric Gray, Dan Romascanu and Gonzalo Camarillo for helping with the liaisons with the IEEE and 3GPP. And thanks to Georg Mayer and particularly to Erik Guttman for the extensive search and categorisation of any 3GPP specifications that cite ECN specifications.

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

11. Comments Solicited

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

12. References

12.1. Normative References

[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>.

12.2. 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.

[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. and O. Zia, "Generic UDP Encapsulation", draft-ietf-intarea-gue-06 (work in progress), August 2018.

[I-D.ietf-nvo3-geneve]
Gross, J., Ganga, I., and T. Sridhar, "Geneve: Generic Network Virtualization Encapsulation", draft-ietf-nvo3-geneve-08 (work in progress), October 2018.

[I-D.ietf-trill-ecn-support]
Eastlake, D. 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. and B. Briscoe, "Identifying Modified Explicit Congestion Notification (ECN) Semantics for Ultra-Low Queuing Delay (L4S)", draft-ietf-tsvwg-ecn-l4s-id-05 (work in progress), November 2018.

[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-06 (work in progress), March 2018.

[IEEE802.1Qah]
IEEE, "IEEE Standard for Local and Metropolitan Area Networks--Virtual Bridged Local Area Networks--Amendment 6: Provider Backbone Bridges", IEEE Std 802.1Qah-2008, August 2008, <http://www.ieee802.org/1/pages/802.1ah.html>.

(Access Controlled link within page)

[IEEE802.1Qau]
Finn, N., Ed., "IEEE Standard for Local and Metropolitan Area Networks--Virtual Bridged Local Area Networks - Amendment 13: Congestion Notification", IEEE Std 802.1Qau-2010, March 2010, <http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5454061>.

(Access Controlled link within page)

[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>.

[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.

[RFC1323] Jacobson, V., Braden, R., and D. Borman, "TCP Extensions for High Performance", RFC 1323, DOI 10.17487/RFC1323, May 1992, <https://www.rfc-editor.org/info/rfc1323>.

[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>.

[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>.

[UTRAN] 3GPP, "UTRAN Overall Description", Technical Specification TS 25.401.

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

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
Huawei
5340 Legacy Drive, Suite 175
Plano, Texas 75024
USA

EMail: john.kaippallimalil@huawei.com

Pat Thaler
Broadcom Corporation
5025 Keane Drive
Carmichael, CA 95608
USA

EMail: pthaler@broadcom.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.

Table of Contents

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:

- Declares that the ECN nonce experiment [RFC3540] has concluded, and notes the absence of widespread deployment.

- 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:

- The sender congestion response to CE marking in the network, and
- 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:

- "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)
- "A host MUST NOT set ECT on SYN or SYN-ACK packets." (Section 6.1.1)
- "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)
- "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, Gorry 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:

- 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:

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

- Add explanation of exception to "SHOULD NOT drop" requirement in 4.3
- 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.
- Significant editorial changes based on reviews by Mirja Kuehlewind, Michael Welzl and Bob Briscoe.

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

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

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

- Change name of "Generalized ECN" experimentation area to "TCP Control Packets and Retransmissions."
- 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:

- Minor editorial changes from Area Director review

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

- Add summary of RFC 3168 changes to remove the ECN nonce, and use lower-case "nonce" instead of "Nonce" to match RFC 3168 usage.
- 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.
- Other minor editorial changes from IETF Last Call

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

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

- Any Experimental RFC that takes advantage of this memo has to be in the IETF document stream.
- 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
- 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.
- 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
- Fix L4S draft reference for discussion of ECN Nonce alternatives - it's Appendix C.1, not B.1.
- Numerous additional editorial changes from IESG Evaluation

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

- 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: May 9, 2019

K. De Schepper
Nokia Bell Labs
B. Briscoe, Ed.
CableLabs

November 5, 2018

Identifying Modified Explicit Congestion Notification (ECN) Semantics for Ultra-Low Queuing Delay (L4S)
draft-ietf-tsvwg-ecn-l4s-id-05

Abstract

This specification defines the identifier to be used on IP packets for a new network service called low latency, low loss and scalable throughput (L4S). It is similar to the original (or 'Classic') Explicit Congestion Notification (ECN). 'Classic' ECN marking was required to be equivalent to a drop, both when applied in the network and when responded to by a transport. Unlike 'Classic' ECN marking, for packets carrying the L4S identifier, the network applies marking more immediately and more aggressively than drop, and the transport response to each mark is reduced and smoothed relative to that for drop. The two changes counterbalance each other so that the throughput of an L4S flow will be roughly the same as a 'Classic' flow under the same conditions. However, the much more frequent control signals and the finer responses to them result in ultra-low queuing delay without compromising link utilization, and low delay is maintained during high load. Examples of new active queue management (AQM) marking algorithms and examples of new transports (whether TCP-like or real-time) are specified separately. The new L4S identifier is the key piece that enables them to interwork and distinguishes them from 'Classic' traffic. It gives an incremental migration path so that existing 'Classic' TCP traffic will be no worse off, but it can be prevented from degrading the ultra-low delay and loss of the new scalable transports.

Status of This Memo

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

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 9, 2019.

Table of Contents

1. Introduction

This specification defines the identifier to be used on IP packets for a new network service called low latency, low loss and scalable throughput (L4S). It is similar to the original (or 'Classic') Explicit Congestion Notification (ECN [RFC3168]). 'Classic' ECN marking was required to be equivalent to a drop, both when applied in the network and when responded to by a transport. Unlike 'Classic' ECN marking, the network applies L4S marking more immediately and more aggressively than drop, and the transport response to each mark is reduced and smoothed relative to that for drop. The two changes counterbalance each other so that the throughput of an L4S flow will be roughly the same as a 'Classic' flow under the same conditions. Nonetheless, the much more frequent control signals and the finer responses to them result in ultra-low queuing delay without compromising link utilization, and low delay is maintained during high load.

An example of a scalable transport that would enable the L4S service is Data Centre TCP (DCTCP), which until now has been applicable solely to controlled environments like data centres [RFC8257], because it is too aggressive to co-exist with existing TCP. 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 transports like DCTCP to co-exist with existing traffic, each getting roughly the same flow rate when they compete under similar conditions. Note that a transport such as DCTCP is still not safe to deploy on the Internet unless it satisfies the requirements listed in Section 4.

The new L4S identifier is the key piece that enables L4S hosts and L4S network nodes to interwork and distinguishes their traffic from 'Classic' traffic. It gives an incremental migration path so that existing 'Classic' TCP traffic will be no worse off, but it can be prevented from degrading the ultra-low delay and loss of the new scalable transports. The performance improvement is so great that it is motivating initial deployment of the separate parts of this system.

1.1. Problem

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

The Diffserv architecture provides Expedited Forwarding [RFC3246], so that low latency traffic can jump the queue of other traffic. However, on access links dedicated to individual sites (homes, small enterprises or mobile devices), often all traffic at any one time will be latency-sensitive. Then Diffserv is of little use. Instead, we need to remove the causes of any unnecessary delay.

The bufferbloat project has shown that excessively-large buffering ('bufferbloat') has been introducing significantly more delay than the underlying propagation time. These delays appear only intermittently--only when a capacity-seeking (e.g. TCP) flow is long enough for the queue to fill the buffer, making every packet in other flows sharing the buffer sit through the queue.

Active queue management (AQM) was originally developed to solve this problem (and others). Unlike Diffserv, which gives low latency to some traffic at the expense of others, AQM controls latency for _all_ traffic in a class. In general, AQMs introduce an increasing level of discard from the buffer the longer the queue persists above a shallow threshold. This gives sufficient signals to capacity-seeking (aka. greedy) flows to keep the buffer empty for its intended purpose: absorbing bursts. However, RED [RFC2309] and other algorithms from the 1990s were sensitive to their configuration and hard to set correctly. So, AQM was not widely deployed.

More recent state-of-the-art AQMs, e.g. fq_CoDel [RFC8290], PIE [RFC8033], Adaptive RED [ARED01], are easier to configure, because they define the queuing threshold in time not bytes, so it is invariant for different link rates. However, no matter how good the AQM, the sawtoothing rate of TCP will either cause queuing delay to vary or cause the link to be under-utilized. Even with a perfectly tuned AQM, the additional queuing delay will be of the same order as the underlying speed-of-light delay across the network. Flow-queuing can isolate one flow from another, but it cannot isolate a TCP flow from the delay variations it inflicts on itself, and it has other problems - it overrides the flow rate decisions of variable rate video applications, it does not recognise the flows within IPSec VPN tunnels and it is relatively expensive to implement.

Latency is not our only concern: It was known when TCP was first developed that it would not scale to high bandwidth-delay products [TCP-CA]. Given regular broadband bit-rates over WAN distances are already [RFC3649] beyond the scaling range of 'Classic' TCP Reno, 'less unscalable' Cubic [RFC8312] and Compound [I-D.sridharan-tcpm-ctcp] variants of TCP have been successfully deployed. However, these are now approaching their scaling limits. Unfortunately, fully scalable TCPs such as DCTCP [RFC8257] cause 'Classic' TCP to starve itself, which is why they have been confined to private data centres or research testbeds (until now).

It turns out that a TCP algorithm like DCTCP that solves the latency problem also solves TCP's scalability problem. The finer sawteeth have low amplitude, so they cause very little queuing delay variation and the number of sawteeth per round trip remains invariant, which maintains constant tight control as flow-rate scales. A supporting paper [DCttH15] gives the full explanation of why the design solves both the latency and the scaling problems, both in plain English and in more precise mathematical form. The explanation is summarised without the maths in the L4S architecture document [I-D.ietf-tsvwg-l4s-arch].

1.2. Terminology

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

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

Low-Latency, Low-Loss and Scalable (L4S) service: The 'L4S' service is intended for traffic from scalable TCP algorithms such as Data Centre TCP. But it is also more general--it allows the set of congestion controls with similar scaling properties to DCTCP to evolve (e.g. Relentless TCP [Mathis09] and the L4S variant of SCREAM for real-time media [RFC8298].

Both Classic and L4S services can cope with a proportion of unresponsive or less-responsive traffic as well, as long as it does not build a queue (e.g. DNS, VoIP, game sync datagrams, etc).

Classic ECN: The original Explicit Congestion Notification (ECN) protocol [RFC3168].

1.3. Scope

The new L4S identifier defined in this specification is applicable for IPv4 and IPv6 packets (as for classic ECN [RFC3168]). It is applicable for the unicast, multicast and anycast forwarding modes.

The L4S identifier is an orthogonal packet classification to the Differentiated Services Code Point (DSCP [RFC2474]). Section 5.4 explains what this means in practice.

This document is intended for experimental status, so it does not update any standards track RFCs. Therefore it depends on [RFC8311], which is a standards track specification that:

- updates the ECN proposed standard [RFC3168] to allow experimental track RFCs to relax the requirement that an ECN mark must be equivalent to a drop, both when applied by the network, and when responded to by the sender;

- 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].

2. Consensus Choice of L4S Packet Identifier: Requirements

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

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

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

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

- it SHOULD be incrementally deployable;

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

- it SHOULD consume minimal extra codepoints;

- it SHOULD not lead to some packets of a transport-layer flow being served by a different queue from others.

Whether the identifier would be recoverable if the experiment failed is a factor that could be taken into account. However, this has not been made a requirement, because that would favour schemes that would be easier to fail, rather than those more likely to succeed.

It is recognised that the chosen identifier is unlikely to satisfy all these requirements, particularly given the limited space left in the IP header. Therefore a compromise will be necessary, which is why all the requirements are expressed with the word 'SHOULD' not 'MUST'. Appendix B discusses the pros and cons of the compromises

made in various competing identification schemes against the above requirements.

On the basis of this analysis, "ECT(1) and CE codepoints" is the best compromise. Therefore this scheme is defined in detail in the following sections, while Appendix B records the rationale for this decision.

3. L4S Packet Identification at Run-Time

The L4S treatment is an experimental track alternative packet marking treatment [RFC4774] to the classic ECN treatment [RFC3168], which has been updated by [RFC8311] to allow this experiment (amongst others). Like classic ECN, L4S ECN identifies both network and host behaviour: it identifies the marking treatment that network nodes are expected to apply to L4S packets, and it identifies packets that have been sent from hosts that are expected to comply with a broad type of sending behaviour.

For a packet to receive L4S treatment as it is forwarded, the sender sets the ECN field in the IP header to the ECT(1) codepoint. See Section 4 for full transport layer behaviour requirements, including feedback and congestion response.

A network node that implements the L4S service normally classifies arriving ECT(1) and CE packets for L4S treatment. See Section 5 for full network element behaviour requirements, including classification, ECN-marking and interaction of the L4S identifier with other identifiers and per-hop behaviours.

4. Prerequisite Transport Layer Behaviour

4.1. Prerequisite Codepoint Setting

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

4.2. Prerequisite Transport Feedback

In general, a scalable congestion control needs feedback of the extent of CE marking on the forward path. Due to the history of TCP development, when ECN was added TCP reported no more than one CE mark per round trip. Some transport protocols derived from TCP mimic this behaviour while others report the accurate extent of TCP marking. This means that some transport protocols will need to be updated as a prerequisite for scalable congestion control. The position for a few well-known transport protocols is given below.

TCP: Support for accurate ECN feedback (AccECN [I-D.ietf-tcpm-accurate-ecn]) by both ends is a prerequisite for scalable congestion control. Therefore, the presence of ECT(1) in the IP headers even in one direction of a TCP connection will imply that both ends support AccECN. However, the converse does not apply. So even if both ends support AccECN, either of the two ends can choose not to use a scalable congestion control, whatever the other end's choice.

SCTP: An ECN feedback protocol such as that specified in [I-D.stewart-tsvwg-sctpecn] would be a prerequisite for scalable congestion control. That draft would update the ECN feedback protocol sketched out in Appendix A of the standards track specification of SCTP [RFC4960] by adding a field to report the number of CE marks.

RTP over UDP: A prerequisite for scalable congestion control is for both (all) ends of one media-level hop to signal ECN support using the ecn-capable-rtp attribute [RFC6679]. Therefore, the presence of ECT(1) implies that both (all) ends of that hop support ECN. However, the converse does not apply, so each end of a media-level hop can independently choose not to use a scalable congestion control, even if both ends support ECN.

DCCP: The ACK vector in DCCP [RFC4340] is already sufficient to report the extent of CE marking as needed by a scalable congestion control.

4.3. Prerequisite Congestion Response

As a condition for a host to send packets with the L4S identifier (ECT(1)), it SHOULD implement a congestion control behaviour that ensures the flow rate is inversely proportional to the proportion of bytes in packets marked with the CE codepoint. This is termed a scalable congestion control, because the number of control signals (ECN marks) per round trip remains roughly constant for any flow rate. As with all transport behaviours, a detailed specification will need to be defined for each type of transport or application, including the timescale over which the proportionality is averaged, and control of burstiness. The inverse proportionality requirement above is worded as a 'SHOULD' rather than a 'MUST' to allow reasonable flexibility when defining these specifications.

Data Center TCP (DCTCP [RFC8257]) is an example of a scalable congestion control.

Each sender in a session can use a scalable congestion control independently of the congestion control used by the receiver(s) when

they send data. Therefore there might be ECT(1) packets in one direction and ECT(0) or Not-ECT in the other.

In order to coexist safely with other Internet traffic, a scalable congestion control MUST NOT tag its packets with the ECT(1) codepoint unless it complies with the following bulleted requirements. The specification of a particular scalable congestion control MUST describe in detail how it satisfies each requirement:

- A scalable congestion control MUST react to packet loss in a way that will coexist safely with a TCP Reno congestion control [RFC5681] (see Appendix A.1.3 for rationale).

- A scalable congestion control MUST react to ECN marking from a non-L4S but ECN-capable bottleneck in a way that will coexist with a TCP Reno congestion control [RFC5681] (see Appendix A.1.4 for rationale).

- A scalable congestion control MUST reduce or eliminate RTT bias over as wide a range of RTTs as possible, or at least over the typical range of RTTs that will interact in the intended deployment scenario (see Appendix A.1.5 for rationale).

- A scalable congestion control MUST remain responsive to congestion when the RTT is significantly smaller than in the current public Internet (see Appendix A.1.6 for rationale).

- A scalable congestion control MUST detect loss by counting in units of time, which is scalable, and MUST NOT count in units of packets (as in the 3 DupACK rule of traditional TCP), which is not scalable (see Appendix A.1.7 for rationale).

5. Prerequisite Network Node Behaviour

5.1. Prerequisite Classification and Re-Marking Behaviour

A network node that implements the L4S service MUST classify arriving ECT(1) packets for L4S treatment and it SHOULD classify arriving CE packets for L4S treatment as well. Section 5.3 describes a possible exception to this latter rule for some CE packets.

An L4S AQM treatment follows similar codepoint transition rules to those in RFC 3168. Specifically, the ECT(1) codepoint MUST NOT be changed to any other codepoint than CE, and CE MUST NOT be changed to any other codepoint. An ECT(1) packet is classified as ECN-capable and, if congestion increases, an L4S AQM algorithm will mark the ECN field as CE for an increasing proportion of packets, otherwise forwarding packets unchanged as ECT(1). Necessary conditions for an

L4S marking treatment are defined in Section 5.2. Under persistent overload an L4S marking treatment SHOULD turn off ECN marking, using drop as a congestion signal until the overload episode has subsided, as recommended for all AQMs in [RFC7567] (Section 4.2.1), which follows the similar advice in RFC 3168 (Section 7).

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

5.2. The Meaning of L4S CE Relative to Drop

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

$$p_C \tilde{=} (p_L / k)^2$$

The constant of proportionality (k) does not have to be standardised for interoperability, but a value of 2 is RECOMMENDED.

[I-D.ietf-tsvwg-aqm-dualq-coupled] specifies the essential aspects of an L4S AQM, as well as recommending other aspects. It gives example implementations in appendices.

The term 'likelihood' is used above to allow for marking and dropping to be either probabilistic or deterministic. The example AQMs in [I-D.ietf-tsvwg-aqm-dualq-coupled] drop and mark probabilistically, so the drop probability is arranged to be the square of the marking probability. Nonetheless, an alternative AQM that dropped and marked deterministically would be valid, as long as the dropping frequency was proportional to the square of the marking frequency.

Note that, contrary to RFC 3168, a Dual AQM implementing the L4S and Classic treatments does not mark an ECT(1) packet under the same conditions that it would have dropped a Not-ECT packet, as allowed by [RFC8311], which updates RFC 3168. However, it does mark an ECT(0) packet under the same conditions that it would have dropped a Not-ECT packet.

5.3. Exception for L4S Packet Identification by Network Nodes with Transport-Layer Awareness

To implement the L4S treatment, a network node does not need to identify transport-layer flows. Nonetheless, if an implementer is willing to identify transport-layer flows at a network node, and if the most recent ECT packet in the same flow was ECT(0), the node MAY classify CE packets for classic ECN [RFC3168] treatment. In all other cases, a network node MUST classify CE packets for L4S treatment. Examples of such other cases are: i) if no ECT packets have yet been identified in a flow; ii) if it is not desirable for a network node to identify transport-layer flows; or iii) if the most recent ECT packet in a flow was ECT(1).

If an implementer uses flow-awareness to classify CE packets, to determine whether the flow is using ECT(0) or ECT(1) it only uses the most recent ECT packet of a flow (this advice will need to be verified as part of L4S experiments). This is because a sender might have to switch from sending ECT(1) (L4S) packets to sending ECT(0) (Classic) packets, or back again, in the middle of a transport-layer flow. Such a switch-over is likely to be very rare, but It could be necessary if the path bottleneck moves from a network node that supports L4S to one that only supports Classic ECN. A host ought to be able to detect such a change from a change in RTT variation.

5.4. Interaction of the L4S Identifier with other Identifiers

5.4.1. Examples of Other Identifiers Complementing L4S Identifiers

5.4.1.1. Inclusion of Additional Traffic with L4S

In a typical case for the public Internet a network element that implements L4S might want to classify some low-rate but unresponsive traffic (e.g. DNS, voice, game sync packets) into the low latency queue to mix with L4S traffic. Such non-ECN-based packet types MUST be safe to mix with L4S traffic without harming the low latency service.

In this case it would not be appropriate to call the queue an L4S queue, because it is shared by L4S and non-L4S traffic. Instead it will be called the low latency or L queue. The L queue then offers two different treatments:

- The L4S treatment, which is a combination of the L4S AQM treatment and a priority scheduling treatment;
- The low latency treatment, which is solely the priority scheduling treatment, without ECN-marking by the AQM.

To identify packets for just the scheduling treatment, it would be inappropriate to use the L4S ECT(1) identifier, because such traffic is unresponsive to ECN marking. Therefore, a network element that implements L4S MAY classify additional packets into the L queue if they carry certain non-ECN identifiers. For instance:

- addresses of specific applications or hosts configured to be safe (but for example cannot set the ECN field for some temporary reason);
- certain protocols that are usually lightweight (e.g. ARP, DNS);
- specific Diffserv codepoints that indicate traffic with limited burstiness such as the EF (Expedited Forwarding) and Voice-Admit service classes or equivalent local-use DSCPs (see [I-D.briscoe-tsvwg-l4s-diffserv]).

For clarity, non-ECN identifiers, such as the examples itemized above, might be used by some network operators who believe they identify non-L4S traffic that would be safe to mix with L4S traffic. They are not alternative ways for a host to indicate that it is sending L4S packets. Only the ECT(1) and CE ECN codepoints indicate to a network element that a host is sending L4S packets - specifically that the host claims its behaviour satisfies the pre-requisite transport requirements in Section 4.

5.4.1.2. Exclusion of Traffic From L4S Treatment

To extend the above example, an operator might want to exclude some traffic from the L4S treatment for policy reason, e.g. security (traffic from malicious sources) or commercial (initially the operator may wish to confine the benefits of L4S to business customers).

In this exclusion case, the operator MUST classify on the relevant locally-used identifiers (e.g. source addresses) before classifying the non-matching traffic on the end-to-end L4S ECN identifier.

The operator MUST NOT re-mark the end-to-end L4S identifier, because its decision to exclude certain traffic from L4S treatment is local-only. The end-to-end L4S identifier then survives for other operators to use, or indeed, they can apply their own policy, independently based on their own choice of locally-used identifiers. This approach also allows any operator to remove its locally-applied exclusions in future, e.g. if it wishes to widen the benefit of the L4S treatment to all its customers.

5.4.2. Generalized Combination of L4S and Other Identifiers

L4S concerns low latency, which it can provide for all traffic without differentiation and without affecting bandwidth allocation. Diffserv provides for differentiation of both bandwidth and low latency, but its control of latency depends on its control of bandwidth. The two can be combined if a network operator wants to control bandwidth allocation but it also wants to provide low latency for any amount of traffic within one of these allocations of bandwidth (rather than only providing low latency by limiting bandwidth) [I-D.briscoe-tsvwg-l4s-diffserv].

The examples above were framed in the context of splitting the default Best Efforts Per-Hop Behaviour (PHB) into a Low Latency (L) queue and a Classic (C) Queue. But, more generally, an operator might choose to control bandwidth allocation through a hierarchy of Diffserv PHBs at a node, and to split one or more of these PHBs into a low latency and a classic variant of that PHB.

In the first case, where there are no other PHBs except the DualQ, if a packet carries ECT(1) or CE, a network element would classify it for the L4S treatment irrespective of its DSCP. And, if a packet carried (say) the EF DSCP, the network element could classify it for into L queue irrespective of its ECN codepoint. However, where the DualQ is in a hierarchy of other PHBs, the classifier would classify some traffic into other PHBs based on DSCP before classifying between the latency and classic queues (based on ECT(1), CE and perhaps the EF DSCP or other identifiers as in the above example).

[I-D.briscoe-tsvwg-l4s-diffserv] describes how an operator might use L4S to offer low latency for all L4S traffic as well as using Diffserv for bandwidth differentiation. It identifies two main types of approach, which can be combined: the operator might split certain Diffserv PHBs between L4S and a corresponding Classic service. Or it might split the L4S and/or the Classic service into multiple Diffserv PHBs. In any of these cases, a packet would have to be classified on its Diffserv and ECN codepoints.

In summary, there are numerous ways in which the L4S ECN identifier (ECT(1) and CE) could be combined with other identifiers to achieve particular objectives. The following categorization articulates those that are valid, but it is not necessarily exhaustive. Those tagged 'Global-use' could be set by the sending host or a network. Those tagged 'Local-use' would only be set by a network:

1. Identifiers Complementing the L4S Identifier

    A. Including More Traffic in the L Queue

(Global-use or Local-use)

B. Excluding Certain Traffic from the L Queue (Local-use only)

2. Identifiers to place L4S classification in a PHB Hierarchy (Global-use or Local-use)

A. PHBs Before L4S ECN Classification

B. PHBs After L4S ECN Classification

6. L4S Experiments

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

7. IANA Considerations

This specification contains no IANA considerations.

8. Security Considerations

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

9. Acknowledgements

Thanks to Richard Scheffenegger, John Leslie, David Taeht, Jonathan Morton, Gorry Fairhurst, Michael Welzl, Mikael Abrahamsson and Andrew McGregor for the discussions that led to this specification. Ing-jyh (Inton) Tsang was a contributor to the early drafts of this document. Appendix A listing the TCP Prague Requirements is based on text authored by Marcelo Bagnulo Braun that was originally an appendix to [I-D.ietf-tsvwg-l4s-arch]. That text was in turn based on the collective output of the attendees listed in the minutes of a 'bar BoF' on DCTCP Evolution during IETF-94 [TCPPrague].

The authors' contributions were part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). Bob Briscoe was also

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

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>.

[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>.

10.2. Informative References

[Alizadeh-stability] Alizadeh, M., Javanmard, A., and B. Prabhakar, "Analysis of DCTCP: Stability, Convergence, and Fairness", ACM SIGMETRICS 2011 , June 2011.

[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>.

[DCttH15] De Schepper, K., Bondarenko, O., Briscoe, B., and I. Tsang, "'Data Centre to the Home': Ultra-Low Latency for All", RITE Project Technical Report , 2015, <http://riteproject.eu/publications/>.

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

[I-D.ietf-tcpm-accurate-ecn]
Briscoe, B., Kuehlewind, M., and R. Scheffenegger, "More Accurate ECN Feedback in TCP", draft-ietf-tcpm-accurate-ecn-07 (work in progress), July 2018.

[I-D.ietf-tcpm-generalized-ecn]
Bagnulo, M. and B. Briscoe, "ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control Packets", draft-ietf-tcpm-generalized-ecn-03 (work in progress), October 2018.

[I-D.ietf-tcpm-rack]
Cheng, Y., Cardwell, N., Dukkipati, N., and P. Jha, "RACK: a time-based fast loss detection algorithm for TCP", draft-ietf-tcpm-rack-04 (work in progress), July 2018.

[I-D.ietf-tsvwg-aqm-dualq-coupled]
Schepper, K., Briscoe, B., Bondarenko, O., and I. Tsang, "DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)", draft-ietf-tsvwg-aqm-dualq-coupled-08 (work in progress), November 2018.

[I-D.ietf-tsvwg-ecn-encap-guidelines]
Briscoe, B., Kaippallimalil, J., and P. Thaler, "Guidelines for Adding Congestion Notification to Protocols that Encapsulate IP", draft-ietf-tsvwg-ecn-encap-guidelines-10 (work in progress), March 2018.

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

[I-D.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.

[I-D.stewart-tsvwg-sctpecn]
Stewart, R., Tuexen, M., and X. Dong, "ECN for Stream Control Transmission Protocol (SCTP)", draft-stewart-tsvwg-sctpecn-05 (work in progress), January 2014.

[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>.

[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>.

[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>.

[RFC2983] Black, D., "Differentiated Services and Tunnels", RFC 2983, DOI 10.17487/RFC2983, October 2000, <https://www.rfc-editor.org/info/rfc2983>.

[RFC3246] Davie, B., Charny, A., Bennet, J., Benson, K., Le Boudec, J., Courtney, W., Davari, S., Firoiu, V., and D. Stiliadis, "An Expedited Forwarding PHB (Per-Hop Behavior)", RFC 3246, DOI 10.17487/RFC3246, March 2002, <https://www.rfc-editor.org/info/rfc3246>.

[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>.

[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>.

[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>.

[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>.

[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>.

[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>.

[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>.

[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>.

[TCP-CA] Jacobson, V. and M. Karels, "Congestion Avoidance and Control", Laurence Berkeley Labs Technical Report , November 1988, <http://ee.lbl.gov/papers/congavoid.pdf>.

[TCP-sub-mss-w] 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, <http://www.bobbriscoe.net/projects/latency/sub-mss-w.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. The 'TCP Prague Requirements'

This appendix is informative, not normative. It gives a list of modifications to current scalable transport protocols so that they can be deployed over the public Internet and coexist safely with existing traffic. The list complements the normative requirements in Section 4 that a sender has to comply with before it can set the L4S identifier in packets it sends into the Internet. As well as necessary safety improvements (requirements) this appendix also includes preferable performance improvements (optimizations).

These recommendations have become know as the TCP Prague Requirements, because they were originally identified at an ad hoc meeting during IETF-94 in Prague [TCPPrague]. The wording has been generalized to apply to all scalable congestion controls, not just TCP congestion control specifically.

DCTCP [RFC8257] is currently the most widely used scalable transport protocol. In its current form, DCTCP is specified to be deployable only in controlled environments. Deploying it in the public Internet would lead to a number of issues, both from the safety and the performance perspective. The modifications and additional mechanisms listed in this section will be necessary for its deployment over the global Internet. Where an example is needed, DCTCP is used as a base, but it is likely that most of these requirements equally apply to other scalable transport protocols.

A.1. Requirements for Scalable Transport Protocols

A.1.1. Use of L4S Packet Identifier

Description: A scalable congestion control needs to distinguish the packets it sends from those sent by classic congestion controls.

Motivation: It needs to be possible for a network node to classify L4S packets without flow state into a queue that applies an L4S ECN marking behaviour and isolates L4S packets from the queuing delay of classic packets.

A.1.2. Accurate ECN Feedback

Description: A scalable transport protocol needs to provide timely, accurate feedback about the extent of ECN marking experienced by all packets.

Motivation: Classic congestion controls only need feedback about the existence of a congestion episode within a round trip, not precisely how many packets were marked with ECN or dropped. Therefore, in 2001, when ECN feedback was added to TCP [RFC3168], it could not inform the sender of more than one ECN mark per RTT. Since then, requirements for more accurate ECN feedback in TCP have been defined in [RFC7560] and [I-D.ietf-tcpm-accurate-ecn] specifies an experimental change to the TCP wire protocol to satisfy these requirements. Most other transport protocols already satisfy this requirement.

A.1.3. Fall back to Reno-friendly congestion control on packet loss

Description: A scalable congestion control needs to react to packet loss in a way that will coexist safely with a TCP Reno congestion control [RFC5681].

Motivation: Part of the safety conditions for deploying a scalable congestion control on the public Internet is to make sure that it behaves properly when it builds a queue at a network bottleneck that

has not been upgraded to support L4S. Packet loss can have many causes, but it usually has to be conservatively assumed that it is a sign of congestion. Therefore, on detecting packet loss, a scalable congestion control will need to fall back to classic congestion control behaviour. If it does not comply with this requirement it could starve classic traffic.

A scalable congestion control can be used for different types of transport, e.g. for real-time media or for reliable bulk transport like TCP. Therefore, the particular classic congestion control behaviour to fall back on will need to be part of the congestion control specification of the relevant transport. In the particular case of DCTCP, the current DCTCP specification states that "It is RECOMMENDED that an implementation deal with loss episodes in the same way as conventional TCP." For safe deployment of a scalable transport in the public Internet, the above requirement would need to be defined as a "MUST".

Packet loss might (rarely) occur in the case that the bottleneck is L4S capable. In this case, the sender may receive a high number of packets marked with the CE bit set and also experience a loss. Current DCTCP implementations react differently to this situation. At least one implementation reacts only to the drop signal (e.g. by halving the CWND) and at least another DCTCP implementation reacts to both signals (e.g. by halving the CWND due to the drop and also further reducing the CWND based on the proportion of marked packet). We believe that further experimentation is needed to understand what is the best behaviour for the public Internet, which may or not be one of these existing approaches.

A.1.4. Fall back to Reno-friendly congestion control on classic ECN bottlenecks

Description: A scalable congestion control needs to react to ECN marking from a non-L4S but ECN-capable bottleneck in a way that will coexist with a TCP Reno congestion control [RFC5681].

Motivation: Similarly to the requirement in Appendix A.1.3, this requirement is a safety condition to ensure a scalable congestion control behaves properly when it builds a queue at a network bottleneck that has not been upgraded to support L4S. On detecting classic ECN marking (see below), a scalable congestion control will need to fall back to classic congestion control behaviour. If it does not comply with this requirement it could starve classic traffic.

It would take time for endpoints to distinguish classic and L4S ECN marking. An increase in queuing delay or in delay variation would be

a tell-tale sign, but it is not yet clear where a line would be drawn between the two behaviours. It might be possible to cache what was learned about the path to help subsequent attempts to detect the type of marking.

A.1.5. Reduce RTT dependence

Description: A scalable congestion control needs to reduce or eliminate RTT bias over as wide a range of RTTs as possible, or at least over the typical range of RTTs that will interact in the intended deployment scenario.

Motivation: Classic TCP's throughput is known to be inversely proportional to RTT, so one would expect flows over very low RTT paths to nearly starve flows over larger RTTs. However, Classic TCP has never allowed a very low RTT path to exist because it induces a large queue. For instance, consider two paths with base RTT 1ms and 100ms. If Classic TCP induces a 100ms queue, it turns these RTTs into 101ms and 200ms leading to a throughput ratio of about 2:1. Whereas if a Scalable TCP induces only a 1ms queue, the ratio is 2:101, leading to a throughput ratio of about 50:1.

Therefore, with very small queues, long RTT flows will essentially starve, unless scalable congestion controls comply with this requirement.

A.1.6. Scaling down to fractional congestion windows

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

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

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

more than two such parallel TCP flows will become unresponsive and increase queuing delay.

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

A.1.7. Measuring Reordering Tolerance in Time Units

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

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

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

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

For receivers that need their packets in order, it would seem that relaxing network ordering would simply shift this reordering delay from the network to the receiver. However, that is not true in the general case because links generally do not recognize transport layer flows and often cannot even see application layer streams within the flows (as in SCTP, HTTP/2 or QUIC). So a link will often be holding back packets from one flow or stream while waiting for those from another. Relaxing strict ordering in the network will remove this head-of-line blocking delay. {ToDo: this is being quantified experimentally - will need to add the figures here.}

Classic TCP implementations are switching over to the time-based approach of RACK (Recent ACKnowledgements [I-D.ietf-tcpm-rack]). However, it will be many years (decades?) before networks no longer have to allow for the presence of traditional TCP senders still using the 3 DupACK rule. This specification (Section 4.3) says that senders are not entitled to identify packets as L4S in the IP/ECN field unless they use the time-based approach. Then networks that identify L4S traffic separately (e.g. using [I-D.ietf-tsvwg-aqm-dualq-coupled]) can know for certain that all L4S traffic is using the scalable time-based approach.

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

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

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

On the other hand, it is also important not to wait too long before deeming that a gap is due to a loss (termed a long reordering window), otherwise loss recovery would be slow.

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

A.2. Scalable Transport Protocol Optimizations

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

Description: To improve performance, scalable transport protocols ought to enable ECN at the IP layer in TCP control packets (SYN, SYN-ACK, pure ACKs, etc.) and in retransmitted packets. The same is true for derivatives of TCP, e.g. SCTP.

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

A.2.2. Faster than Additive Increase

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

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

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

A.2.3. Faster Convergence at Flow Start

Description: Particularly when a flow starts, scalable congestion controls need to converge (reach their steady-state share of the capacity) at least as fast as classic TCP and preferably faster. This does not just affect TCP Prague, but also the flow start

behaviour of any L4S congestion control derived from a Classic transport that uses TCP slow start.

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

Appendix B. Alternative Identifiers

This appendix is informative, not normative. It records the pros and cons of various alternative ways to identify L4S packets to record the rationale for the choice of ECT(1) (Appendix B.1) as the L4S identifier. At the end, Appendix B.6 summarises the distinguishing features of the leading alternatives. It is intended to supplement, not replace the detailed text.

The leading solutions all use the ECN field, sometimes in combination with the Diffserv field. Both the ECN and Diffserv fields have the additional advantage that they are no different in either IPv4 or IPv6. A couple of alternatives that use other fields are mentioned at the end, but it is quickly explained why they are not serious contenders.

B.1. ECT(1) and CE codepoints

Definition:

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

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

* Given shortage of codepoints, both L4S and classic ECN sides of an AQM would have to use the same CE codepoint to indicate that a packet had experienced congestion. If a packet that had

already been marked CE in an upstream buffer arrived at a subsequent AQM, this AQM would then have to guess whether to classify CE packets as L4S or classic ECN. Choosing the L4S treatment would be a safer choice, because then a few classic packets might arrive early, rather than a few L4S packets arriving late;

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

Cons:

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

ECN hard in some lower layers: It is not always possible to support ECN in an AQM acting in a buffer below the IP layer [I-D.ietf-tsvwg-ecn-encap-guidelines]. In such cases, the L4S service would have to drop rather than mark frames even though they might contain an ECN-capable packet. However, such cases would be unusual.

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

Non-L4S service for control packets: The classic ECN RFCs [RFC3168] and [RFC5562] require a sender to clear the ECN field to Not-ECT

for retransmissions and certain control packets specifically pure ACKs, window probes and SYNs. When L4S packets are classified by the ECN field alone, these control packets would not be classified into an L4S queue, and could therefore be delayed relative to the other packets in the flow. This would not cause re-ordering (because retransmissions are already out of order, and the control packets carry no data). However, it would make critical control packets more vulnerable to loss and delay. To address this problem, [I-D.ietf-tcpm-generalized-ecn] proposes an experiment in which all TCP control packets and retransmissions are ECN-capable.

Pros:

Should work e2e: The ECN field generally works end-to-end across the Internet. Unlike the DSCP, the setting of the ECN field is at least forwarded unchanged by networks that do not support ECN, and networks rarely clear it to zero;

Should work in tunnels: Unlike Diffserv, ECN is defined to always work across tunnels. However, tunnels do not always implement ECN processing as they should do, particularly because IPsec tunnels were defined differently for a few years.

Could migrate to one codepoint: If all classic ECN senders eventually evolve to use the L4S service, the ECT(0) codepoint could be reused for some future purpose, but only once use of ECT(0) packets had reduced to zero, or near-zero, which might never happen.

B.2. ECN Plus a Diffserv Codepoint (DSCP)

Definition:

For packets with a defined DSCP, all codepoints of the ECN field (except Not-ECT) would signify alternative L4S semantics to those for classic ECN [RFC3168], specifically:

* The L4S DSCP would signifiy that the packet came from an L4S-capable sender;

* ECT(0) and ECT(1) would both signify that the packet was travelling between transport endpoints that were both ECN-capable;

* CE would signify that the packet had been marked by an AQM implementing the L4S service.

Use of a DSCP is the only approach for alternative ECN semantics given as an example in [RFC4774]. However, it was perhaps considered more for controlled environments than new end-to-end services;

Cons:

Consumes DSCP pairs: A DSCP is obviously not orthogonal to Diffserv. Therefore, wherever the L4S service is applied to multiple Diffserv scheduling behaviours, it would be necessary to replace each DSCP with a pair of DSCPs.

Uses critical lower-layer header space: The resulting increased number of DSCPs might be hard to support for some lower layer technologies, e.g. 802.1p and MPLS both offer only 3-bits for a maximum of 8 traffic class identifiers. Although L4S should reduce and possibly remove the need for some DSCPs intended for differentiated queuing delay, it will not remove the need for Diffserv entirely, because Diffserv is also used to allocate bandwidth, e.g. by prioritising some classes of traffic over others when traffic exceeds available capacity.

Not end-to-end (host-network): Very few networks honour a DSCP set by a host. Typically a network will zero (bleach) the Diffserv field from all hosts. Sometimes networks will attempt to identify applications by some form of packet inspection and, based on network policy, they will set the DSCP considered appropriate for the identified application. Network-based application identification might use some combination of protocol ID, port numbers(s), application layer protocol headers, IP address(es), VLAN ID(s) and even packet timing.

Not end-to-end (network-network): Very few networks honour a DSCP received from a neighbouring network. Typically a network will zero (bleach) the Diffserv field from all neighbouring networks at an interconnection point. Sometimes bilateral arrangements are made between networks, such that the receiving network remarks some DSCPs to those it uses for roughly equivalent services. The likelihood that a DSCP will be bleached or ignored depends on the type of DSCP:

Local-use DSCP: These tend to be used to implement application-specific network policies, but a bilateral arrangement to remark certain DSCPs is often applied to DSCPs in the local-use range simply because it is easier not to change all of a network's internal configurations when a new arrangement is made with a neighbour;

Global-use DSCP: These do not tend to be honoured across network interconnections more than local-use DSCPs. However, if two networks decide to honour certain of each other's DSCPs, the reconfiguration is a little easier if both of their globally recognised services are already represented by the relevant global-use DSCPs.

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

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

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

Pros:

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

B.3. ECN capability alone

Definition:

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

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

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

This approach would only be feasible if

A. it was generally agreed that there was little chance of any classic [RFC3168] ECN deployment in any network nodes;

B. it was generally agreed that there was little chance of any client devices being deployed with classic [RFC3168] TCP-ECN on by default (note that classic TCP-ECN is already on-by-default on many servers);

C. for TCP connections, developers of client OSs would all have to agree not to encourage further deployment of classic ECN. Specifically, at the start of a TCP connection classic ECN could be disabled during negotation of the ECN capability:

+ an L4S-capable host would have to disable ECN if the corresponding host did not support accurate ECN feedback [RFC7560], which is a prerequisite for the L4S service;

+ developers of operating systems for user devices would only enable ECN by default for TCP once the stack implemented L4S and accurate ECN feedback [RFC7560] including requesting accurate ECN feedback by default.

Cons:

Near-infeasible deployment constraints: The constraints for deployment above represent a highly unlikely, but not completely impossible, set of circumstances. If, despite the above measures, a pair of hosts did negotiate to use classic ECN, their packets would be classified into the same queue as L4S traffic, and if they had to compete with a long-running L4S flow they would get a very small capacity share;

ECN hard in some lower layers: See the same issue with "ECT(1) and CE codepoints" (Appendix B.1);

Non-L4S service for control packets: See the same issue with "ECT(1) and CE codepoints" (Appendix B.1).

Pros:

Consumes no additional codepoints: The ECT(1) codepoint and all spare Diffserv codepoints would remain available for future use;

Should work e2e: As with "ECT(1) and CE codepoints" (Appendix B.1);

Should work in tunnels: As with "ECT(1) and CE codepoints" (Appendix B.1).

B.4. Protocol ID

It has been suggested that a new ID in the IPv4 Protocol field or the IPv6 Next Header field could identify L4S packets. However this approach is ruled out by numerous problems:

- A new protocol ID would need to be paired with the old one for each transport (TCP, SCTP, UDP, etc.);
- In IPv6, there can be a sequence of Next Header fields, and it would not be obvious which one would be expected to identify a network service like L4S;
- A new protocol ID would rarely provide an end-to-end service, because It is well-known that new protocol IDs are often blocked by numerous types of middlebox;
- The approach is not a solution for AQMs below the IP layer;

B.5. Source or destination addressing

Locally, a network operator could arrange for L4S service to be applied based on source or destination addressing, e.g. packets from its own data centre and/or CDN hosts, packets to its business customers, etc. It could use addressing at any layer, e.g. IP addresses, MAC addresses, VLAN IDs, etc. Although addressing might be a useful tactical approach for a single ISP, it would not be a feasible approach to identify an end-to-end service like L4S. Even for a single ISP, it would require packet classifiers in buffers to be dependent on changing topology and address allocation decisions elsewhere in the network. Therefore this approach is not a feasible solution.

B.6. Summary: Merits of Alternative Identifiers

Table 1 provides a very high level summary of the pros and cons detailed against the schemes described respectively in Appendix B.2, Appendix B.3 and Appendix B.1, for six issues that set them apart.

Issue	DSCP + ECN		ECN	ECT(1) + CE	
	initial	eventual	initial	initial	eventual
end-to-end	N . .	. ? .	. . Y	. . Y	. . Y
tunnels	. O .	. O .	. . ?	. . ?	. . Y
lower layers	N . .	. ? .	. O .	. O .	. . ?
codepoints	N . .	. . ?	. . Y	N . .	. . ?
reordering	. . Y	. . Y	. . Y	. O .	. . ?
ctrl pkts	. . Y	. . Y	. O .	. O .	. . ?
			Note 1		

Note 1: Only feasible if classic ECN is obsolete.

Table 1: Comparison of the Merits of Three Alternative Identifiers

The schemes are scored based on both their capabilities now ('initial') and in the long term ('eventual'). The 'ECN' scheme shares the 'eventual' scores of the 'ECT(1) + CE' scheme. The scores are one of 'N, O, Y', meaning 'Poor', 'Ordinary', 'Good' respectively. The same scores are aligned vertically to aid the eye. A score of "?" in one of the positions means that this approach might optimisitically become this good, given sufficient effort. The table summarises the text and is not meant to be understandable without having read the text.

Appendix C. Potential Competing Uses for the ECT(1) Codepoint

The ECT(1) codepoint of the ECN field has already been assigned once for the ECN nonce [RFC3540], which has now been categorized as historic [RFC8311]. ECN is probably the only remaining field in the Internet Protocol that is common to IPv4 and IPv6 and still has potential to work end-to-end, with tunnels and with lower layers. Therefore, ECT(1) should not be reassigned to a different experimental use (L4S) without carefully assessing competing potential uses. These fall into the following categories:

C.1. Integrity of Congestion Feedback

Receiving hosts can fool a sender into downloading faster by suppressing feedback of ECN marks (or of losses if retransmissions are not necessary or available otherwise).

The historic ECN nonce protocol [RFC3540] proposed that a TCP sender could set either of ECT(0) or ECT(1) in each packet of a flow and

remember the sequence it had set. If any packet was lost or congestion marked, the receiver would miss that bit of the sequence. An ECN Nonce receiver had to feed back the least significant bit of the sum, so it could not suppress feedback of a loss or mark without a 50-50 chance of guessing the sum incorrectly.

The ECN Nonce RFC [RFC3540] as been reclassified as historic, partly because other ways have been developed to protect TCP feedback integrity [RFC8311] that do not consume a codepoint in the IP header. So it is highly unlikely that ECT(1) will be needed for integrity protection in future.

C.2. Notification of Less Severe Congestion than CE

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

Before assigning ECT(1) as an identifer for L4S, we must carefully consider whether it might be better to hold ECT(1) in reserve for future standardisation of rapid flow acceleration, which is an important and enduring problem [RFC6077].

Pre-Congestion Notification (PCN) is another scheme that assigns alternative semantics to the ECN field. It uses ECT(1) to signify a less severe level of pre-congestion notification than CE [RFC6660]. However, the ECN field only takes on the PCN semantics if packets carry a Diffserv codepoint defined to indicate PCN marking within a controlled environment. PCN is required to be applied solely to the outer header of a tunnel across the controlled region in order not to interfere with any end-to-end use of the ECN field. Therefore a PCN region on the path would not interfere with any of the L4S service identifiers proposed in Appendix B.

Authors' Addresses

Koen De Schepper
Nokia Bell Labs
Antwerp
Belgium

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

Bob Briscoe (editor)
CableLabs
UK

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

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.

Table of Contents

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:

- this FECFRAME update does not prevent nor compromise in any way the support of block FEC codes. Both types of codes can nicely co-exist, just like different block FEC schemes can co-exist;

- each sliding window FEC Scheme is associated to a specific FEC Encoding ID subject to IANA registration, just like block FEC Schemes;

- any receiver, for instance a legacy receiver that only supports block FEC schemes, can easily identify the FEC Scheme used in a FECFRAME session. Indeed, the FEC Encoding ID that identifies the FEC Scheme is carried in the FEC Framework Configuration Information (see section 5.5 of [RFC6363]). For instance, when the Session Description Protocol (SDP) is used to carry the FEC Framework Configuration Information, the FEC Encoding ID can be communicated in the "encoding-id=" parameter of a "fec-repair-flow" attribute [RFC6364]. This mechanism is the basic approach for a FECFRAME receiver to determine whether or not it supports the FEC Scheme used in a given FECFRAME session;

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

2. Definitions and Abbreviations

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

Application Data Unit (ADU): The unit of source data provided as payload to the transport layer. For instance, it can be a

payload containing the result of the RTP packetization of a compressed video frame.

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

AL-FEC: Application-layer Forward Error Correction.

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

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

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

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

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

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

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

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

FEC Repair Packet: At a sender (respectively, at a receiver), a payload submitted to (respectively, received from) the transport

protocol containing one or more repair symbols along with a Repair FEC Payload ID and possibly an RTP header.

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

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

Repair Flow: The packet flow carrying FEC data.

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

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

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

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

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

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

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

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

Encoding Symbol: Unit of data generated by the encoding process. With systematic codes, source symbols are part of the encoding symbols.

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

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

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

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

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

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

3. Summary of Architecture Overview

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

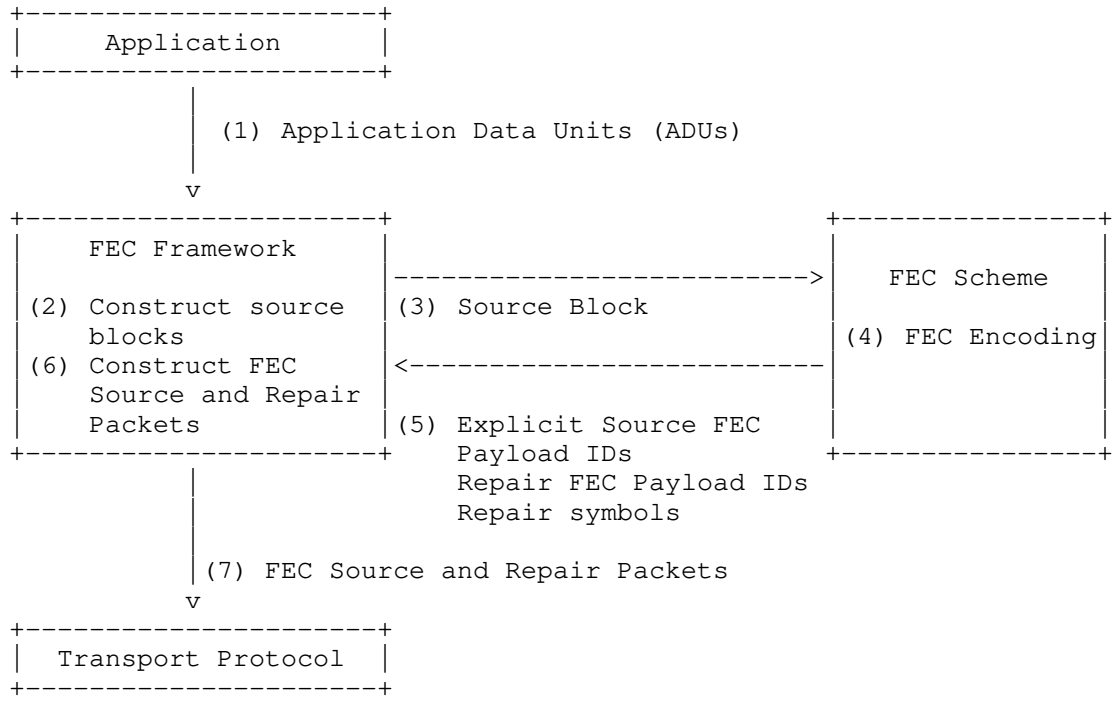


Figure 1: FECFRAME architecture at a sender.

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:

- ADUs-to-source-symbols mapping: in order to manage variable size ADUs, FECFRAME and FEC Schemes can use small, fixed size symbols and create a mapping between ADUs and symbols. To each ADU this mechanism prepends a length field (plus a flow identifier, see below) and pads the result to a multiple of the symbol size. A small ADU may be mapped to a single source symbol while a large one may be mapped to multiple symbols. The mapping details are FEC-Scheme-dependent and must be defined in the associated document;

- Assignment of decoded ADUs to flows in multi-flow configurations: when multiple flows are multiplexed over the same FECFRAME instance, a problem is to assign a decoded ADU to the right flow (UDP port numbers and IP addresses traditionally used to map incoming ADUs to flows are not recovered during FEC decoding). To make it possible, at the FECFRAME sending instance, each ADU is prepended with a flow identifier (1 byte) during the ADU-to-source-symbols mapping (see above). The flow identifiers are also shared between all FECFRAME instances as part of the FEC Framework Configuration Information. This (flow identifier + length + application payload + padding), called ADUI, is then FEC protected. Therefore a decoded ADUI contains enough information to assign the ADU to the right flow.

A few aspects are not covered by FECFRAME, namely:

- [RFC6363] section 8 does not detail any congestion control mechanism, but only provides high level normative requirements;

- 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.

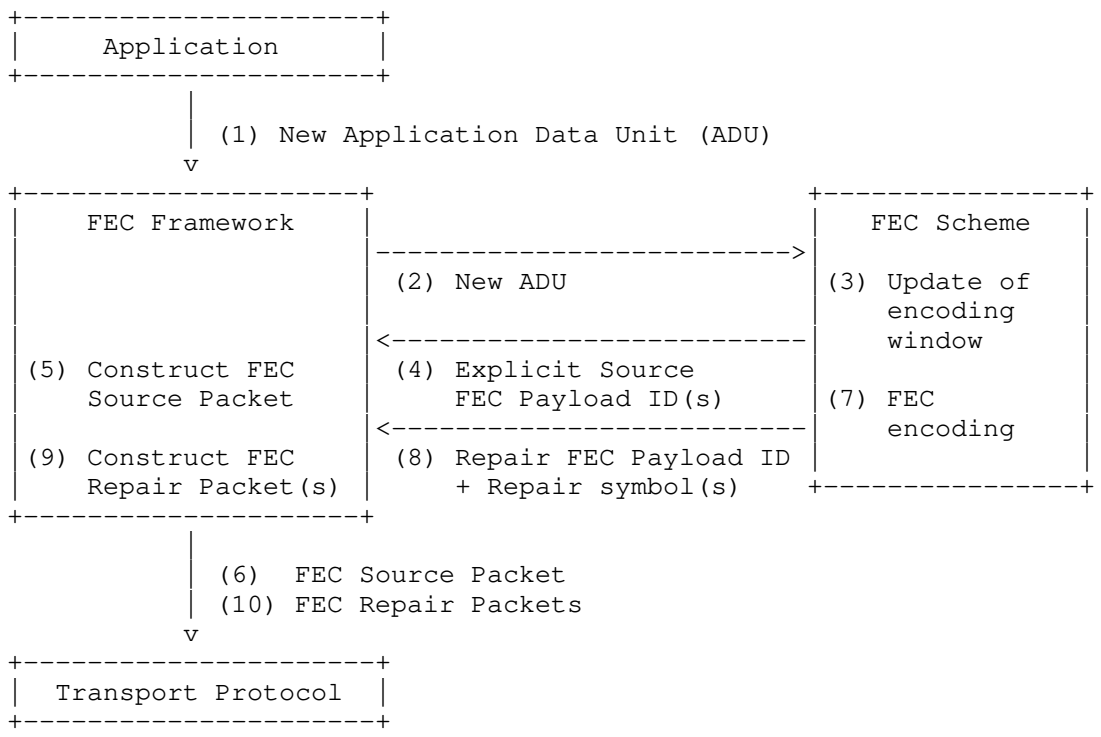


Figure 2: Sender Operation with Sliding Window FEC Codes

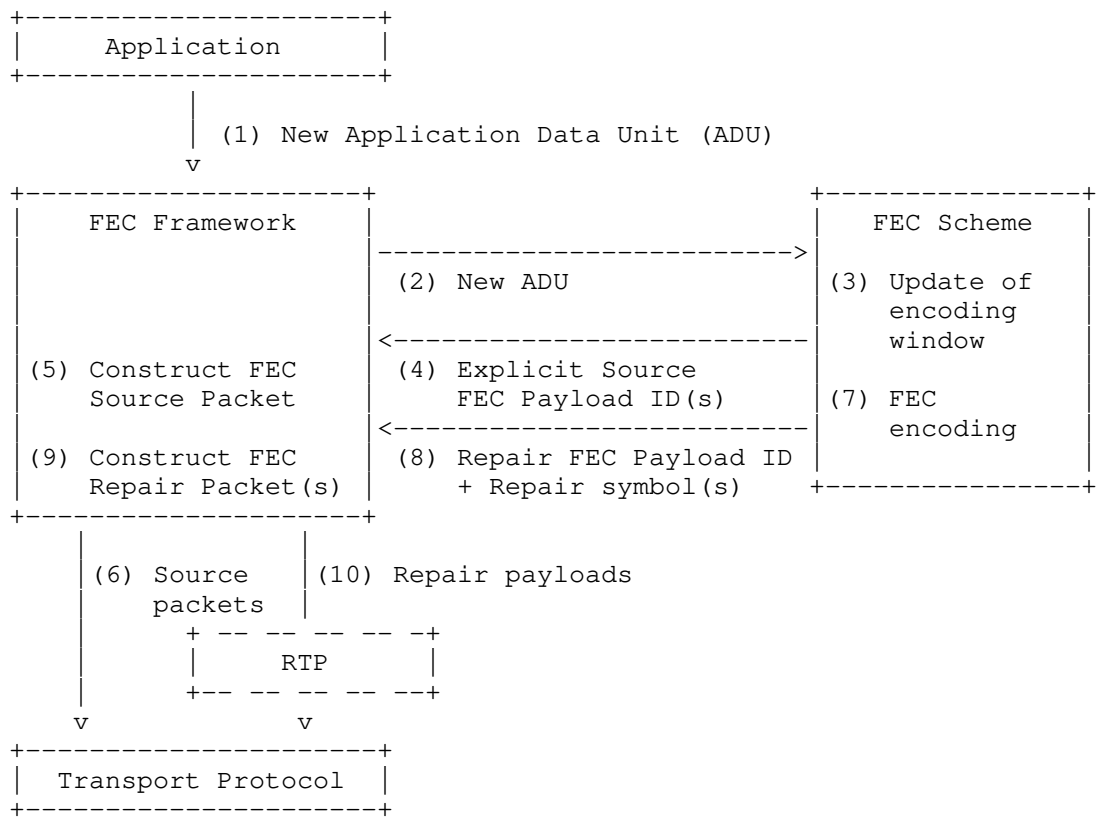


Figure 3: Sender Operation with Sliding Window FEC Codes and RTP Repair Flows

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.

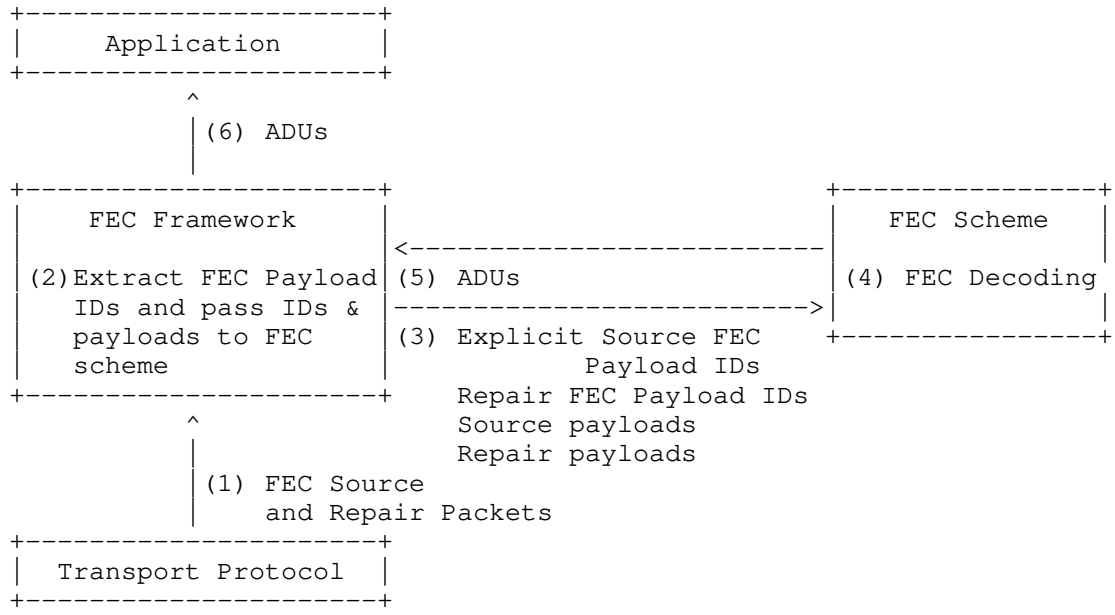


Figure 4: Receiver Operation with Sliding Window FEC Codes

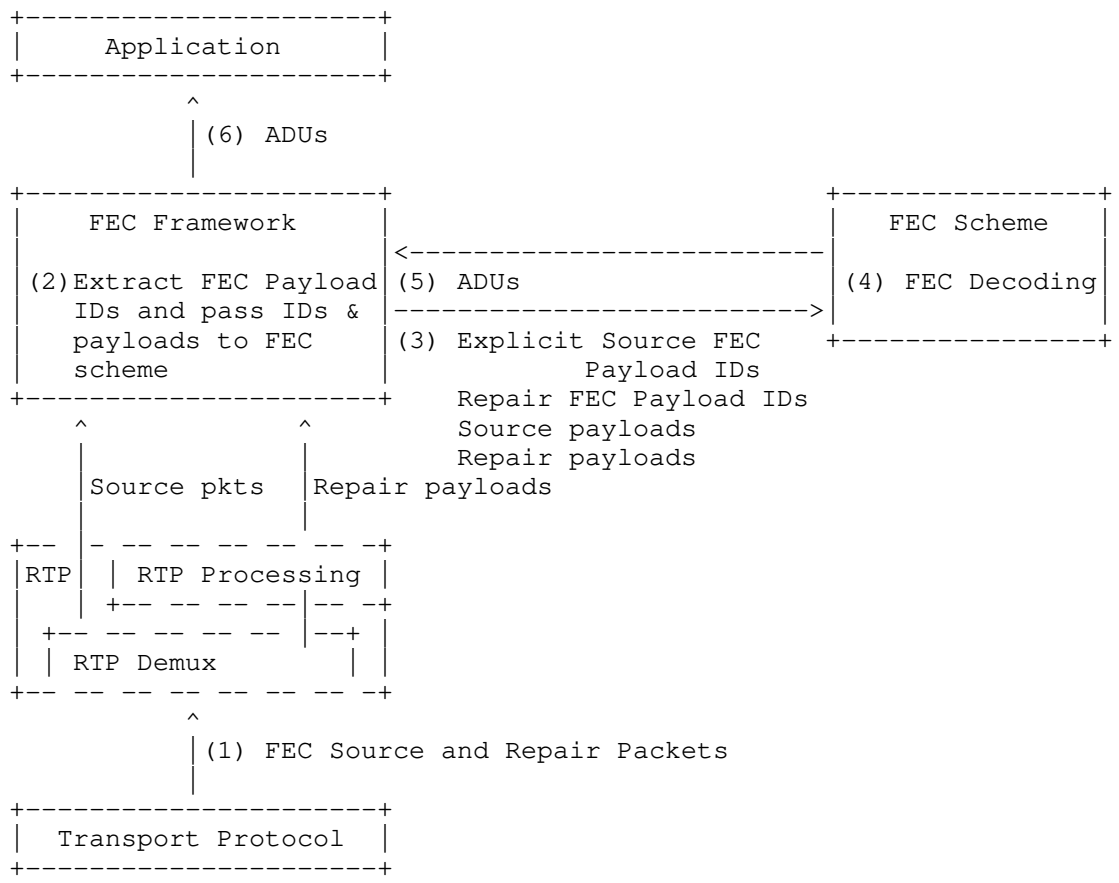


Figure 5: Receiver Operation with Sliding Window FEC Codes and RTP Repair Flows

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:

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

6. Feedback

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

7. Transport Protocols

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

8. Congestion Control

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

9. Implementation Status

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

An implementation of FECFRAME extended to Sliding Window codes exists:

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

10. Security Considerations

This FECFRAME extension does not add any new security consideration. All the considerations of [RFC6363], Section 9, apply to this document as well. However, for the sake of completeness, the

following goal can be added to the list provided in Section 9.1 "Problem Statement" of [RFC6363]:

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

11. Operations and Management Considerations

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

12. IANA Considerations

No IANA actions are required for this document.

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

13. Acknowledgments

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

14. References

14.1. Normative References

[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:

- after a certain delay, when an "old" ADU of a real-time flow times out. The source symbol retention delay in the sliding encoding window should therefore be initialized according to the real-time features of incoming flow(s) when applicable;
- once the sliding encoding window has reached its maximum size (there is usually an upper limit to the sliding encoding window size). In that case the oldest symbol is removed each time a new source symbol is added.

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

- at the source flows level: real-time constraints can limit the total time source symbols can remain in the encoding window;
- at the FEC code level: theoretical or practical limitations (e.g., because of computational complexity) can limit the number of source symbols in the encoding window;
- at the FEC Scheme level: signaling and window management are intrinsically related. For instance, an encoding window composed of a non-sequential set of source symbols requires an appropriate signaling to inform a receiver of the composition of the encoding window, and the associated transmission overhead can limit the maximum encoding window size. On the opposite, an encoding window always composed of a sequential set of source symbols simplifies signaling: providing the identity of the first source symbol plus their number is sufficient, which creates a fixed and relatively small transmission overhead.

Authors' Addresses

Vincent Roca
INRIA
Univ. Grenoble Alpes
France

EMail: vincent.roca@inria.fr

Ali Begen
Networked Media
Konya
Turkey

EMail: ali.begen@networked.media

Transport Area Working Group
Internet-Draft
Intended status: Informational
Expires: April 25, 2019

B. Briscoe, Ed.
CableLabs
K. De Schepper
Nokia Bell Labs
M. Bagnulo Braun
Universidad Carlos III de Madrid
October 22, 2018

Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture
draft-ietf-tsvwg-l4s-arch-03

Abstract

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

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

It is increasingly common for _all_ of a user's applications at any one time to require low delay: interactive Web, Web services, voice, conversational video, interactive video, interactive remote presence, instant messaging, online gaming, remote desktop, cloud-based applications and video-assisted remote control of machinery and industrial processes. In the last decade or so, much has been done to reduce propagation delay by placing caches or servers closer to users. However, queuing remains a major, albeit intermittent, component of latency. For instance spikes of hundreds of milliseconds are common. During a long-running flow, even with state-of-the-art active queue management (AQM), the base speed-of-light path delay roughly doubles. Low loss is also important because, for interactive applications, losses translate into even longer retransmission delays.

It has been demonstrated that, once access network bit rates reach levels now common in the developed world, increasing capacity offers diminishing returns if latency (delay) is not addressed. Differentiated services (Diffserv) offers Expedited Forwarding [RFC3246] for some packets at the expense of others, but this is not sufficient when all (or most) of a user's applications require low latency.

Therefore, the goal is an Internet service with ultra-Low queueing Latency, ultra-Low Loss and Scalable throughput (L4S) - for _all_ traffic. A service for all traffic will need none of the configuration or management baggage (traffic policing, traffic contracts) associated with favouring some packets over others. This document describes the L4S architecture for achieving that goal.

It must be said that queuing delay only degrades performance infrequently [Hohlfeld14]. It only occurs when a large enough

capacity-seeking (e.g. TCP) flow is running alongside the user's traffic in the bottleneck link, which is typically in the access network. Or when the low latency application is itself a large capacity-seeking flow (e.g. interactive video). At these times, the performance improvement from L4S must be so remarkable that network operators will be motivated to deploy it.

Active Queue Management (AQM) is part of the solution to queuing under load. AQM improves performance for all traffic, but there is a limit to how much queuing delay can be reduced by solely changing the network; without addressing the root of the problem.

The root of the problem is the presence of standard TCP congestion control (Reno [RFC5681]) or compatible variants (e.g. TCP Cubic [RFC8312]). We shall call this family of congestion controls 'Classic' TCP. It has been demonstrated that if the sending host replaces Classic TCP with a 'Scalable' alternative, when a suitable AQM is deployed in the network the performance under load of all the above interactive applications can be stunningly improved. For instance, queuing delay under heavy load with the example DCTCP/DualQ solution cited below is roughly 1 millisecond (1 ms) at the 99th percentile without losing link utilization. This compares with 5 to 20 ms on _average_ with a Classic TCP and current state-of-the-art AQMs such as fq_CoDel [RFC8290] or PIE [RFC8033]. Also, with a Classic TCP, 5 ms of queuing is usually only possible by losing some utilization.

It has been convincingly demonstrated [DCttH15] that it is possible to deploy such an L4S service alongside the existing best efforts service so that all of a user's applications can shift to it when their stack is updated. Access networks are typically designed with one link as the bottleneck for each site (which might be a home, small enterprise or mobile device), so deployment at a single network node should give nearly all the benefit. The L4S approach also requires component mechanisms at the endpoints to fulfill its goal. This document presents the L4S architecture, by describing the different components and how they interact to provide the scalable low-latency, low-loss, Internet service.

2. L4S Architecture Overview

There are three main components to the L4S architecture (illustrated in Figure 1):

1) Network: The L4S service traffic needs to be isolated from the queuing latency of the Classic service traffic. However, the two should be able to freely share a common pool of capacity. This is because there is no way to predict how many flows at any one time

might use each service and capacity in access networks is too scarce to partition into two. So a 'semi-permeable' membrane is needed that partitions latency but not bandwidth. The Dual Queue Coupled AQM [I-D.ietf-tsvwg-aqm-dualq-coupled] is an example of such a semi-permeable membrane.

Per-flow queuing such as in [RFC8290] could be used, but it partitions both latency and bandwidth between every end-to-end flow. So it is rather overkill, which brings disadvantages (see Section 5.2), not least that thousands of queues are needed when two are sufficient.

2) Protocol: A host needs to distinguish L4S and Classic packets with an identifier so that the network can classify them into their separate treatments. [I-D.ietf-tsvwg-ecn-l4s-id] considers various alternative identifiers, and concludes that all alternatives involve compromises, but the ECT(1) codepoint of the ECN field is a workable solution.

3) Host: Scalable congestion controls already exist. They solve the scaling problem with TCP that was first pointed out in [RFC3649]. The one used most widely (in controlled environments) is Data Centre TCP (DCTCP [RFC8257]), which has been implemented and deployed in Windows Server Editions (since 2012), in Linux and in FreeBSD. Although DCTCP as-is 'works' well over the public Internet, most implementations lack certain safety features that will be necessary once it is used outside controlled environments like data centres (see later). A similar scalable congestion control will also need to be transplanted into protocols other than TCP (SCTP, RTP/RTCP, RMCAT, etc.)

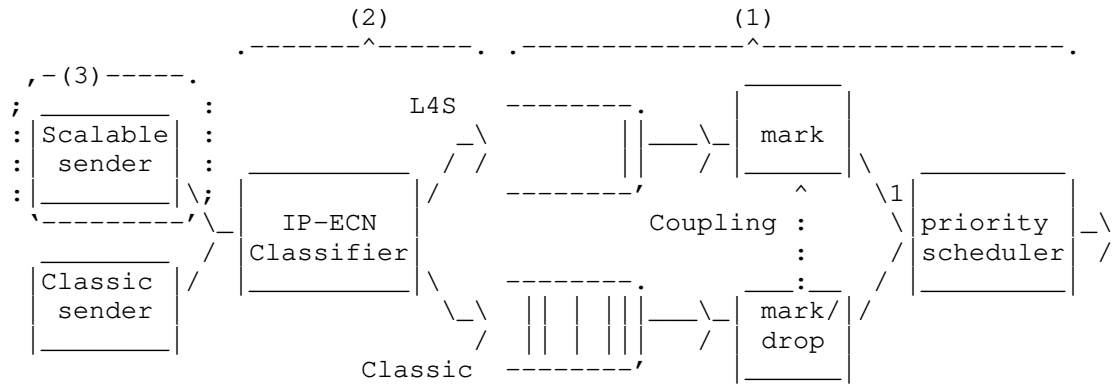


Figure 1: Components of an L4S Solution: 1) Isolation in separate network queues; 2) Packet Identification Protocol; and 3) Scalable Sending Host

3. Terminology

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

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

Low-Latency, Low-Loss and Scalable (L4S) service: The 'L4S' service is intended for traffic from scalable TCP algorithms such as Data Centre TCP. But it is also more general--it will allow a set of congestion controls with similar scaling properties to DCTCP (e.g. Relentless [Mathis09]) to evolve.

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

Scalable Congestion Control: A congestion control where the packet flow rate per round trip (the window) is inversely proportional to the level (probability) of congestion signals. Then, as flow rate scales, the number of congestion signals per round trip remains invariant, maintaining the same degree of control. For instance,

DCTCP averages 2 congestion signals per round-trip whatever the flow rate.

Classic Congestion Control: A congestion control with a flow rate that can co-exist with standard TCP Reno [RFC5681] without starvation. With Classic congestion controls, as capacity increases enabling higher flow rates, the number of round trips between congestion signals (losses or ECN marks) rises in proportion to the flow rate. So control of queuing and/or utilization becomes very slack. For instance, with 1500 B packets and an RTT of 18 ms, as TCP Reno flow rate increases from 2 to 100 Mb/s the number of round trips between congestion signals rises proportionately, from 2 to 100.

The default congestion control in Linux (TCP Cubic) is Reno-compatible for most Internet access scenarios expected for some years. For instance, with a typical domestic round-trip time (RTT) of 18ms, TCP Cubic only switches out of Reno-compatibility mode once the flow rate approaches 1 Gb/s. For a typical data centre RTT of 1 ms, the switch-over point is theoretically 1.3 Tb/s. However, with a less common transcontinental RTT of 100 ms, it only remains Reno-compatible up to 13 Mb/s. All examples assume 1,500 B packets.

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

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

4. L4S Architecture Components

The L4S architecture is composed of the following elements.

Protocols:The L4S architecture encompasses the two protocol changes (an unassignment and an assignment) that we describe next:

a. An essential aspect of a scalable congestion control is the use of explicit congestion signals rather than losses, because the signals need to be sent immediately and frequently--too often to use drops. 'Classic' ECN [RFC3168] requires an ECN signal to be treated the same as a drop, both when it is generated in the network and when it is responded to by hosts. L4S needs networks and hosts to support a different meaning for ECN. So the

standards track [RFC3168] needs to be updated to allow L4S packets to depart from the 'same as drop' constraint.

[RFC8311] is a standards track update to relax specific requirements in RFC 3168 (and certain other standards track RFCs), which clears the way for the experimental changes proposed for L4S. [RFC8311] also reclassifies the original experimental assignment of the ECT(1) codepoint as an ECN nonce [RFC3540] as historic.

b. [I-D.ietf-tsvwg-ecn-l4s-id] recommends ECT(1) is used as the identifier to classify L4S packets into a separate treatment from Classic packets. This satisfies the requirements for identifying an alternative ECN treatment in [RFC4774].

Network components:The Dual Queue Coupled AQM has been specified as generically as possible [I-D.ietf-tsvwg-aqm-dualq-coupled] as a 'semi-permeable' membrane without specifying the particular AQMs to use in the two queues. An informational appendix of the draft is provided for pseudocode examples of different possible AQM approaches. Initially a zero-config variant of RED called Curvy RED was implemented, tested and documented. The aim is for designers to be free to implement diverse ideas. So the brief normative body of the draft only specifies the minimum constraints an AQM needs to comply with to ensure that the L4S and Classic services will coexist. For instance, a variant of PIE called Dual PI Squared [PI2] has been implemented and found to perform better than Curvy RED over a wide range of conditions, so it has been documented in another appendix of [I-D.ietf-tsvwg-aqm-dualq-coupled].

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

a. Data Centre TCP is the most widely used example of a scalable congestion control. It has been documented as an informational record of the protocol currently in use [RFC8257]. It will be necessary to define a number of safety features for a variant usable on the public Internet. A draft list of these, known as the TCP Prague requirements, has been drawn up (see Appendix A of [I-D.ietf-tsvwg-ecn-l4s-id]). The list also includes some optional performance improvements.

b. Transport protocols other than TCP use various congestion controls designed to be friendly with Classic TCP. Before they can use the L4S service, it will be necessary to implement scalable variants of each of these congestion control behaviours. The following standards track RFCs currently define these protocols: ECN in TCP [RFC3168], in SCTP [RFC4960], in RTP

[RFC6679], and in DCCP [RFC4340]. Not all are in widespread use, but those that are will eventually need to be updated to allow a different congestion response, which they will have to indicate by using the ECT(1) codepoint. Scalable variants are under consideration for some new transport protocols that are themselves under development, e.g. QUIC [I-D.ietf-quic-transport] and certain real-time media congestion avoidance techniques (RMCAT) protocols.

c. ECN feedback is sufficient for L4S in some transport protocols (RTCP, DCCP) but not others:

   * For the case of TCP, the feedback protocol for ECN embeds the assumption from Classic ECN that an ECN mark is the same as a drop, making it unusable for a scalable TCP. Therefore, the implementation of TCP receivers will have to be upgraded [RFC7560]. Work to standardize more accurate ECN feedback for TCP (AccECN [I-D.ietf-tcpm-accurate-ecn]) is in progress.

   * ECN feedback is only roughly sketched in an appendix of the SCTP specification. A fuller specification has been proposed [I-D.stewart-tsvwg-sctpecn], which would need to be implemented and deployed before SCTCP could support L4S.

5. Rationale

5.1. Why These Primary Components?

Explicit congestion signalling (protocol): Explicit congestion signalling is a key part of the L4S approach. In contrast, use of drop as a congestion signal creates a tension because drop is both a useful signal (more would reduce delay) and an impairment (less would reduce delay). Explicit congestion signals can be used many times per round trip, to keep tight control, without any impairment. Under heavy load, even more explicit signals can be applied so the queue can be kept short whatever the load. Whereas state-of-the-art AQMs have to introduce very high packet drop at high load to keep the queue short. Further, when using ECN, TCP's sawtooth reduction can be smaller and therefore return to the operating point more often, without worrying that this causes more signals (one at the top of each smaller sawtooth). The consequent smaller amplitude sawteeth fit between a very shallow marking threshold and an empty queue, so delay variation can be very low, without risk of under-utilization.

All the above makes it clear that explicit congestion signalling is only advantageous for latency if it does not have to be considered 'the same as' drop (as required with Classic ECN

[RFC3168]). Therefore, in a DualQ AQM, the L4S queue uses a new L4S variant of ECN that is not equivalent to drop [I-D.ietf-tsvwg-ecn-l4s-id], while the Classic queue uses either classic ECN [RFC3168] or drop, which are equivalent.

Before Classic ECN was standardized, there were various proposals to give an ECN mark a different meaning from drop. However, there was no particular reason to agree on any one of the alternative meanings, so 'the same as drop' was the only compromise that could be reached. RFC 3168 contains a statement that:

> "An environment where all end nodes were ECN-Capable could allow new criteria to be developed for setting the CE codepoint, and new congestion control mechanisms for end-node reaction to CE packets. However, this is a research issue, and as such is not addressed in this document."

Latency isolation with coupled congestion notification (network): Using just two queues is not essential to L4S (more would be possible), but it is the simplest way to isolate all the L4S traffic that keeps latency low from all the legacy Classic traffic that does not.

Similarly, coupling the congestion notification between the queues is not necessarily essential, but it is a clever and simple way to allow senders to determine their rate, packet-by-packet, rather than be overridden by a network scheduler. Because otherwise a network scheduler would have to inspect at least transport layer headers, and it would have to continually assign a rate to each flow without any easy way to understand application intent.

L4S packet identifier (protocol): Once there are at least two separate treatments in the network, hosts need an identifier at the IP layer to distinguish which treatment they intend to use.

Scalable congestion notification (host): A scalable congestion control keeps the signalling frequency high so that rate variations can be small when signalling is stable, and rate can track variations in available capacity as rapidly as possible otherwise.

5.2. Why Not Alternative Approaches?

All the following approaches address some part of the same problem space as L4S. In each case, it is shown that L4S complements them or improves on them, rather than being a mutually exclusive alternative:

Diffserv: Diffserv addresses the problem of bandwidth apportionment for important traffic as well as queuing latency for delay-sensitive traffic. L4S solely addresses the problem of queuing latency (as well as loss and throughput scaling). Diffserv will still be necessary where important traffic requires priority (e.g. for commercial reasons, or for protection of critical infrastructure traffic) - see [I-D.briscoe-tsvwg-l4s-diffserv]. Nonetheless, if there are Diffserv classes for important traffic, the L4S approach can provide low latency for _all_ traffic within each Diffserv class (including the case where there is only one Diffserv class).

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

State-of-the-art AQMs: AQMs such as PIE and fq_CoDel give a significant reduction in queuing delay relative to no AQM at all. The L4S work is intended to complement these AQMs, and we definitely do not want to distract from the need to deploy them as widely as possible. Nonetheless, without addressing the large saw-toothing rate variations of Classic congestion controls, AQMs alone cannot reduce queuing delay too far without significantly reducing link utilization. The L4S approach resolves this tension by ensuring hosts can minimize the size of their sawteeth without appearing so aggressive to legacy flows that they starve them.

Per-flow queuing: Similarly per-flow queuing is not incompatible with the L4S approach. However, one queue for every flow can be thought of as overkill compared to the minimum of two queues for all traffic needed for the L4S approach. The overkill of per-flow queuing has side-effects:

A. fq makes high performance networking equipment costly (processing and memory) - in contrast dual queue code can be very simple;

B. fq requires packet inspection into the end-to-end transport layer, which doesn't sit well alongside encryption for privacy - in contrast the use of ECN as the classifier for L4S requires no deeper inspection than the IP layer;

C. fq isolates the queuing of each flow from the others but not from itself so, unlike L4S, it does not support applications that need both capacity-seeking behaviour and very low latency.

   It might seem that self-inflicted queuing delay should not count, because if the delay wasn't in the network it would just shift to the sender. However, modern adaptive applications, e.g. HTTP/2 [RFC7540] or the interactive media applications described in Section 6, can keep low latency objects at the front of their local send queue by shuffling priorities of other objects dependent on the progress of other transfers. They cannot shuffle packets once they have released them into the network.

D. fq prevents any one flow from consuming more than 1/N of the capacity at any instant, where N is the number of flows. This is fine if all flows are elastic, but it does not sit well with a variable bit rate real-time multimedia flow, which requires wriggle room to sometimes take more and other times less than a 1/N share.

   It might seem that an fq scheduler offers the benefit that it prevents individual flows from hogging all the bandwidth. However, L4S has been deliberately designed so that policing of individual flows can be added as a policy choice, rather than requiring one specific policy choice as the mechanism itself. A scheduler (like fq) has to decide packet-by-packet which flow to schedule without knowing application intent. Whereas a separate policing function can be configured less strictly, so that senders can still control the instantaneous rate of each flow dependent on the needs of each application (e.g. variable rate video), giving more wriggle-room before a flow is deemed non-compliant. Also policing of queuing and of flow-rates can be applied independently.

Alternative Back-off ECN (ABE): Yet again, L4S is not an alternative to ABE but a complement that introduces much lower queuing delay. ABE [I-D.ietf-tcpm-alternativebackoff-ecn] alters the host behaviour in response to ECN marking to utilize a link better and give ECN flows faster throughput, but it assumes the network still treats ECN and drop the same. Therefore ABE exploits any lower queuing delay that AQMs can provide. But as explained above, AQMs still cannot reduce queuing delay too far without losing link utilization (to allow for other, non-ABE, flows).

6. Applicability

6.1. Applications

A transport layer that solves the current latency issues will provide new service, product and application opportunities.

With the L4S approach, the following existing applications will immediately experience significantly better quality of experience under load in the best effort class:

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

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

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

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

Using a swiping finger gesture or head movement to pan a video are extremely latency-demanding actions--far more demanding than VoIP.

Because human vision can detect extremely low delays of the order of single milliseconds when delay is translated into a visual lag between a video and a reference point (the finger or the orientation of the head sensed by the balance system in the inner ear (the vestibular system).

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

Other low latency high bandwidth applications such as:

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

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

6.2. Use Cases

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

- Where the bottleneck is one of various types of access network: DSL, cable, mobile, satellite
  - Radio links (cellular, WiFi, satellite) that are distant from the source are particularly challenging. The radio link capacity can vary rapidly by orders of magnitude, so it is often desirable to hold a buffer to utilise sudden increases of capacity;
  - cellular networks are further complicated by a perceived need to buffer in order to make hand-overs imperceptible;
  - Satellite networks generally have a very large base RTT, so even with minimal queuing, overall delay can never be extremely low;
  - Nonetheless, it is certainly desirable not to hold a buffer purely because of the sawteeth of Classic TCP, when it is more than is needed for all the above reasons.

- Private networks of heterogeneous data centres, where there is no single administrator that can arrange for all the simultaneous changes to senders, receivers and network needed to deploy DCTCP:
  - a set of private data centres interconnected over a wide area with separate administrations, but within the same company
  - a set of data centres operated by separate companies interconnected by a community of interest network (e.g. for the finance sector)
  - multi-tenant (cloud) data centres where tenants choose their operating system stack (Infrastructure as a Service - IaaS)
- Different types of transport (or application) congestion control:
  - elastic (TCP/SCTP);
  - real-time (RTP, RMCAT);
  - query (DNS/LDAP).
- Where low delay quality of service is required, but without inspecting or intervening above the IP layer [I-D.smith-encrypted-traffic-management]:
  - mobile and other networks have tended to inspect higher layers in order to guess application QoS requirements. However, with growing demand for support of privacy and encryption, L4S offers an alternative. There is no need to select which traffic to favour for queuing, when L4S gives favourable queuing to all traffic.
- If queuing delay is minimized, applications with a fixed delay budget can communicate over longer distances, or via a longer chain of service functions [RFC7665] or onion routers.

6.3. Deployment Considerations

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

L4S involves both end systems and the network, so Section 6.3.2 suggests some typical sequences to deploy each part, and why there

will be an immediate and significant benefit after deploying just one part.

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

6.3.1. Deployment Topology

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

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

Therefore, the full benefit of the L4S service should be available in the downstream direction when the DualQ AQM is deployed at the ingress to this bottleneck link (or links for multihomed sites). And similarly, the full upstream service will be available once the DualQ is deployed at the upstream ingress.

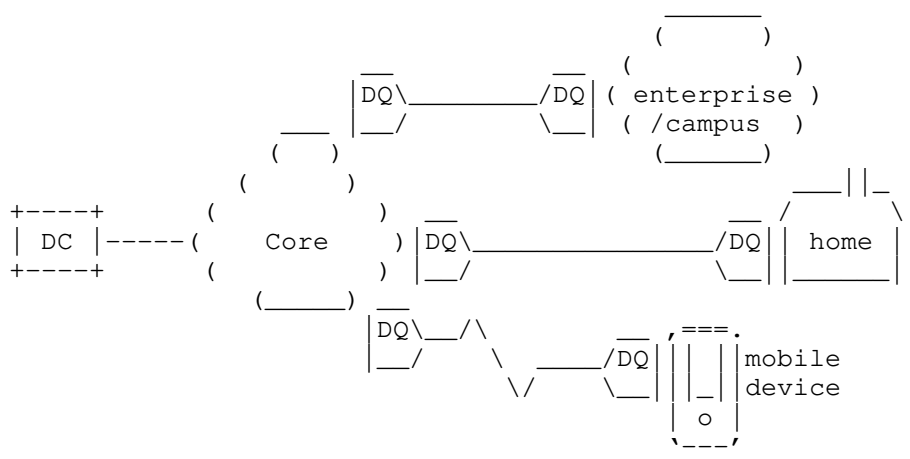


Figure 2: Likely location of DualQ (DQ) Deployments in common access topologies

Deployment in mesh topologies depends on how over-booked the core is. If the core is non-blocking, or at least generously provisioned so that the edges are nearly always the bottlenecks, it would only be necessary to deploy the DualQ AQM at the edge bottlenecks. For example, some data-centre networks are designed with the bottleneck in the hypervisor or host NICs, while others bottleneck at the top-of-rack switch (both the output ports facing hosts and those facing the core).

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

6.3.2. Deployment Sequences

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

Firstly, L4S deployment exploits the fact that DCTCP already exists on many Internet hosts (Windows, FreeBSD and Linux); both servers and clients. Therefore, just deploying DualQ AQM at a network bottleneck immediately gives a working deployment of all the L4S parts. DCTCP needs some safety concerns to be fixed for general use over the public Internet (see Section 2.3 of [I-D.ietf-tsvwg-ecn-l4s-id]), but

DCTCP is not on by default, so these issues can be managed within controlled deployments or controlled trials.

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

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

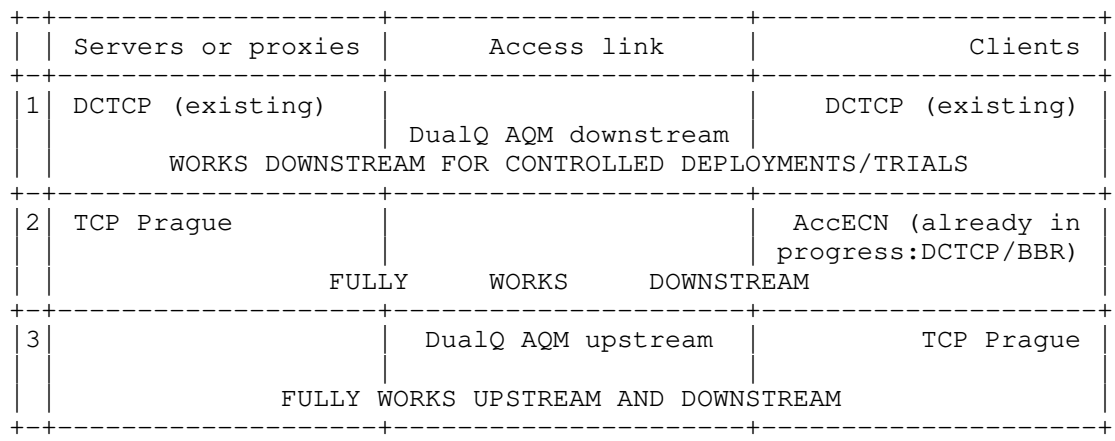


Figure 3: Example L4S Deployment Sequences

Figure 3 illustrates some example sequences in which the parts of L4S might be deployed. It consists of the following stages:

1. Here, the immediate benefit of a single AQM deployment can be seen, but limited to a controlled trial or controlled deployment. In this example downstream deployment is first, but in other scenarios the upstream might be deployed first. If no AQM at all was previously deployed for the downstream access, the DualQ AQM greatly improves the Classic service (as well as adding the L4S service). If an AQM was already deployed, the Classic service will be unchanged (and L4S will still be added).

2. In this stage, the name 'TCP Prague' is used to represent a variant of DCTCP that is safe to use in a production environment. If the application is primarily unidirectional, 'TCP Prague' at one end will provide all the benefit needed. Accurate ECN feedback (AccECN) [I-D.ietf-tcpm-accurate-ecn] is needed at the other end, but it is a generic ECN feedback facility that is already planned to be deployed for other purposes, e.g. DCTCP, BBR [BBR]. The two ends can be deployed in either order, because TCP Prague only enables itself if it has negotiated the use of AccECN feedback with the other end during the connection handshake. Thus, deployment of TCP Prague on a server enables L4S trials to move to a production service in one direction, wherever AccECN is deployed at the other end. This stage might be further motivated by the performance improvements of TCP Prague relative to DCTCP (see Appendix A.2 of [I-D.ietf-tsvwg-ecn-l4s-id]).

3. This is a two-move stage to enable L4S upstream. The DualQ or TCP Prague can be deployed in either order as already explained. To motivate the first of two independent moves, the deferred benefit of enabling new services after the second move has to be worth it to cover the first mover's investment risk. As explained already, the potential for new interactive services provides this motivation. The DualQ AQM also greatly improves the upstream Classic service, assuming no other AQM has already been deployed.

Note that other deployment sequences might occur. For instance: the upstream might be deployed first; a non-TCP protocol might be used end-to-end, e.g. QUIC, RMCAT; a body such as the 3GPP might require L4S to be implemented in 5G user equipment, or other random acts of kindness.

6.3.3. L4S Flow but Non-L4S Bottleneck

If L4S is enabled between two hosts but there is no L4S AQM at the bottleneck, any drop from the bottleneck will trigger the L4S sender to fall back to a classic ('TCP-Friendly') behaviour (see Appendix A.1.3 of [I-D.ietf-tsvwg-ecn-l4s-id]).

Unfortunately, as well as protecting legacy traffic, this rule degrades the L4S service whenever there is a loss, even if the loss was not from a non-DualQ bottleneck (false negative). And unfortunately, prevalent drop can be due to other causes, e.g.:

- congestion loss at other transient bottlenecks, e.g. due to bursts in shallower queues;
- transmission errors, e.g. due to electrical interference;
- rate policing.

Three complementary approaches are in progress to address this issue, but they are all currently research:

- In TCP Prague, ignore certain losses deemed unlikely to be due to congestion (using some ideas from BBR [BBR] but with no need to ignore nearly all losses). This could mask any of the above types of loss (requires consensus on how to safely interoperate with drop-based congestion controls).
- A combination of RACK, reconfigured link retransmission and L4S could address transmission errors [I-D.ietf-tsvwg-ecn-l4s-id];
- Hybrid ECN/drop policers (see Section 8.3).

L4S deployment scenarios that minimize these issues (e.g. over wireline networks) can proceed in parallel to this research, in the expectation that research success will continually widen L4S applicability.

Classic ECN support is starting to materialize (in the upstream of some home routers as of early 2017), so an L4S sender will have to fall back to a classic ('TCP-Friendly') behaviour if it detects that ECN marking is accompanied by greater queuing delay or greater delay variation than would be expected with L4S (see Appendix A.1.4 of [I-D.ietf-tsvwg-ecn-l4s-id]).

6.3.4. Other Potential Deployment Issues

An L4S AQM uses the ECN field to signal congestion. So, in common with Classic ECN, if the AQM is within a tunnel or at a lower layer, correct functioning of ECN signalling requires correct propagation of the ECN field up the layers [I-D.ietf-tsvwg-ecn-encap-guidelines].

7. IANA Considerations

This specification contains no IANA considerations.

8. Security Considerations

8.1. Traffic (Non-)Policing

Because the L4S service can serve all traffic that is using the capacity of a link, it should not be necessary to police access to the L4S service. In contrast, Diffserv only works if some packets get less favourable treatment than others. So Diffserv has to use traffic policers to limit how much traffic can be favoured, In turn, traffic policers require traffic contracts between users and networks as well as pairwise between networks. Because L4S will lack all this management complexity, it is more likely to work end-to-end.

During early deployment (and perhaps always), some networks will not offer the L4S service. These networks do not need to police or re-mark L4S traffic - they just forward it unchanged as best efforts traffic, as they already forward traffic with ECT(1) today. At a bottleneck, such networks will introduce some queuing and dropping. When a scalable congestion control detects a drop it will have to respond as if it is a Classic congestion control (as required in Section 2.3 of [I-D.ietf-tsvwg-ecn-l4s-id]). This will ensure safe interworking with other traffic at the 'legacy' bottleneck, but it will degrade the L4S service to no better (but never worse) than classic best efforts, whenever a legacy (non-L4S) bottleneck is encountered on a path.

Certain network operators might choose to restrict access to the L4S class, perhaps only to selected premium customers as a value-added service. Their packet classifier (item 2 in Figure 1) could identify such customers against some other field (e.g. source address range) as well as ECN. If only the ECN L4S identifier matched, but not the source address (say), the classifier could direct these packets (from non-premium customers) into the Classic queue. Allowing operators to use an additional local classifier is intended to remove any incentive to bleach the L4S identifier. Then at least the L4S ECN identifier will be more likely to survive end-to-end even though the service may not be supported at every hop. Such arrangements would only require simple registered/not-registered packet classification, rather than the managed, application-specific traffic policing against customer-specific traffic contracts that Diffserv requires.

8.2. 'Latency Friendliness'

The L4S service does rely on self-constraint - not in terms of limiting rate, but in terms of limiting latency (burstiness). It is hoped that standardisation of dynamic behaviour (cf. TCP slow-start) and self-interest will be sufficient to prevent transports from sending excessive bursts of L4S traffic, given the application's own latency will suffer most from such behaviour.

Whether burst policing becomes necessary remains to be seen. Without it, there will be potential for attacks on the low latency of the L4S service. However it may only be necessary to apply such policing reactively, e.g. punitively targeted at any deployments of new bursty malware.

8.3. Interaction between Rate Policing and L4S

As mentioned in Section 5.2, L4S should remove the need for low latency Diffserv classes. However, those Diffserv classes that give certain applications or users priority over capacity, would still be applicable. Then, within such Diffserv classes, L4S would often be applicable to give traffic low latency and low loss as well. Within such a Diffserv class, the bandwidth available to a user or application is often limited by a rate policer. Similarly, in the default Diffserv class, rate policers are used to partition shared capacity.

A classic rate policer drops any packets exceeding a set rate, usually also giving a burst allowance (variants exist where the policer re-marks non-compliant traffic to a discard-eligible Diffserv codepoint, so they may be dropped elsewhere during contention). Whenever L4S traffic encounters one of these rate policers, it will experience drops and the source has to fall back to a Classic congestion control, thus losing the benefits of L4S. So, in networks that already use rate policers and plan to deploy L4S, it will be preferable to redesign these rate policers to be more friendly to the L4S service.

This is currently a research area. It might be achieved by setting a threshold where ECN marking is introduced, such that it is just under the policed rate or just under the burst allowance where drop is introduced. This could be applied to various types of policer, e.g. [RFC2697], [RFC2698] or the 'local' (non-ConEx) variant of the ConEx congestion policer [I-D.briscoe-conex-policing]. It might also be possible to design scalable congestion controls to respond less catastrophically to loss that has not been preceded by a period of increasing delay.

The design of L4S-friendly rate policers will require a separate dedicated document. For further discussion of the interaction between L4S and Diffserv, see [I-D.briscoe-tsvwg-l4s-diffserv].

8.4. ECN Integrity

Receiving hosts can fool a sender into downloading faster by suppressing feedback of ECN marks (or of losses if retransmissions are not necessary or available otherwise). Various ways to protect TCP feedback integrity have been developed. For instance:

- The sender can test the integrity of the receiver's feedback by occasionally setting the IP-ECN field to the congestion experienced (CE) codepoint, which is normally only set by a congested link. Then the sender can test whether the receiver's feedback faithfully reports what it expects (see 2nd para of Section 20.2 of [RFC3168]).

- A network can enforce a congestion response to its ECN markings (or packet losses) by auditing congestion exposure (ConEx) [RFC7713].

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

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

Appendix C.1 of [I-D.ietf-tsvwg-ecn-l4s-id] gives more details of these techniques including their applicability and pros and cons.

9. Acknowledgements

Thanks to Wes Eddy, Karen Nielsen and David Black for their useful review comments.

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

10. References

10.1. Normative References

[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>.

10.2. Informative References

[BBR] Cardwell, N., Cheng, Y., Gunn, C., Yeganeh, S., and V. Jacobson, "BBR: Congestion-Based Congestion Control; Measuring bottleneck bandwidth and round-trip propagation time", ACM Queue (14)5, December 2016.

[DCttH15] De Schepper, K., Bondarenko, O., Tsang, I., and B. Briscoe, "'Data Centre to the Home': Ultra-Low Latency for All", 2015, <http://www.bobbriscoe.net/projects/latency/dctth_preprint.pdf>.

(Under submission)

[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.

[I-D.briscoe-conex-policing] Briscoe, B., "Network Performance Isolation using Congestion Policing", draft-briscoe-conex-policing-01 (work in progress), February 2014.

[I-D.briscoe-tsvwg-l4s-diffserv] Briscoe, B., "Interactions between Low Latency, Low Loss, Scalable Throughput (L4S) and Differentiated Services", draft-briscoe-tsvwg-l4s-diffserv-01 (work in progress), July 2018.

[I-D.ietf-quic-transport] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-15 (work in progress), October 2018.

[I-D.ietf-tcpm-accurate-ecn]
Briscoe, B., Kuehlewind, M., and R. Scheffenegger, "More Accurate ECN Feedback in TCP", draft-ietf-tcpm-accurate-ecn-07 (work in progress), July 2018.

[I-D.ietf-tcpm-alternativebackoff-ecn]
Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst, "TCP Alternative Backoff with ECN (ABE)", draft-ietf-tcpm-alternativebackoff-ecn-12 (work in progress), September 2018.

[I-D.ietf-tcpm-generalized-ecn]
Bagnulo, M. and B. Briscoe, "ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control Packets", draft-ietf-tcpm-generalized-ecn-03 (work in progress), October 2018.

[I-D.ietf-tsvwg-aqm-dualq-coupled]
Schepper, K., Briscoe, B., Bondarenko, O., and I. Tsang, "DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)", draft-ietf-tsvwg-aqm-dualq-coupled-06 (work in progress), July 2018.

[I-D.ietf-tsvwg-ecn-encap-guidelines]
Briscoe, B., Kaippallimalil, J., and P. Thaler, "Guidelines for Adding Congestion Notification to Protocols that Encapsulate IP", draft-ietf-tsvwg-ecn-encap-guidelines-10 (work in progress), March 2018.

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

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

[I-D.stewart-tsvwg-sctpecn]
Stewart, R., Tuexen, M., and X. Dong, "ECN for Stream Control Transmission Protocol (SCTP)", draft-stewart-tsvwg-sctpecn-05 (work in progress), January 2014.

[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)>.

[Mathis09]
Mathis, M., "Relentless Congestion Control", PFLDNeT'09 , May 2009, <http://www.hpcc.jp/pfldnet2009/Program_files/1569198525.pdf>.

[NewCC_Proc]
Eggert, L., "Experimental Specification of New Congestion Control Algorithms", IETF Operational Note ion-tsv-alt-cc, July 2007.

[PI2] De Schepper, K., Bondarenko, O., Tsang, I., and B. Briscoe, "PI^2 : 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>.

[RFC2697] Heinanen, J. and R. Guerin, "A Single Rate Three Color Marker", RFC 2697, DOI 10.17487/RFC2697, September 1999, <https://www.rfc-editor.org/info/rfc2697>.

[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>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <https://www.rfc-editor.org/info/rfc3168>.

[RFC3246] Davie, B., Charny, A., Bennet, J., Benson, K., Le Boudec, J., Courtney, W., Davari, S., Firoiu, V., and D. Stiliadis, "An Expedited Forwarding PHB (Per-Hop Behavior)", RFC 3246, DOI 10.17487/RFC3246, March 2002, <https://www.rfc-editor.org/info/rfc3246>.

[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>.

[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>.

[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>.

[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>.

[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>.

[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>.

[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>.

[TCP-sub-mss-w] 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, <http://www.bobbriscoe.net/projects/latency/sub-mss-w.pdf>.

Appendix A. Standardization items

The following table includes all the items that will need to be standardized to provide a full L4S architecture.

The table is too wide for the ASCII draft format, so it has been split into two, with a common column of row index numbers on the left.

The columns in the second part of the table have the following meanings:

WG: The IETF WG most relevant to this requirement. The "tcpm/iccrg" combination refers to the procedure typically used for congestion control changes, where tcpm owns the approval decision, but uses the iccrg for expert review [NewCC_Proc];

TCP: Applicable to all forms of TCP congestion control;

DCTCP: Applicable to Data Centre TCP as currently used (in controlled environments);

DCTCP bis: Applicable to an future Data Centre TCP congestion control intended for controlled environments;

XXX Prague: Applicable to a Scalable variant of XXX (TCP/SCTP/RMCAT) congestion control.

Req #	Requirement	Reference
0	ARCHITECTURE	
1	L4S IDENTIFIER	[I-D.ietf-tsvwg-ecn-l4s-id]
2	DUAL QUEUE AQM	[I-D.ietf-tsvwg-aqm-dualq-coupled]
3	Suitable ECN Feedback	[I-D.ietf-tcpm-accurate-ecn], [I-D.stewart-tsvwg-sctpecn].
	SCALABLE TRANSPORT - SAFETY ADDITIONS	
4-1	Fall back to Reno/Cubic on loss	[I-D.ietf-tsvwg-ecn-l4s-id] S.2.3, [RFC8257]
4-2	Fall back to Reno/Cubic if classic ECN bottleneck detected	[I-D.ietf-tsvwg-ecn-l4s-id] S.2.3
4-3	Reduce RTT-dependence	[I-D.ietf-tsvwg-ecn-l4s-id] S.2.3
4-4	Scaling TCP's Congestion Window for Small Round Trip Times	[I-D.ietf-tsvwg-ecn-l4s-id] S.2.3, [TCP-sub-mss-w]
	SCALABLE TRANSPORT - PERFORMANCE ENHANCEMENTS	
5-1	Setting ECT in TCP Control Packets and Retransmissions	[I-D.ietf-tcpm-generalized-ecn]
5-2	Faster-than-additive increase	[I-D.ietf-tsvwg-ecn-l4s-id] (Appx A.2.2)
5-3	Faster Convergence at Flow Start	[I-D.ietf-tsvwg-ecn-l4s-id] (Appx A.2.2)

#	WG	TCP	DCTCP	DCTCP-bis	TCP Prague	SCTP Prague	RMCAT Prague
0	tsvwg	Y	Y	Y	Y	Y	Y
1	tsvwg			Y	Y	Y	Y
2	tsvwg	n/a	n/a	n/a	n/a	n/a	n/a
3	tcpm	Y	Y	Y	Y	n/a	n/a
4-1	tcpm		Y	Y	Y	Y	Y
4-2	tcpm/ iccrg?				Y	Y	?
4-3	tcpm/ iccrg?			Y	Y	Y	?
4-4	tcpm	Y	Y	Y	Y	Y	?
5-1	tcpm	Y	Y	Y	Y	n/a	n/a
5-2	tcpm/ iccrg?			Y	Y	Y	?
5-3	tcpm/ iccrg?			Y	Y	Y	?

Authors' Addresses

Bob Briscoe (editor)
CableLabs
UK

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

Koen De Schepper
Nokia Bell Labs
Antwerp
Belgium

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

Marcelo Bagnulo
Universidad Carlos III de Madrid
Av. Universidad 30
Leganes, Madrid 28911
Spain

Phone: 34 91 6249500
Email: marcelo@it.uc3m.es
URI: http://www.it.uc3m.es

Internet Engineering Task Force
Internet-Draft
Obsoletes: 3662 (if approved)
Updates: 4594,8325 (if approved)
Intended status: Standards Track
Expires: August 19, 2019

R. Bless
Karlsruhe Institute of Technology (KIT)

February 15, 2019

A Lower Effort Per-Hop Behavior (LE PHB)
draft-ietf-tsvwg-le-phb-09

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

Table of Contents

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". Alternatively, the effect of this type of traffic on all other network traffic is strictly limited ("no harm" property). This is distinct from "best-effort" (BE) traffic since the network makes no commitment to deliver LE packets. In contrast, BE traffic receives an implied "good faith" commitment of at least some available network resources. This document proposes a Lower Effort Differentiated Services per-hop behavior (LE PHB) for handling this "optional" traffic in a differentiated services node.

2. Requirements Language

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

3. Applicability

A Lower Effort PHB is applicable for many applications that otherwise use best-effort delivery. More specifically, it is suitable for traffic and services that can tolerate strongly varying throughput for their data flows, especially periods of very low throughput or even starvation (i.e., long interruptions due to significant or even complete packet loss). Therefore, an application sending an LE

marked flow needs to be able to tolerate short or (even very) long interruptions due to the presence of severe congestion conditions during the transmission of the flow. Thus, there ought to be an expectation that packets of the LE PHB could be excessively delayed or dropped when any other traffic is present. 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 backoff 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. Alternatively, or in addition, packets can be designated for the LE PHB when the goal is to protect all other packet traffic from competition with the LE aggregate while not completely banning LE traffic from the network. An LE PHB SHOULD NOT be used for a customer’s "normal Internet" traffic nor should packets be "downgraded" to the LE PHB instead of being dropped, particularly when the packets are unauthorized traffic. The LE PHB is expected to have applicability in networks that have at least some unused capacity at certain periods.

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

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

Example uses for the LE PHB:

- For traffic caused by world-wide web search engines while they gather information from web servers.
- For software updates or dissemination of new releases of operating systems.
- For reporting errors or telemetry data from operating systems or applications.
- For backup traffic or non-time critical synchronization or mirroring traffic.
- For content distribution transfers between caches.
- For preloading or prefetching objects from web sites.

- For network news and other "bulk mail" of the Internet.
- For "downgraded" traffic from some other PHB when this does not violate the operational objectives of the other PHB.
- For multicast traffic from untrusted (e.g., non-local) sources.

4. PHB Description

The LE PHB is defined in relation to the default PHB (best-effort). A packet forwarded with the LE PHB SHOULD have lower precedence than packets forwarded with the default PHB, i.e., in the case of congestion, LE marked traffic SHOULD be dropped prior to dropping any default PHB traffic. Ideally, LE packets 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.

5. Traffic Conditioning Actions

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

As for most other PHBs an initial classification and marking can be also performed at the first DS boundary node according to the DS domain's own policies (e.g., as protection measure against untrusted sources). However, non-LE traffic (e.g., BE traffic) SHOULD NOT be remarked to LE on a regular basis without consent or knowledge of the user. See also remarks with respect to downgrading in Section 3 and Section 8.

6. Recommended DS Codepoint

The RECOMMENDED codepoint for the LE PHB is '000001'.

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

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

7. Deployment Considerations

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

- A BA classifier (Behavior Aggregate classifier, see [RFC2475]) that classifies packets according to the LE DSCP
- A dedicated LE queue
- A suitable scheduling discipline, e.g., simple priority queueing

Alternatively, implementations could use active queue management mechanisms instead of a dedicated LE queue, e.g., dropping all arriving LE packets when certain queue length or sojourn time thresholds are exceeded.

Internet-wide deployment of the LE PHB is eased by the following properties:

- No harm to other traffic: since the LE PHB has the lowest forwarding priority it does not consume resources from other PHBs.

Deployment across different provider domains with LE support causes no trust issues or attack vectors to existing (non LE) traffic. Thus, providers can trust LE markings from end-systems, i.e., there is no need to police or remark incoming LE traffic.

- No PHB parameters or configuration of traffic profiles: the LE PHB itself possesses no parameters that need to be set or configured. Similarly, since LE traffic requires no admission or policing, it is not necessary to configure traffic profiles.

- No traffic conditioning mechanisms: the LE PHB requires no traffic meters, droppers, or shapers. See also Section 5 for further discussion.

Operators of DS domains that cannot or do not want to implement the LE PHB (e.g., because there is no separate LE queue available in the corresponding nodes) SHOULD NOT drop packets marked with the LE DSCP. They SHOULD map packets with this DSCP to the default PHB and SHOULD preserve the LE DSCP marking. DS domains operators that do not implement the LE PHB should be aware that they violate the "no harm" property of LE. See also Section 8 for further discussion of forwarding LE traffic with the default PHB instead.

8. Remarking to other DSCPs/PHBs

"DSCP bleaching", i.e., setting the DSCP to ’000000’ (default PHB) is NOT RECOMMENDED for this PHB. This may cause effects that are in contrast to the original intent in protecting BE traffic from LE traffic (no harm property). In the case that a DS domain does not support the LE PHB, its nodes SHOULD treat LE marked packets with the default PHB instead (by mapping the LE DSCP to the default PHB), but they SHOULD do so without remarking to DSCP ’000000’. The reason for this is that later traversed DS domains may then have still the possibility to treat such packets according to the LE PHB.

Operators of DS domains that forward LE traffic within the BE aggregate need to be aware of the implications, i.e., induced congestion situations and quality-of-service degradation of the original BE traffic. In this case, the LE property of not harming other traffic is no longer fulfilled. To limit the impact in such cases, traffic policing of the LE aggregate MAY be used.

In case LE marked packets are effectively carried within the default PHB (i.e., forwarded as best-effort traffic) they get a better forwarding treatment than expected. For some applications and services, it is favorable if the transmission is finished earlier than expected. However, in some cases it may be against the original intention of the LE PHB user to strictly send the traffic only if

otherwise unused resources are available. In case LE traffic is mapped to the default PHB, LE traffic may compete with BE traffic for the same resources and thus adversely affect the original BE aggregate. Applications that want to ensure the lower precedence compared to BE traffic even in such cases SHOULD use additionally a corresponding Lower-than-Best-Effort transport protocol [RFC6297], e.g., LEDBAT [RFC6817].

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

9. Multicast Considerations

Basically the multicast considerations in [RFC3754] apply. However, using the Lower Effort PHB for multicast requires to pay 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 forwarding 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 get only forwarded in case of available resources at the output ports. The previously mentioned redundancy data traffic could nicely use the varying available residual bandwidth being utilized the by LE PHB, but only if the previously specific requirements 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'."

- 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

- Section 2.3 of [RFC4594] specifies: "In network segments that use IP precedence marking, only one of the two service classes can be supported, High-Throughput Data or Low-Priority Data. We RECOMMEND that the DSCP value(s) of the unsupported service class be changed to 000xx1 on ingress and changed back to original value(s) on egress of the network segment that uses precedence marking. For example, if Low-Priority Data is mapped to Standard service class, then 000001 DSCP marking MAY be used to distinguish it from Standard marked packets on egress." This document removes this recommendation, because by using the herein defined LE DSCP such remarking is not necessary. So even if Low-Priority Data is unsupported (i.e., mapped to the default PHB) the LE DSCP should be kept across the domain as RECOMMENDED in Section 8. That removed text is replaced by: "In network segments that use IP Precedence marking, the Low-Priority Data service class receives the same Diffserv QoS as the Standard service class when the LE DSCP is used for Low-Priority Data traffic. This is acceptable behavior for the Low-Priority Data service class, although it is not the preferred behavior."

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

11. The Update to RFC 8325

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

marked CS1 DSCP. [RFC4594] as updated by [RFCXXXX] recommends Low-Priority Data be marked LE DSCP."

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

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

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

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:

- Name: LE
- Value (Binary): 000001
- Value (Decimal): 1
- Reference: [RFC number of this memo]

14. Security Considerations

There are no specific security exposures for this PHB. Since it defines a new class of low forwarding priority, remarking other traffic as LE traffic may lead to quality-of-service degradation of such traffic. Thus, any attacker that is able to modify the DSCP of a packet to LE may carry out a downgrade attack. See the general security considerations in [RFC2474] and [RFC2475].

With respect to privacy, an attacker could use the information from the DSCP to infer that the transferred (probably even encrypted) content is considered of low priority or low urgency by a user, in case the DSCP was set on the user's request. On the one hand, this disclosed information is useful only if correlation with metadata (such as the user's IP address) and/or other flows reveal user identity. On the other hand, it might help an observer (e.g., a state level actor) who is interested in learning about the user's behavior from observed traffic: LE marked background traffic (such as software downloads, operating system updates, or telemetry data) may be less interesting for surveillance than general web traffic. Therefore, the LE marking may help the observer to focus on potentially more interesting traffic (however, the user may exploit this particular assumption and deliberately hide interesting traffic in the LE aggregate). Apart from such considerations, the impact of disclosed information by the LE DSCP is likely negligible in most cases given the numerous traffic analysis possibilities and general privacy threats (e.g., see [RFC6973]).

15. References

15.1. Normative References

[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>.

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>.

[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>.

[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>.

[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, Gorry Fairhurst, Ruediger Geib, and Kyle Rose provided helpful comments and (partially also text) suggestions.

Appendix C. Change History

This section briefly lists changes between Internet-Draft versions for convenience.

Changes in Version 09:

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

Changes in Version 08:

- revised two sentences as suggested by Spencer Dawkins

Changes in Version 07:

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

Changes in Version 06:

- added Multicast Considerations section with input from Toerless Eckert
- incorporated suggestions by David Black with respect to better reflect legacy CS1 handling

Changes in Version 05:

- added scavenger service class into abstract
- added some more history
- added reference for "Myth of Over-Provisioning" in RFC3439 and references to presentations w.r.t. codepoint choices
- added text to update draft-ietf-tsvwg-rtcweb-qos
- revised text on congestion control in case of remarking to BE
- added reference to DSCP measurement talk @IETF99
- small typo fixes

Changes in Version 04:

- Several editorial changes according to review from Gorry Fairhurst
- Changed the section structure a bit (moved subsections 1.1 and 1.2 into own sections 3 and 7 respectively)
- updated section 2 on requirements language
- added updates to RFC 8325
- tried to be more explicit what changes are required to RFCs 4594 and 8325

Changes in Version 03:

- Changed recommended codepoint to 000001
- Added text to explain the reasons for the DSCP choice
- Removed LE-min,LE-strict discussion
- Added one more potential use case: reporting errors or telemetry data from OSs
- Added privacy considerations to the security section (not worth an own section I think)
- Changed IANA considerations section

Changes in Version 02:

- Applied many editorial suggestions from David Black

- Added Multicast traffic use case
- Clarified what is required for deployment in section 1.2 (Deployment Considerations)
- Added text about implementations using AQMs and ECN usage
- Updated IANA section according to David Black's suggestions
- Revised text in the security section
- Changed copyright Notice to pre5378Trust200902

Changes in Version 01:

- Now obsoletes RFC 3662.
- Tried to be more precise in section 1.1 (Applicability) according to R. Geib's suggestions, so rephrased several paragraphs. Added text about congestion control
- Change section 2 (PHB Description) according to R. Geib's suggestions.
- Added RFC 2119 language to several sentences.
- Detailed the description of remarking implications and recommendations in Section 8.
- Added Section 10 to explicitly list changes with respect to RFC 4594, because this document will update it.

Appendix D. Note to RFC Editor

This section lists actions for the RFC editor during final formatting.

- Please replace the occurrences of RFCXXXX in Section 10 and Section 11 with the assigned RFC number for this document.
- Delete Appendix C.
- Delete this section.

Author's Address

Roland Bless
Karlsruhe Institute of Technology (KIT)
Kaiserstr. 12
Karlsruhe 76131
Germany

Phone: +49 721 608 46413
Email: roland.bless@kit.edu

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2019

R. Stewart
Netflix, Inc.
M. Tuexen
I. Ruengeler
Muenster Univ. of Appl. Sciences

July 2, 2018

Stream Control Transmission Protocol (SCTP) Network Address Translation Support

draft-ietf-tsvwg-natsupp-12.txt

Abstract

The Stream Control Transmission Protocol (SCTP) provides a reliable communications channel between two end-hosts in many ways similar to the Transmission Control Protocol (TCP). With the widespread deployment of Network Address Translators (NAT), specialized code has been added to NAT for TCP that allows multiple hosts to reside behind a NAT and yet use only a single globally unique IPv4 address, even when two hosts (behind a NAT) choose the same port numbers for their connection. This additional code is sometimes classified as Network Address and Port Translation (NAPT).

This document describes the protocol extensions required for the SCTP endpoints and the mechanisms for NATs necessary to provide similar features of NAPT in the single-point and multi-point traversal scenario.

Status of This Memo

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

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

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

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

Table of Contents

1. Introduction

Stream Control Transmission Protocol [RFC4960] provides a reliable communications channel between two end-hosts in many ways similar to TCP [RFC0793]. With the widespread deployment of Network Address Translators (NAT), specialized code has been added to NAT for TCP that allows multiple hosts to reside behind a NAT using private addresses (see [RFC6890]) and yet use only a single globally unique IPv4 address, even when two hosts (behind a NAT) choose the same port numbers for their connection. This additional code is sometimes classified as Network Address and Port Translation (NAPT). Please note that this document focuses on the case where the NAT maps multiple private addresses to a single public address. To date, specialized code for SCTP has not yet been added to most NATs so that only true NAT is available. The end result of this is that only one SCTP capable host can be behind a NAT and this host can only be single-homed. The only alternative for supporting legacy NATs is to use UDP encapsulation as specified in [RFC6951].

This document describes an SCTP specific variant NAT and specific packets and procedures to help NATs provide similar features of NAPT in the single-point and multi-point traversal scenario. An SCTP implementation supporting this extension will follow these procedures to assure that in both single-homed and multi-homed cases a NAT will maintain the proper state without needing to change port numbers.

It is possible and desirable to make these changes for a number of reasons:

- It is desirable for SCTP internal end-hosts on multiple platforms to be able to share a NAT's public IP address in the same way that a TCP session can use a NAT.

- If a NAT does not need to change any data within an SCTP packet it will reduce the processing burden of NAT'ing SCTP by NOT needing to execute the CRC32c checksum required by SCTP.

- Not having to touch the IP payload makes the processing of ICMP messages in NATs easier.

An SCTP-aware NAT will need to follow these procedures for generating appropriate SCTP packet formats.

When considering this feature it is possible to have multiple levels of support. At each level, the Internal Host, External Host and NAT may or may not support the features described in this document. The following table illustrates the results of the various combinations of support and if communications can occur between two endpoints.

Internal Host	NAT	External 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 we can see that when a NAT does not support the extension no communication can occur. This is because for the most part of the current situation i.e. SCTP packets sent externally from behind a NAT are discarded by the NAT. In some cases, where the NAT supports the feature but one of the two external hosts does not support the feature, communication may occur but in a limited way. For example only one host may be able to have a connection when a collision case occurs.

2. Conventions

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

3. Terminology

This document uses the following terms, which are depicted in Figure 1. Familiarity with the terminology used in [RFC4960] and [RFC5061] is assumed.

Private-Address (Priv-Addr): The private address that is known to the internal host.

Internal-Port (Int-Port): The port number that is in use by the host holding the Private-Address.

Internal-VTag (Int-VTag): The SCTP Verification Tag (VTag) that the internal host has chosen for its communication. The VTag is a unique 32-bit tag that must accompany any incoming SCTP packet for this association to the Private-Address.

External-Address (Ext-Addr): The address that an internal host is attempting to contact.

External-Port (Ext-Port): The port number of the peer process at the External-Address.

External-VTag (Ext-VTag): The Verification Tag that the host holding the External-Address has chosen for its communication. The VTag is a unique 32-bit tag that must accompany any incoming SCTP packet for this association to the External-Address.

Public-Address (Pub-Addr): The public address assigned to the NAT box which it uses as a source address when sending packets towards the External-Address.

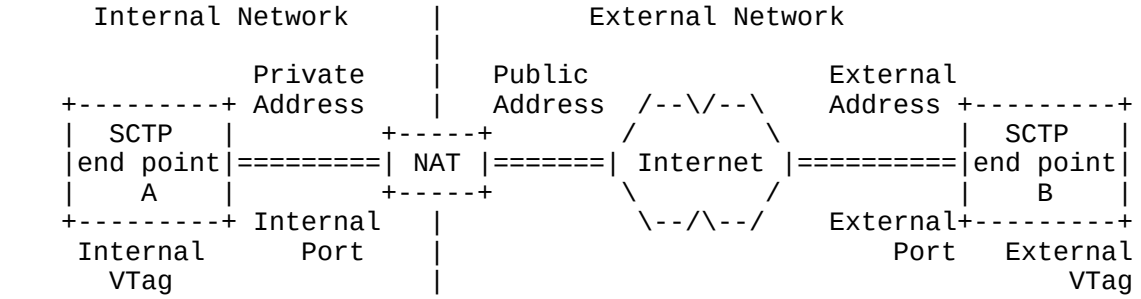


Figure 1: Basic network setup

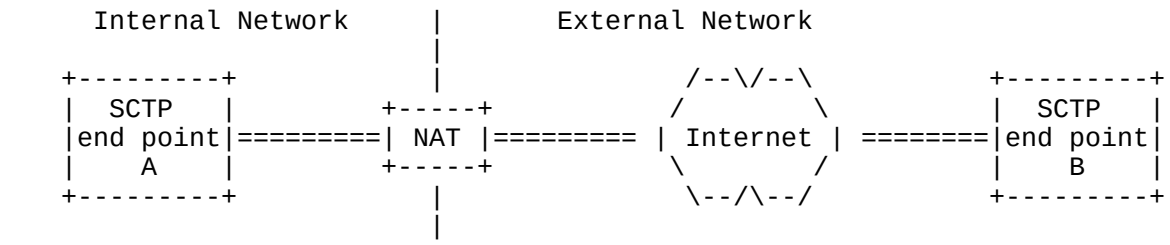
4. Motivation

4.1. SCTP NAT Traversal Scenarios

This section defines the notion of single and multi-point NAT traversal.

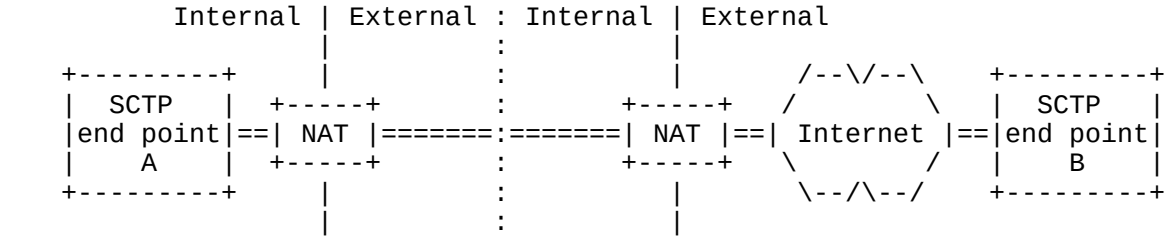
4.1.1. Single Point Traversal

In this case, all packets in the SCTP association go through a single NAT, as shown below:



Single NAT scenario

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



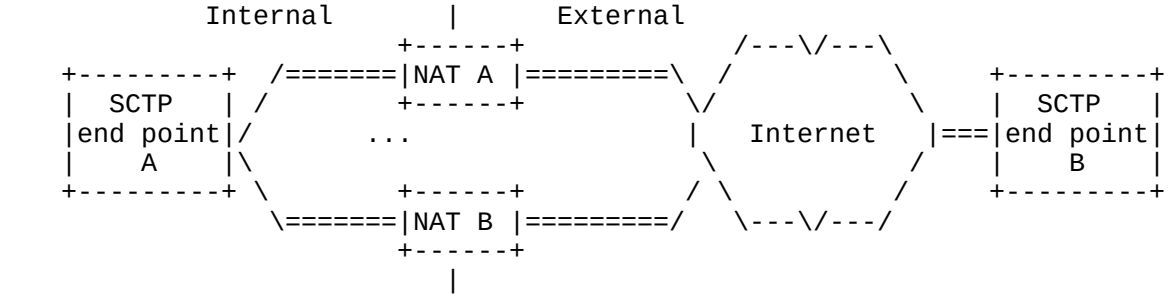
Serial NATs scenario

In this single point traversal scenario, we must acknowledge that while one of the main benefits of SCTP multi-homing is redundant paths, the NAT function represents a single point of failure in the path of the SCTP multi-home association. However, the rest of the path may still benefit from path diversity provided by SCTP multi-homing.

The two SCTP endpoints in this case can be either single-homed or multi-homed. However, the important thing is that the NAT (or NATs) in this case sees all the packets of the SCTP association.

4.1.2. Multi Point Traversal

This case involves multiple NATs and each NAT only sees some of the packets in the SCTP association. An example is shown below:



Parallel NATs scenario

This case does NOT apply to a single-homed SCTP association (i.e., BOTH endpoints in the association use only one IP address). The advantage here is that the existence of multiple NAT traversal points can preserve the path diversity of a multi-homed association for the entire path. This in turn can improve the robustness of the communication.

4.2. Limitations of Classical NAPT for SCTP

Using classical NAPT may result in changing one of the SCTP port numbers during the processing which requires the recomputation of the transport layer checksum. Whereas for UDP and TCP this can be done very efficiently, for SCTP the checksum (CRC32c) over the entire packet needs to be recomputed. This would considerably add to the NAT computational burden, however hardware support may mitigate this in some implementations.

An SCTP endpoint may have multiple addresses but only has a single port number. To make multipoint traversal work, all the NATs involved must recognize the packets they see as belonging to the same SCTP association and perform port number translation in a consistent way. One possible way of doing this is to use pre-defined table of ports and addresses configured within each NAT. Other mechanisms could make use of NAT to NAT communication. Such mechanisms are not to be deployable on a wide scale base and thus not a recommended solution. Therefore the SCTP variant of NAT has been developed.

4.3. The SCTP Specific Variant of NAT

In this section we assume that we have multiple SCTP capable hosts behind a NAT which has one Public-Address. Furthermore we are focusing in this section on the single point traversal scenario.

The modification of SCTP packets sent to the public Internet is easy. The source address of the packet has to be replaced with the Public-Address. It may also be necessary to establish some state in the NAT box to handle incoming packets, which is discussed later.

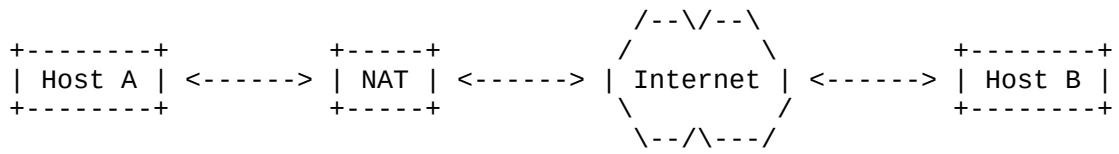
For SCTP packets coming from the public Internet the destination address of the packets has to be replaced with the Private-Address of the host the packet has to be delivered to. The lookup of the Private-Address is based on the External-VTag, External-Port, Internal-VTag and the Internal-Port.

For the SCTP NAT processing the NAT box has to maintain a table of Internal-VTag, Internal-Port, External-VTag, External-Port, Private-Address, and whether the restart procedure is disabled or not. An entry in that table is called a NAT state control block. The function Create() obtains the just mentioned parameters and returns a NAT-State control block.

The entries in this table fulfill some uniqueness conditions. There must not be more than one entry with the same pair of Internal-Port and External-Port. This rule can be relaxed, if all entries with the same Internal-Port and External-Port have the support for the restart

procedure enabled. In this case there must be no more than one entry with the same Internal-Port, External-Port and Ext-VTag and no more than one entry with the same Internal-Port, External-Port and Int-VTag.

The processing of outgoing SCTP packets containing an INIT-chunk is described in the following figure. The scenario shown is valid for all message flows in this section.



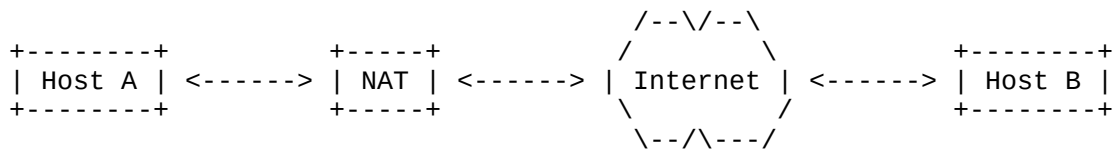

```
              INIT[Initiate-Tag]
Priv-Addr:Int-Port ------> Ext-Addr:Ext-Port
                   Ext-VTag=0

              Create(Initiate-Tag, Int-Port, 0, Ext-Port, Priv-Addr,
                     RestartSupported)
              Returns(NAT-State control block)

            Translate To:

                        INIT[Initiate-Tag]
            Pub-Addr:Int-Port ------> Ext-Addr:Ext-Port
                              Ext-VTag=0
```

Normally a NAT control block will be created. However, it is possible that there is already a NAT control block with the same External-Address, External-Port, Internal-Port, and Internal-VTag but different Private-Address. In this case the INIT MUST be dropped by the NAT and an ABORT MUST be sent back to the SCTP host with the M-Bit set and an appropriate error cause (see Section 5.1.1 for the format). The source address of the packet containing the ABORT chunk MUST be the destination address of the packet containing the INIT chunk.

It is also possible that a connection to External-Address and External-Port exists without an Internal-VTag conflict but the External-Address does not support the DISABLE_RESTART feature (noted in the NAT control block when the prior connection was established). In such a case the INIT SHOULD be dropped by the NAT and an ABORT

SHOULD be sent back to the SCTP host with the M-Bit set and an appropriate error cause (see Section 5.1.1 for the format).

The processing of outgoing SCTP packets containing no INIT-chunk is described in the following figure.

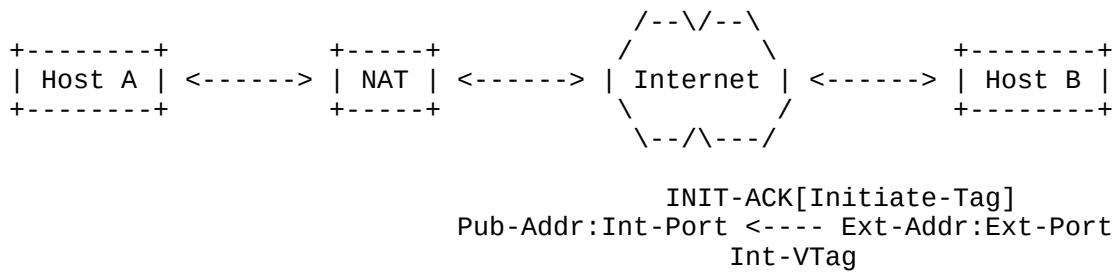


Priv-Addr:Int-Port ------> Ext-Addr:Ext-Port
Ext-VTag

Translate To:

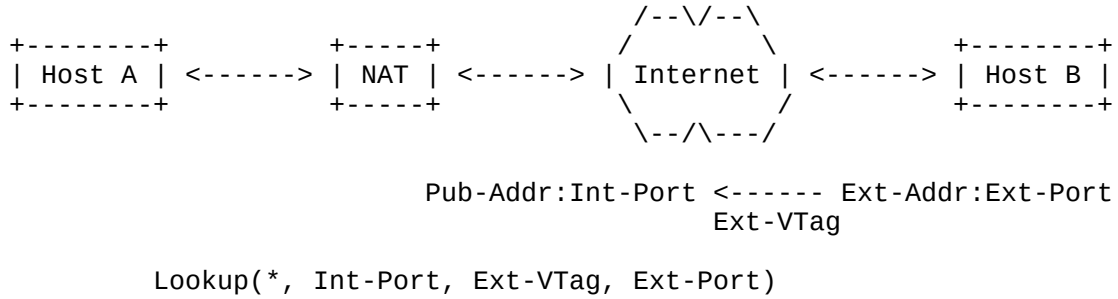
Pub-Addr:Int-Port ------> Ext-Addr:Ext-Port
Ext-VTag

The processing of incoming SCTP packets containing INIT-ACK chunks is described in the following figure. The Lookup() function getting as input the Internal-VTag, Internal-Port, External-VTag, and External-Port, returns the corresponding entry of the NAT table and updates the External-VTag by substituting it with the value of the Initiate-Tag of the INIT-ACK chunk. The wildcard character signifies that the parameter's value is not considered in the Lookup() function or changed in the Update() function, respectively.

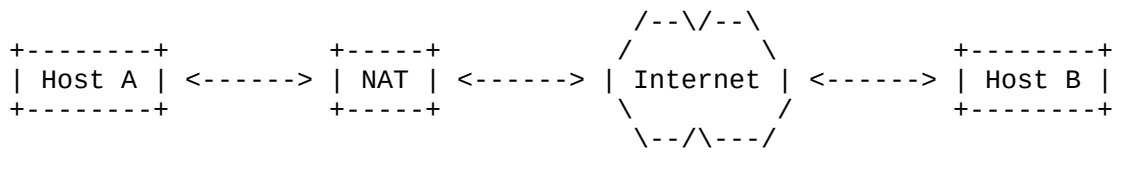


In the case Lookup fails, the SCTP packet is dropped. The Update routine inserts the External-VTag (the Initiate-Tag of the INIT-ACK chunk) in the NAT state control block.

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



The processing of other incoming SCTP packets is described in the following figure.



Pub-Addr:Int-Port <------ Ext-Addr:Ext-Port
Int-VTag

Lookup(Int-VTag, Int-Port, *, Ext-Port)

Returns(NAT-State control block containing Local-Address)

Priv-Addr:Int-Port <------ Ext-Addr:Ext-Port
Int-VTag

For an incoming packet containing an INIT-chunk a table lookup is made only based on the addresses and port numbers. If an entry with an External-VTag of zero is found, it is considered a match and the External-VTag is updated.

This allows the handling of INIT-collision through NAT.

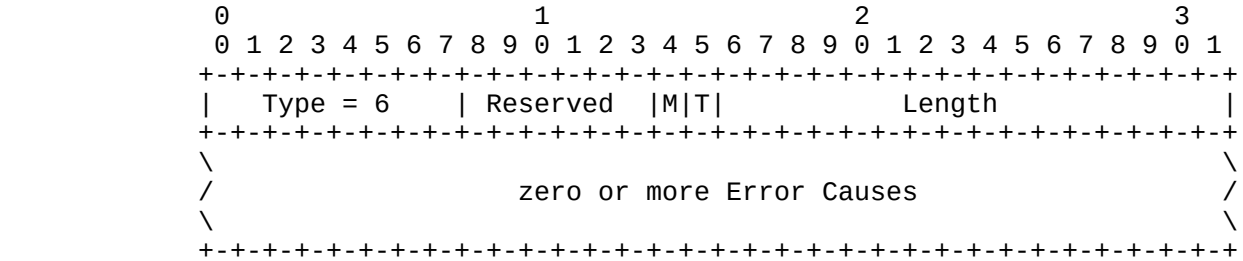
5. Data Formats

This section defines the formats used to support NAT traversal. Section 5.1 and Section 5.2 describe chunks and error causes sent by NATs and received by SCTP end points. Section 5.3 describes parameters sent by SCTP end points and used by NATs and SCTP end points.

5.1. Modified Chunks

This section presents existing chunks defined in [RFC4960] that are modified by this document.

5.1.1. Extended ABORT Chunk



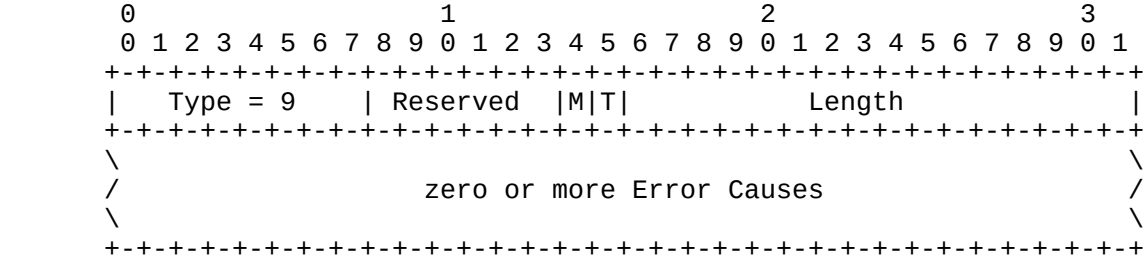
The ABORT chunk is extended to add the new 'M-bit'. The M-bit indicates to the receiver of the ABORT chunk that the chunk was not generated by the peer SCTP endpoint, but instead by a middle box.

[NOTE:

ASSIGNMENT OF M-BIT TO BE CONFIRMED BY IANA.

]

5.1.2. Extended ERROR Chunk



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

[NOTE:

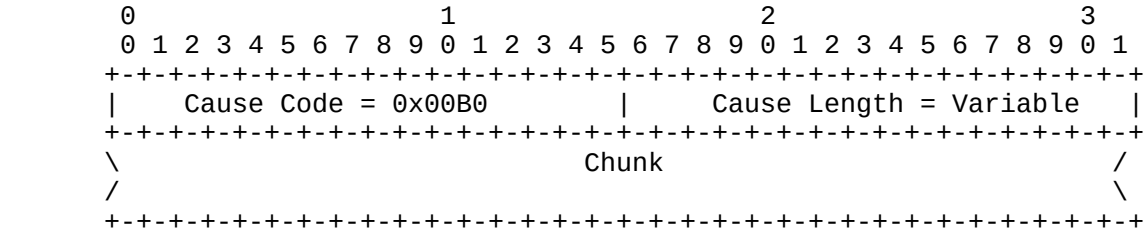
ASSIGNMENT OF M-BIT TO BE CONFIRMED BY IANA.

]

5.2. New Error Causes

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

5.2.1. VTag and Port Number Collision Error Cause



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

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

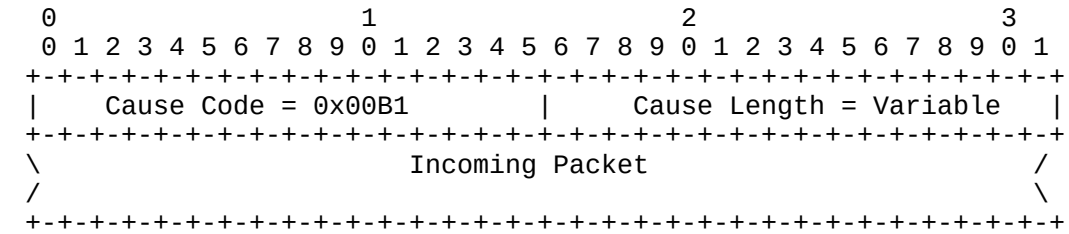
Chunk: variable length
The Cause-Specific Information is filled with the chunk that caused this error. This can be an INIT, INIT-ACK, or ASCONF chunk. Note that if the entire chunk will not fit in the ERROR chunk or ABORT chunk being sent then the bytes that do not fit are truncated.

[NOTE:

ASSIGNMENT OF CAUSE-CODE TO BE CONFIRMED BY IANA.

]

5.2.2. Missing State Error Cause



Cause Code: 2 bytes (unsigned integer)
This field holds the IANA defined cause code for the 'Missing State' Error Cause. The suggested value of this field for IANA is 0x00B1.

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

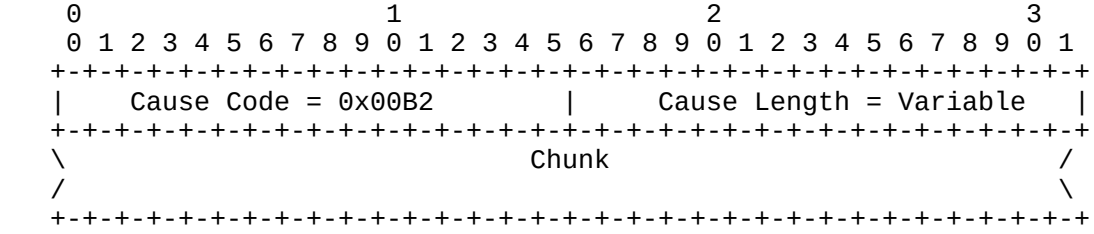
Incoming Packet: variable length
The Cause-Specific Information is filled with the IPv4 or IPv6 packet that caused this error. The IPv4 or IPv6 header MUST be included. Note that if the packet will not fit in the ERROR chunk or ABORT chunk being sent then the bytes that do not fit are truncated.

[NOTE:

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. The suggested value of this field for IANA is 0x00B2.

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

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

[NOTE:

ASSIGNMENT OF CAUSE-CODE TO BE CONFIRMED BY IANA.

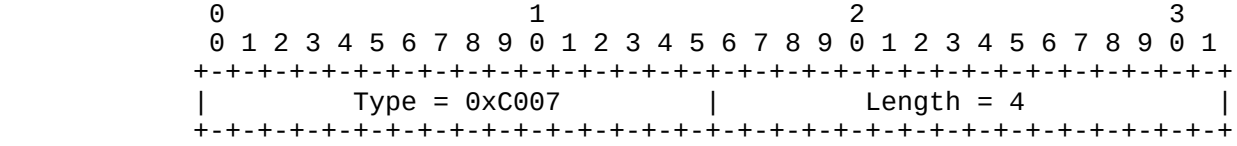
]

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. The suggested value of this field for IANA is 0xC007.

Parameter Length: 2 bytes (unsigned integer)
This field holds the length in bytes of the parameter. The value MUST be 4.

[NOTE:

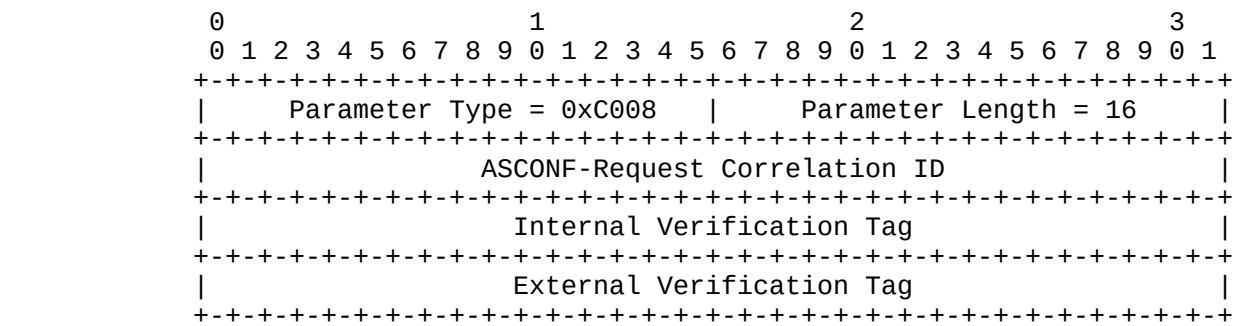
ASSIGNMENT OF PARAMETER TYPE TO BE CONFIRMED BY IANA.

]

This parameter MAY appear in INIT, INIT-ACK and ASCONF chunks and MUST NOT appear in any other chunk.

5.3.2. VTags Parameter

This parameter is used to help a NAT recover from state loss.



Parameter Type: 2 bytes (unsigned integer)
This field holds the IANA defined parameter type for the VTags Parameter. The suggested value of this field for IANA is 0xC008.

Parameter Length: 2 bytes (unsigned integer)
This field holds the length in bytes of the parameter. The value MUST be 16.

ASCONF-Request Correlation ID: 4 bytes (unsigned integer)
This is an opaque integer assigned by the sender to identify each request parameter. The receiver of the ASCONF Chunk will copy this 32-bit value into the ASCONF Response Correlation ID field of the ASCONF-ACK response parameter. The sender of the ASCONF can use this same value in the ASCONF-ACK to find which request the response is for. Note that the receiver MUST NOT change this 32-bit value.

Internal Verification Tag: 4 bytes (unsigned integer)
The Verification Tag that the internal host has chosen for its communication. The Verification Tag is a unique 32-bit tag that must accompany any incoming SCTP packet for this association to the Private-Address.

External Verification Tag: 4 bytes (unsigned integer) The Verification Tag that the host holding the External-Address has chosen for its communication. The VTag is a unique 32-bit tag that must accompany any incoming SCTP packet for this association to the External-Address.

[NOTE:

ASSIGNMENT OF PARAMETER TYPE TO BE CONFIRMED BY IANA.

]

This parameter MAY appear in ASCONF chunks and MUST NOT appear in any other chunk.

6. Procedures for SCTP End Points and NATs

6.1. Overview

When an SCTP endpoint is behind an SCTP-aware NAT a number of problems may arise as it tries to communicate with its peer:

- IP addresses can not not be included in the SCTP packet. This is discussed in Section 6.2.

- More than one host behind a NAT may pick the same VTag and source port when talking to the same peer server. This creates a situation where the NAT will not be able to tell the two associations apart. This situation is discussed in Section 6.3.

- When an SCTP endpoint is a server communicating with multiple peers and the peers are behind the same NAT, then the two endpoints cannot be distinguished by the server. This case is discussed in Section 6.4.

- A restart of a NAT during a conversation could cause a loss of its state. This problem and its solution is discussed in Section 6.5.

- NAT boxes need to deal with SCTP packets being fragmented at the IP layer. This is discussed in Section 6.6.

- An SCTP endpoint may be behind two NATs providing redundancy. The method to set up this scenario is discussed in Section 6.7.

Each of these mechanisms requires additional chunks and parameters, defined in this document, and possibly modified handling procedures from those specified in [RFC4960].

6.2. Association Setup Considerations

The association setup procedure defined in [RFC4960] allows multi-homed SCTP end points to exchange its IP-addresses by using IPv4 or IPv6 address parameters in the INIT and INIT-ACK chunks. However, this can’t be used when NATs are present.

Every association MUST initially be set up single-homed. There MUST NOT be any IPv4 Address parameter, IPv6 Address parameter, or Supported Address Types parameter in the INIT-chunk. The INIT-ACK chunk MUST NOT contain any IPv4 Address parameter or IPv6 Address parameter.

If the association should finally be multi-homed, the procedure in Section 6.7 MUST be used.

The INIT and INIT-ACK chunk SHOULD contain the Disable Restart parameter defined in Section 5.3.1.

6.3. Handling of Internal Port Number and Verification Tag Collisions

Consider the case where two hosts in the Private-Address space want to set up an SCTP association with the same service provided by some hosts in the Internet. This means that the External-Port is the same. If they both choose the same Internal-Port and Internal-VTag,

the NAT box cannot distinguish between incoming packets anymore. But this is very unlikely. The Internal-VTags are chosen at random and if the Internal-Ports are also chosen from the ephemeral port range at random this gives a 46-bit random number which has to match. In the TCP-like NAPT case the NAT box can control the 16-bit Natted Port and therefore avoid collisions deterministically.

The same can happen with the External-VTag when an INIT-ACK chunk or an ASCONF chunk is processed by the NAT.

However, in this unlikely event the NAT box MUST send an ABORT chunk with the M-bit set if the collision is triggered by an INIT or INIT-ACK chunk or send an ERROR chunk with the M-bit set if the collision is triggered by an ASCONF chunk. The M-bit is a new bit defined by this document to express to SCTP that the source of this packet is a "middle" box, not the peer SCTP endpoint (see Section 5.1.1). If a packet containing an INIT-ACK chunk triggers the collision, the corresponding packet containing the ABORT chunk MUST contain the same source and destination address and port numbers as the packet containing the INIT-ACK chunk. In the other two cases, the source and destination address and port numbers MUST be swapped.

The sender of the packet containing the INIT chunk or the receiver of the INIT-ACK chunk, upon reception of an ABORT chunk with M-bit set and the appropriate error cause code for colliding NAT table state is included, MUST reinitiate the association setup procedure after choosing a new initiate tag, if the association is in COOKIE-WAIT state. In any other state, the SCTP endpoint MUST NOT respond.

The sender of the ASCONF chunk, upon reception of an ERROR chunk with M-bit set, MUST stop adding the path to the association.

The sender of the ERROR or ABORT chunk MUST include the error cause with cause code ’VTag and Port Number Collision’ (see Section 5.2.1).

6.4. Handling of Internal Port Number Collisions

When two SCTP hosts are behind an SCTP-aware NAT it is possible that two SCTP hosts in the Private-Address space will want to set up an SCTP association with the same server running on the same host in the Internet. For the NAT, appropriate tracking may be performed by assuring that the VTags are unique between the two hosts.

But for the external SCTP server on the Internet this means that the External-Port and the External-Address are the same. If they both have chosen the same Internal-Port the server cannot distinguish between both associations based on the address and port numbers. For the server it looks like the association is being restarted. To

overcome this limitation the client sends a Disable Restart parameter in the INIT-chunk.

When the server receives this parameter it MUST do the following:

- Include a Disable Restart parameter in the INIT-ACK to inform the client that it will support the feature.
- Disable the restart procedures defined in [RFC4960] for this association.

Servers that support this feature will need to be capable of maintaining multiple connections to what appears to be the same peer (behind the NAT) differentiated only by the VTags.

The NAT, when processing the INIT-ACK, should note in its internal table that the association supports the Disable Restart extension. This note is used when establishing future associations (i.e. when processing an INIT from an internal host) to decide if the connection should be allowed. The NAT MUST do the following when processing an INIT:

- If the INIT is destined to an external address and port for which the NAT has no outbound connection, allow the INIT creating an internal mapping table.
- If the INIT matches the external address and port of an already existing connection, validate that the external server supports the Disable Restart feature, if it does allow the INIT to be forwarded.
- If the external server does not support the Disable Restart extension the NAT MUST send an ABORT with the M-bit set.

The 'Port Number Collision' error cause (see Section 5.2.3) MUST be included in the ABORT chunk.

If the collision is triggered by an ASCONF chunk, a packet containing an ERROR chunk with the 'Port Number Collision' error cause MUST be sent back.

6.5. Handling of Missing State

If the NAT box receives a packet from the internal network for which the lookup procedure does not find an entry in the NAT table, a packet containing an ERROR chunk is sent back with the M-bit set. The source address of the packet containing the ERROR chunk MUST be the destination address of the incoming SCTP packet. The

verification tag is reflected and the T-bit is set. Please note that such a packet containing an ERROR chunk SHOULD NOT be sent if the received packet contains an ABORT, SHUTDOWN-COMPLETE or INIT-ACK chunk. An ERROR chunk MUST NOT be sent if the received packet contains an ERROR chunk with the M-bit set.

When sending the ERROR chunk, the new error cause 'Missing State' (see Section 5.2.2) MUST be included and the new M-bit of the ERROR chunk MUST be set (see Section 5.1.2).

Upon reception of this ERROR chunk by an SCTP endpoint the receiver SHOULD take the following actions:

- o Validate that the verification tag is reflected by looking at the VTag that would have been included in the outgoing packet.

- o Validate that the peer of the SCTP association supports the dynamic address extension, if it does not discard the incoming ERROR chunk.

- o Generate a new ASCONF chunk containing the VTags parameter (see Section 5.3.2) and the Disable Restart parameter if the association is using the disabled restart feature. By processing this packet the NAT can recover the appropriate state. The procedures for generating an ASCONF chunk can be found in [RFC5061].

If the NAT box receives a packet for which it has no NAT table entry and the packet contains an ASCONF chunk with the VTags parameter, the NAT box MUST update its NAT table according to the verification tags in the VTags parameter and the optional Disable Restart parameter.

The peer SCTP endpoint receiving such an ASCONF chunk SHOULD either add the address and respond with an acknowledgment, if the address is new to the association (following all procedures defined in [RFC5061]). Or, if the address is already part of the association, the SCTP endpoint MUST NOT respond with an error, but instead should respond with an ASCONF-ACK chunk acknowledging the address but take no action (since the address is already in the association).

Note that it is possible that upon receiving an ASCONF chunk containing the VTags parameter the NAT will realize that it has an 'Internal Port Number and Verification Tag collision'. In such a case the NAT MUST send an ERROR chunk with the error cause code set to 'VTag and Port Number Collision' (see Section 5.2.1).

If an SCTP endpoint receives an ERROR with 'Internal Port Number and Verification Tag collision' as the error cause and the packet in the

Error Chunk contains an ASCONF with the VTags parameter, careful examination of the association is required. The endpoint MUST do the following:

- Validate that the verification tag is reflected by looking at the VTag that would have been included in the outgoing packet.

- Validate that the peer of the SCTP association supports the dynamic address extension, if it does not discard the incoming ERROR chunk.

- If the association is attempting to add an address (i.e. following the procedures in Section 6.7) then the endpoint MUST-NOT consider the address part of the association and SHOULD make no further attempt to add the address (i.e. cancel any ASCONF timers and remove any record of the path), since the NAT has a VTag collision and the association cannot easily create a new VTag (as it would if the error occurred when sending an INIT).

- If the endpoint has no other path, i. e. the procedure was executed due to missing a state in the NAT, then the endpoint MUST abort the association. This would occur only if the local NAT restarted and accepted a new association before attempting to repair the missing state (Note that this is no different than what happens to all TCP connections when a NAT looses its state).

6.6. Handling of Fragmented SCTP Packets

A NAT box MUST support IP reassembly of received fragmented SCTP packets. The fragments may arrive in any order.

When an SCTP packet has to be fragmented by the NAT box and the IP header forbids fragmentation a corresponding ICMP packet SHOULD be sent.

6.7. Multi-Point Traversal Considerations

If a multi-homed SCTP endpoint behind a NAT connects to a peer, it SHOULD first set up the association single-homed with only one address causing the first NAT to populate its state. Then it SHOULD add each IP address using ASCONF chunks sent via their respective NATs. The address to add is the wildcard address and the lookup address SHOULD also contain the VTags parameter and optionally the Disable Restart parameter as illustrated above.

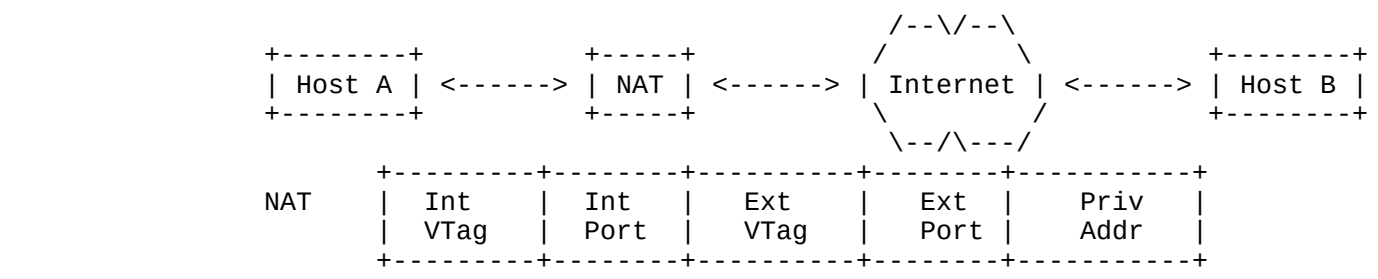
7. Various Examples of NAT Traversals

Please note that this section is informational only.

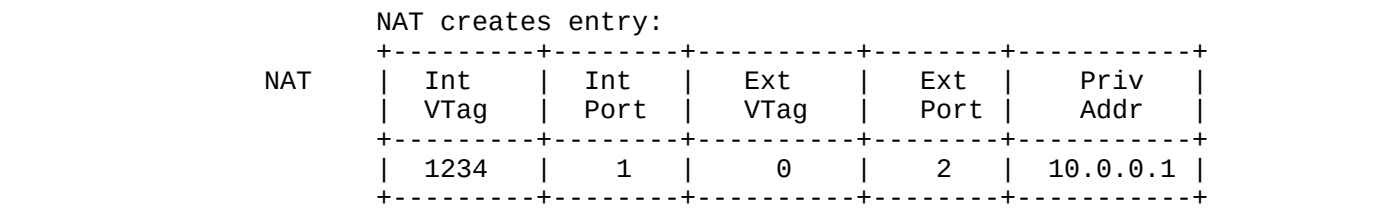
The addresses being used in the following examples are IPv4 addresses for private-use networks and for documentation as specified in [RFC6890]. However, the method described here is not limited to this NAT44 case.

7.1. Single-homed Client to Single-homed Server

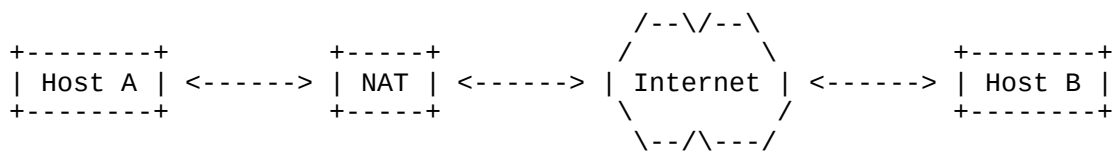
The internal client starts the association with the external server via a four-way-handshake. Host A starts by sending an INIT chunk.



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



Host B receives the INIT and sends an INIT-ACK with the NAT's external address as destination address.



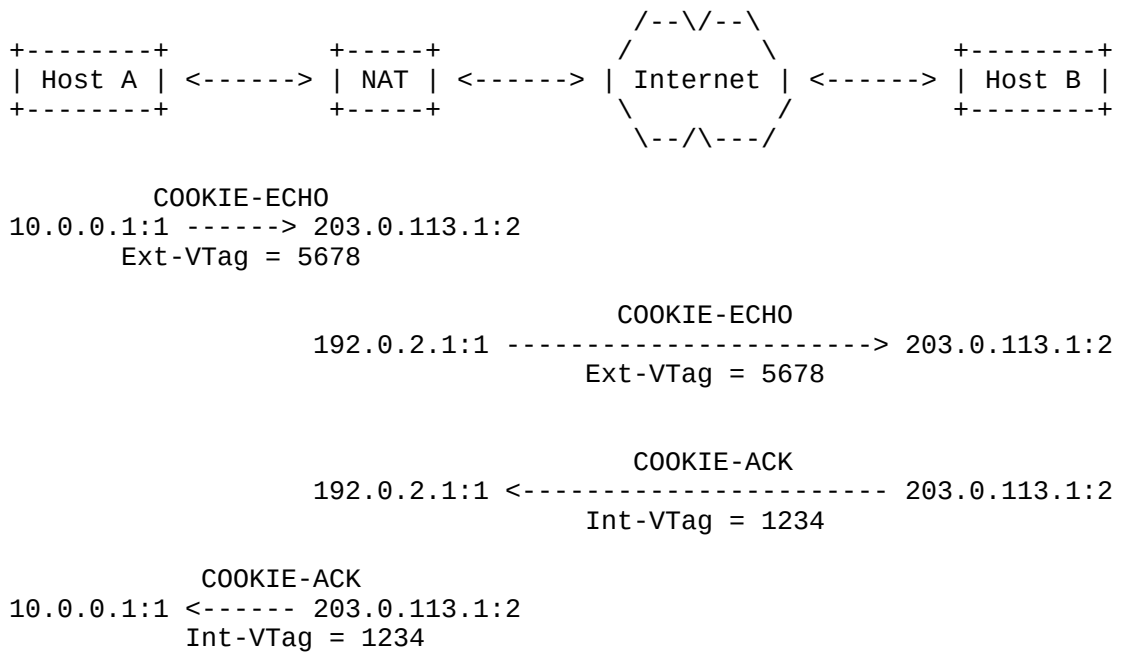
INIT-ACK[Initiate-Tag = 5678]
192.0.2.1:1 <---------- 203.0.113.1:2
Int-VTag = 1234

NAT updates entry:

NAT	Int VTag	Int Port	Ext VTag	Ext Port	Priv Addr
	1234	1	5678	2	10.0.0.1

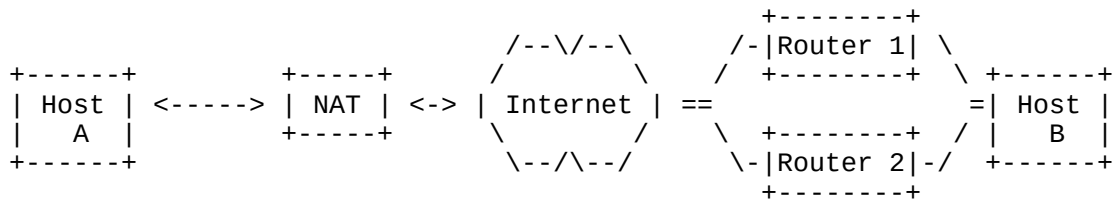
INIT-ACK[Initiate-Tag = 5678]
10.0.0.1:1 <------ 203.0.113.1:2
Int-VTag = 1234

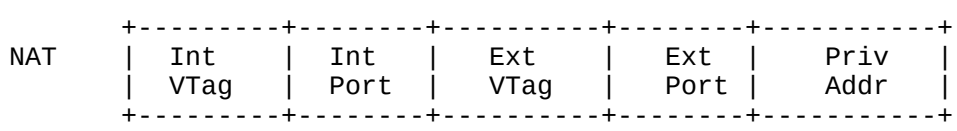
The handshake finishes with a COOKIE-ECHO acknowledged by a COOKIE-ACK.



7.2. Single-homed Client to Multi-homed Server

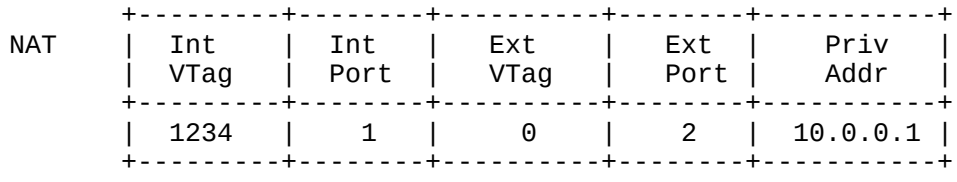
The internal client is single-homed whereas the external server is multi-homed. The client (Host A) sends an INIT like in the single-homed case.





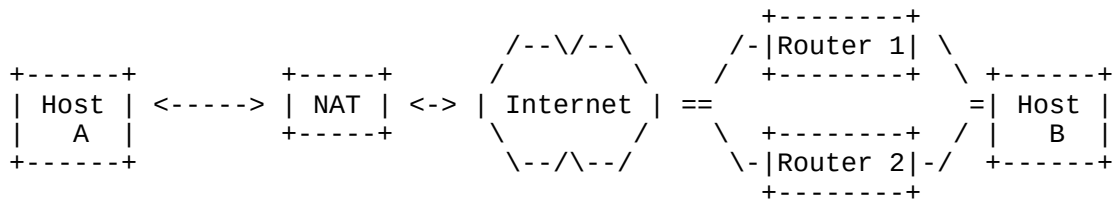
INIT[Initiate-Tag = 1234]
10.0.0.1:1 ---> 203.0.113.1:2
Ext-VTag = 0

NAT creates entry:



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

The server (Host B) includes its two addresses in the INIT-ACK chunk, which results in two NAT entries.



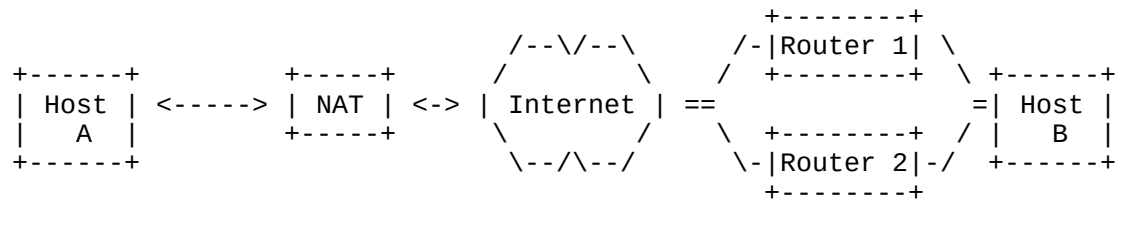
INIT-ACK[Initiate-tag = 5678, IP-Addr = 203.0.113.129]
192.0.2.1:1 <--------------------------- 203.0.113.1:2
Int-VTag = 1234

NAT does need to change the table for second address:

NAT	Int VTag	Int Port	Ext VTag	Ext Port	Priv 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

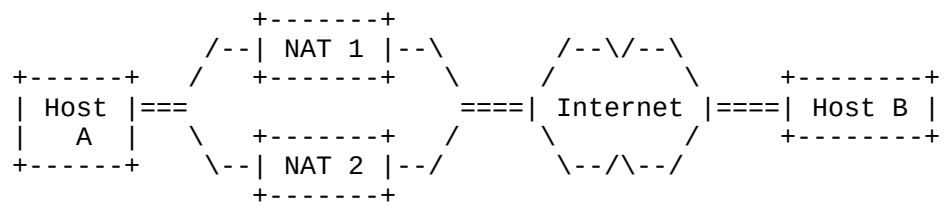
The handshake finishes with a COOKIE-ECHO acknowledged by a COOKIE-ACK.




```
       COOKIE-ECHO
10.0.0.1:1 ---> 203.0.113.1:2
       ExtVTag = 5678

                                COOKIE-ECHO
               192.0.2.1:1 -------------------------> 203.0.113.1:2
                                Ext-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
```

7.3. Multihomed Client and Server

The client (Host A) sends an INIT to the server (Host B), but does not include the second address.



NAT 1	Int VTag	Int Port	Ext VTag	Ext Port	Priv Addr

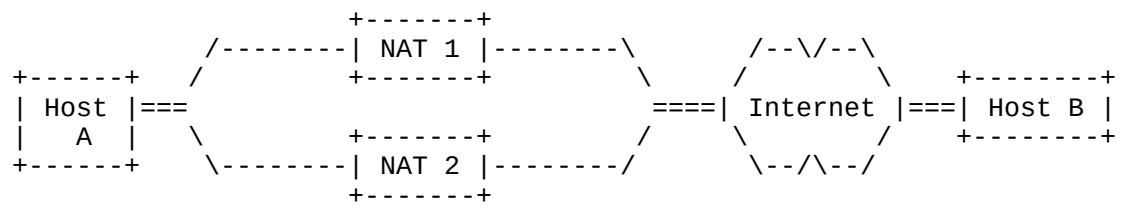
INIT[Initiate-Tag = 1234]
10.0.0.1:1 --------> 203.0.113.1:2
Ext-VTag = 0

NAT 1 creates entry:

NAT 1	Int VTag	Int Port	Ext VTag	Ext Port	Priv Addr
	1234	1	0	2	10.0.0.1

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

Host B includes its second address in the INIT-ACK, which results in two NAT entries in NAT 1.



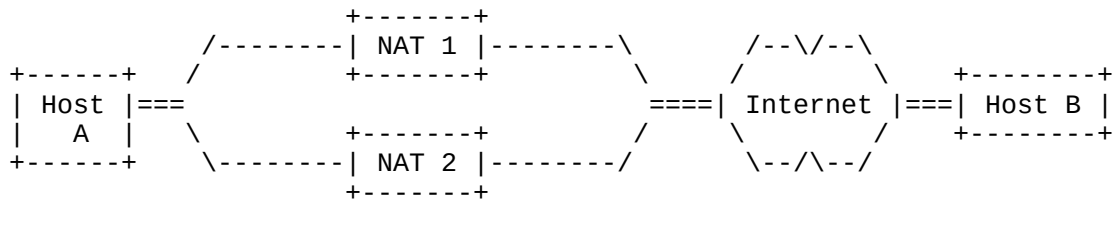
INIT-ACK[Initiate-Tag = 5678, IP-Addr = 203.0.113.129]
192.0.2.1:1 <----------------------- 203.0.113.1:2
Int-VTag = 1234

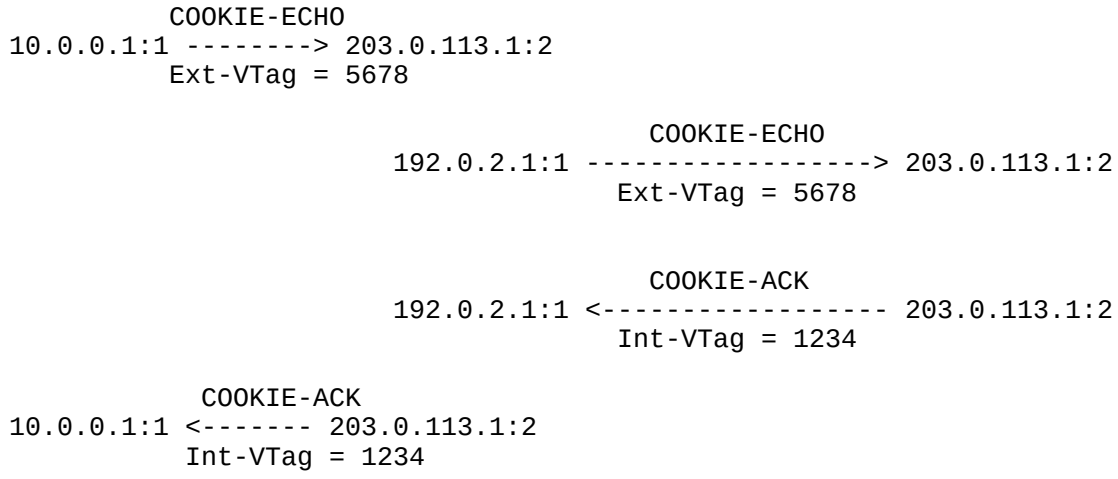
NAT 1 does not need to update the table for second address:

NAT 1	Int VTag	Int Port	Ext VTag	Ext Port	Priv Addr
	1234	1	5678	2	10.0.0.1

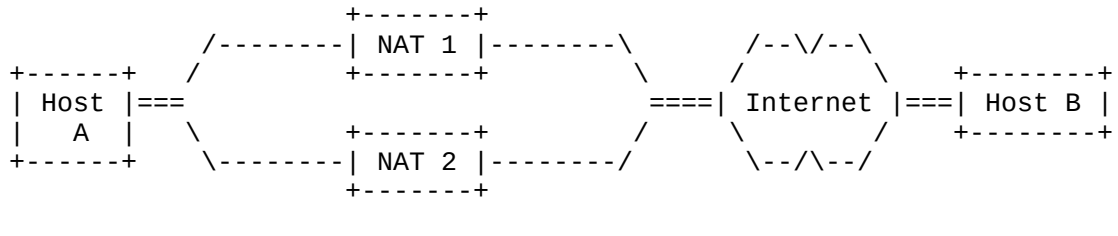
INIT-ACK[Initiate-Tag = 5678]
10.0.0.1:1 <-------- 203.0.113.1:2
Int-VTag = 1234

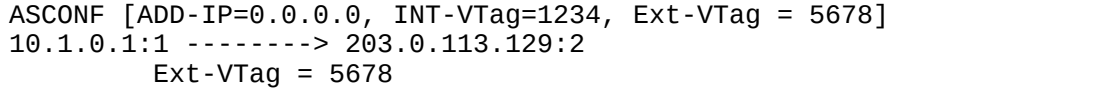
The handshake finishes with a COOKIE-ECHO acknowledged by a COOKIE-ACK.





Host A announces its second address in an ASCONF chunk. The address parameter contains an undefined address (0) to indicate that the source address should be added. The lookup address parameter within the ASCONF chunk will also contain the pair of VTags (external and internal) so that the NAT may populate its table completely with this single packet.





NAT 2 creates complete entry:

```
       +---------+--------+----------+--------+-----------+
NAT 2  |  Int    |  Int   |   Ext    |   Ext  |    Priv   |
       |  VTag   |  Port  |   VTag   |   Port |    Addr   |
       +---------+--------+----------+--------+-----------+
       |  1234   |    1   |    5678  |    2   |  10.1.0.1 |
       +---------+--------+----------+--------+-----------+
```

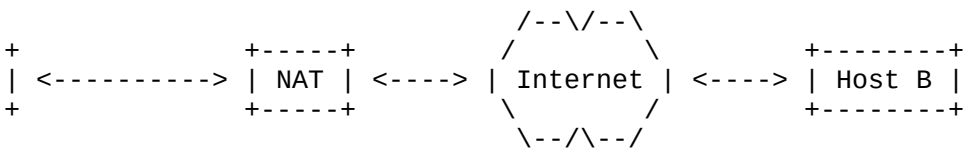
```
                  ASCONF [ADD-IP,Int-VTag=1234, Ext-VTag = 5678]
                   192.0.2.129:1 ---------------------> 203.0.113.129:2
                                          Ext-VTag = 5678

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

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

7.4. NAT Loses Its State

Association is already established between Host A and Host B, when the NAT loses its state and obtains a new public address. Host A sends a DATA chunk to Host B.

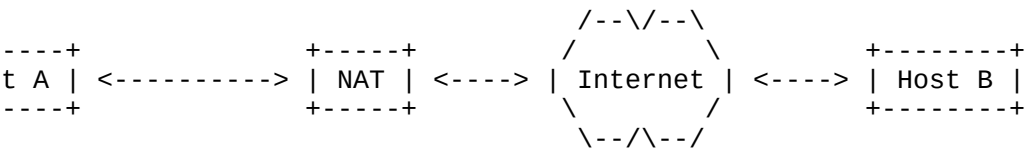
```
                                          /--\/--\
+--------+             +-----+          /        \          +--------+
| Host A | <---------> | NAT | <----> | Internet | <----> | Host B |
+--------+             +-----+          \        /          +--------+
                                  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

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

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

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




```
  ERROR [M-Bit, NAT state missing]
10.0.0.1:1 <---------- 203.0.113.1:2
          Ext-VTag = 5678
```

On reception of the ERROR message, Host A sends an ASCONF chunk indicating that the former information has to be deleted and the source address of the actual packet added.



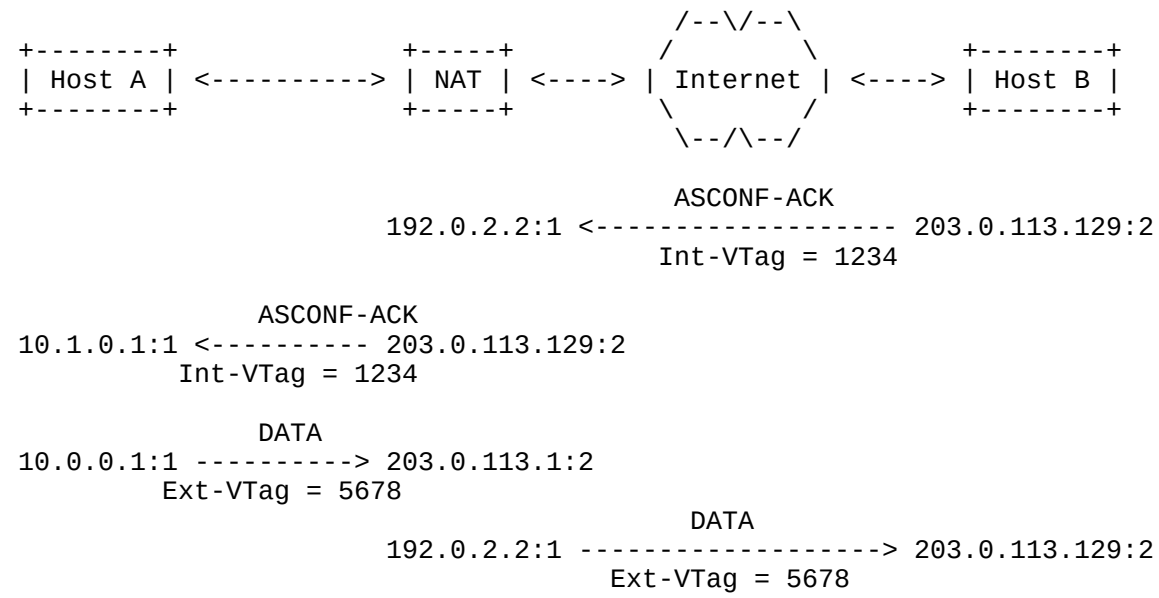

```
ASCONF [ADD-IP,DELETE-IP,Int-VTag=1234, Ext-VTag = 5678]
10.0.0.1:1 ----------> 203.0.113.129:2
          Ext-VTag = 5678
```

NAT

Int VTag	Int Port	Ext VTag	Ext Port	Priv Addr
1234	1	5678	2	10.0.0.1

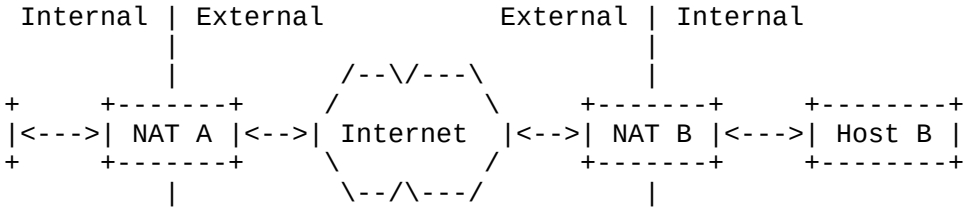
```
        ASCONF [ADD-IP,DELETE-IP,Int-VTag=1234, Ext-VTag = 5678]
                      192.0.2.2:1 --------------------> 203.0.113.129:2
                                          Ext-VTag = 5678
```

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



7.5. Peer-to-Peer Communication

If two hosts are behind NATs, they have to get knowledge of the peer's public address. This can be achieved with a so-called rendezvous server. Afterwards the destination addresses are public, and the association is set up with the help of the INIT collision. The NAT boxes create their entries according to their internal peer's point of view. Therefore, NAT A's Internal-VTag and Internal-Port are NAT B's External-VTag and External-Port, respectively. The naming of the verification tag in the packet flow is done from the sending peer's point of view.



NAT-Tables

	Int VTag	Int Port	Ext VTag	Ext Port	Priv Addr
NAT A					

	Int v-tag	Int port	Ext v-tag	Ext port	Priv Addr
NAT B					

INIT[Initiate-Tag = 1234]
10.0.0.1:1 --> 203.0.113.1:2
Ext-VTag = 0

NAT A creates entry:

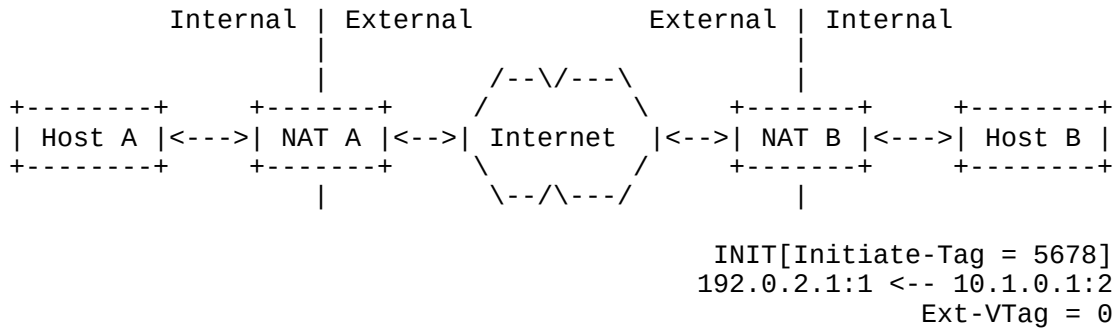
	Int VTag	Int Port	Ext VTag	Ext Port	Priv Addr
NAT A	1234	1	0	2	10.0.0.1

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

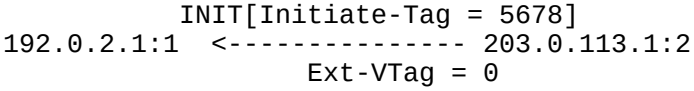
NAT B processes INIT, but cannot find an entry. The SCTP packet is silently discarded and leaves the NAT table of NAT B unchanged.

	Int VTag	Int Port	Ext VTag	Ext Port	Priv Addr
NAT B					

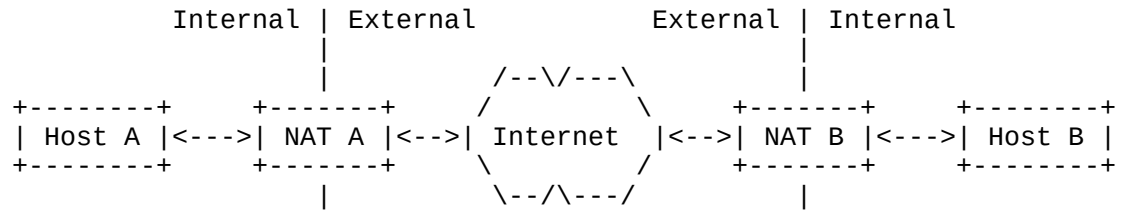
Now Host B sends INIT, which is processed by NAT B. Its parameters are used to create an entry.



NAT B	Int VTag	Int Port	Priv Addr	Ext VTag	Ext Port
	5678	2	10.1.0.1	0	1



NAT A processes INIT. As the outgoing INIT of Host A has already created an entry, the entry is found and updated:



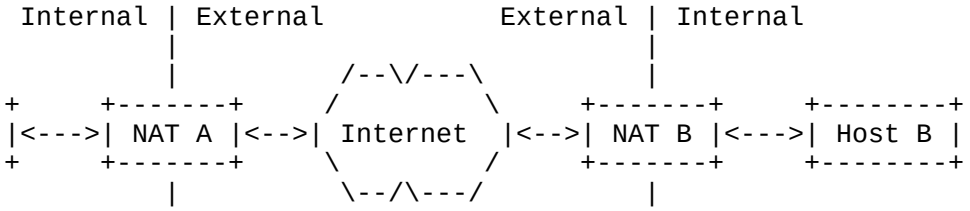
VTag != Int-VTag, but Ext-VTag == 0, find entry.

NAT A

Int VTag	Int Port	Ext VTag	Ext Port	Priv Addr
1234	1	5678	2	10.0.0.1

INIT[Initiate-tag = 5678]
10.0.0.1:1 <-- 203.0.113.1:2
Ext-VTag = 0

Host A send INIT-ACK, which can pass through NAT B:



INIT-ACK[Initiate-Tag = 1234]
10.0.0.1:1 --> 203.0.113.1:2
Ext-VTag = 5678

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

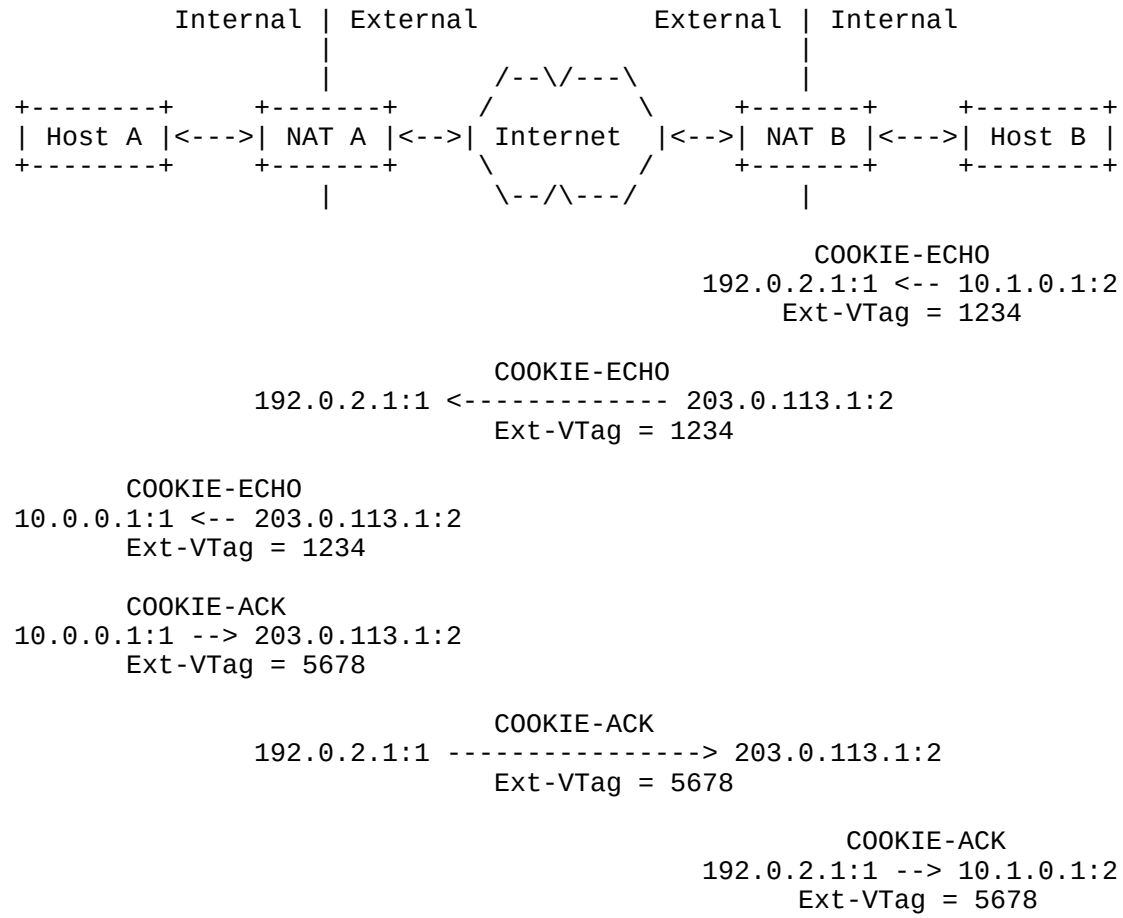
NAT B updates entry:

NAT B

Int VTag	Int Port	Ext VTag	Ext Port	Priv Addr
5678	2	1234	1	10.1.0.1

INIT-ACK[Initiate-Tag = 1234]
192.0.2.1:1 --> 10.1.0.1:2
Ext-VTag = 5678

The lookup for COOKIE-ECHO and COOKIE-ACK is successful.



8. Socket API Considerations

This section describes how the socket API defined in [RFC6458] is extended to provide a way for the application to control NAT friendliness.

Please note that this section is informational only.

A socket API implementation based on [RFC6458] is extended by supporting one new read/write socket option.

8.1. Get or Set the NAT Friendliness (SCTP_NAT_FRIENDLY)

This socket option uses the option_level IPPROTO_SCTP and the option_name SCTP_NAT_FRIENDLY. It can be used to enable/disable the NAT friendliness for future associations and retrieve the value for future and specific ones.

```
struct sctp_assoc_value {
  sctp_assoc_t assoc_id;
  uint32_t assoc_value;
};
```

assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets the application may fill in an association identifier or SCTP_FUTURE_ASSOC for this query. It is an error to use SCTP_{CURRENT|ALL}_ASSOC in assoc_id.

assoc_value: A non-zero value indicates a NAT-friendly mode.

9. IANA Considerations

[NOTE to RFC-Editor:

"RFCXXXX" is to be replaced by the RFC number you assign this document.

]

[NOTE to RFC-Editor:

The suggested values for the chunk type and the chunk parameter types are tentative and to be confirmed by IANA.

]

This document (RFCXXXX) is the reference for all registrations described in this section. The suggested changes are described below.

9.1. New Chunk Flags for Two Existing Chunk Types

As defined in [RFC6096] two chunk flags have to be assigned by IANA for the ERROR chunk. The suggested value for the T bit is 0x01 and for the M bit is 0x02.

This requires an update of the "ERROR Chunk Flags" registry for SCTP:

ERROR Chunk Flags

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	

As defined in [RFC6096] one chunk flag has to be assigned by IANA for the ABORT chunk. The suggested value of the M bit is 0x02.

This requires an update of the "ABORT Chunk Flags" registry for SCTP:

ABORT Chunk Flags

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	

9.2. Three New Error Causes

Three error causes have to be assigned by IANA. It is suggested to use the values given below.

This requires three additional lines in the "Error Cause Codes" registry for SCTP:

Error Cause Codes

Value	Cause Code	Reference
176	VTag and Port Number Collision	[RFCXXXX]
177	Missing State	[RFCXXXX]
178	Port Number Collision	[RFCXXXX]

9.3. Two New Chunk Parameter Types

Two chunk parameter types have to be assigned by IANA. It is suggested to use the values given below. IANA should assign these values from the pool of parameters with the upper two bits set to '11'.

This requires two additional lines in the "Chunk Parameter Types" registry for SCTP:

Chunk Parameter Types

ID Value	Chunk Parameter Type	Reference
49159	Disable Restart (0xC007)	[RFCXXXX]
49160	VTags (0xC008)	[RFCXXXX]

10. Security Considerations

State maintenance within a NAT is always a subject of possible Denial Of Service attacks. This document recommends that at a minimum a NAT runs a timer on any SCTP state so that old association state can be cleaned up.

For SCTP end points, this document does not add any additional security considerations to the ones given in [RFC4960], [RFC4895], and [RFC5061]. In particular, SCTP is protected by the verification tags and the usage of [RFC4895] against off-path attackers.

11. Acknowledgments

The authors wish to thank Gorry Fairhurst, Bryan Ford, David Hayes, Alfred Hines, Karen E. E. Nielsen, Henning Peters, Timo Voelker, Dan Wing, and Qiaobing Xie for their invaluable comments.

In addition, the authors wish to thank David Hayes, Jason But, and Grenville Armitage, the authors of [DOI_10.1145_1496091.1496095], for their suggestions.

12. References

12.1. Normative References

[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>.

[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>.

[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>.

12.2. 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, DOI 10.1145/1496091.1496095, December 2008.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[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>.

Authors' Addresses

Randall R. Stewart
Netflix, Inc.
Chapin, SC 29036
US

Email: randall@lakerest.net

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

Email: tuexen@fh-muenster.de

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

Email: i.ruengeler@fh-muenster.de

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.

Table of Contents

1. Introduction

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

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

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

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

Note that when reading this document one must use care to assure that a field or item is not updated further on within the document. Since this document is a historical record of the sequential changes that

have been found necessary at various inter-op events and through discussion on the list, the last delta in the text is the one which should be applied.

2. Conventions

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

3. Corrections to RFC 4960

[NOTE to RFC-Editor:

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

]

3.1. Path Error Counter Threshold Handling

3.1.1. Description of the Problem

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

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

3.1.2. Text Changes to the Document

Old text: (Section 8.3)

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

New text: (Section 8.3)

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

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

3.1.3. Solution Description

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

3.2. Upper Layer Protocol Shutdown Request Handling

3.2.1. Description of the Problem

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

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

3.2.2. Text Changes to the Document

Old text: (Section 9.2)

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

New text: (Section 9.2)

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

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

3.2.3. Solution Description

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

3.3. Registration of New Chunk Types

3.3.1. Description of the Problem

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

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

3.3.2. Text Changes to the Document

Old text: (Section 14.1)

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

New text: (Section 14.1)

The assignment of new chunk type codes is done through an IETF Consensus action, as defined in [RFC8126]. Documentation of the chunk type MUST contain the following information:

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

3.3.3. Solution Description

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

3.4. Variable Parameters for INIT Chunks

3.4.1. Description of the Problem

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

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

3.4.2. Text Changes to the Document

Old text: (Section 3.3.2)

Variable Parameters Status Type Value

IPv4 Address (Note 1) Optional 5 IPv6 Address
(Note 1) Optional 6 Cookie Preservative
Optional 9 Reserved for ECN Capable (Note 2) Optional
32768 (0x8000) Host Name Address (Note 3) Optional
11 Supported Address Types (Note 4) Optional 12

New text: (Section 3.3.2)

Variable Parameters	Status	Type Value
IPv4 Address (Note 1)	Optional	5
IPv6 Address (Note 1)	Optional	6
Cookie Preservative	Optional	9
Reserved for ECN Capable (Note 2)	Optional	32768 (0x8000)
Host Name Address (Note 3)	Optional	11
Supported Address Types (Note 4)	Optional	12

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

3.4.3. Solution Description

Fix the formatting of the table.

3.5. CRC32c Sample Code on 64-bit Platforms

3.5.1. Description of the Problem

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

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

3.5.2. Text Changes to the Document

```
---------
Old text: (Appendix C)
---------

unsigned long
generate_crc32c(unsigned char *buffer, unsigned int length)
{
  unsigned int i;
  unsigned long crc32 = ~0L;

---------
New text: (Appendix C)
---------

unsigned long
generate_crc32c(unsigned char *buffer, unsigned int length)
{
  unsigned int i;
  unsigned long crc32 = 0xffffffffL;
```

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

3.5.3. Solution Description

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

3.6. Endpoint Failure Detection

3.6.1. Description of the Problem

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

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

3.6.2. Text Changes to the Document

Old text: (Section 8.1)

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

New text: (Section 8.1)

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

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

3.6.3. Solution Description

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

3.7. Data Transmission Rules

3.7.1. Description of the Problem

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

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

3.7.2. Text Changes to the Document

Old text: (Section 6.1 A))

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

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

New text: (Section 6.1 A))

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

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

3.7.3. Solution Description

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

3.8. T1-Cookie Timer

3.8.1. Description of the Problem

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

This issue was reported as an Errata for [RFC4960] with Errata ID 4400.

3.8.2. Text Changes to the Document

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

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

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

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

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

3.8.3. Solution Description

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

3.9. Miscellaneous Typos

3.9.1. Description of the Problem

While processing [RFC4960] some typos were not caught.

One typo was reported as an Errata for [RFC4960] with Errata ID 5003.

3.9.2. Text Changes to the Document

Old text: (Section 1.6)

Transmission Sequence Numbers wrap around when they reach 2**32 - 1. That is, the next TSN a DATA chunk MUST use after transmitting TSN = 2*32 - 1 is TSN = 0.

New text: (Section 1.6)

Transmission Sequence Numbers wrap around when they reach 2**32 - 1. That is, the next TSN a DATA chunk MUST use after transmitting TSN = 2**32 - 1 is TSN = 0.

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

Old text: (Section 3.3.10.9)

No User Data: This error cause is returned to the originator of a

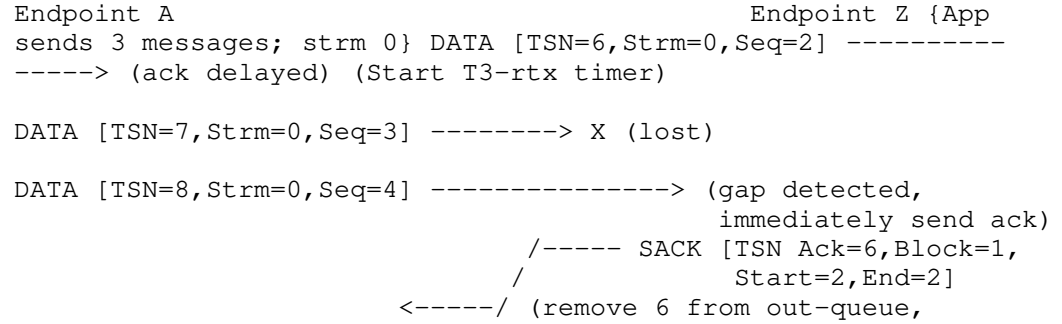
DATA chunk if a received DATA chunk has no user data.

New text: (Section 3.3.10.9)

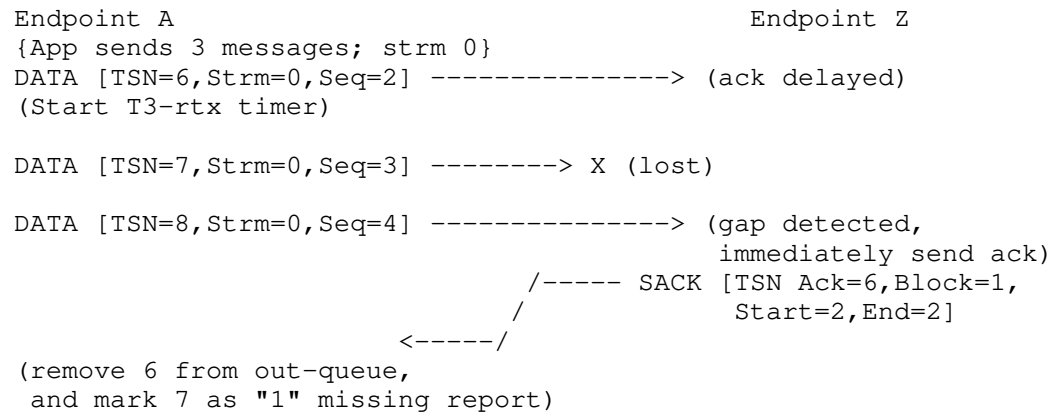
No User Data: This error cause is returned to the originator of a DATA chunk if a received DATA chunk has no user data.

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

Old text: (Section 6.7, Figure 9)



New text: (Section 6.7, Figure 9)



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

Old text: (Section 6.10)

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

including the SCTP packet and IP headers, MUST be less that or equal to the current Path MTU.

New text: (Section 6.10)

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

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

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

o Receive Unacknowledged Message

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

New text: (Section 10.1 O))

O) Receive Unacknowledged Message

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

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

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)

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

New text: (Section 7.2.2)

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

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

3.12.3. Solution Description

The new text defines the exact order of adjustments of partial_bytes_acked and cwnd in the congestion avoidance phase.

3.13. HEARTBEAT ACK and the association error counter

3.13.1. Description of the Problem

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

3.13.2. Text Changes to the Document

Old text: (Section 8.1)

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

New text: (Section 8.1)

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

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

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)

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

New text: (Section 7.2.1)

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

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

3.16.3. Solution Description

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

3.17. Automatically Confirmed Addresses

3.17.1. Description of the Problem

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

3.17.2. Text Changes to the Document

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))

- Stream Sequence Number - this value is returned indicating the Stream Sequence Number that was associated with the message.

- partial flag - if this returned flag is set to 1, then this message is a partial delivery of the whole message. When this flag is set, the stream id and Stream Sequence Number MUST accompany this receive. When this flag is set to 0, it indicates that no more deliveries will be received for this Stream Sequence Number.

New text: (Section 10.1 N))

- stream sequence number - this value is returned indicating the Stream Sequence Number that was associated with the message.

- partial flag - if this returned flag is set to 1, then this message is a partial delivery of the whole message. When this flag is set, the stream id and stream sequence number must accompany this primitive. When this flag is set to 0, it indicates that no more deliveries will be received for this stream sequence number.

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

Old text: (Section 10.1 O))

- 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.

- Duplicate TSNs are not considered in partial_bytes_acked although they confirm that the DATA chunks were successfully received by the peer.

3.22.2. Text Changes to the Document

Old text: (Section 7.2.2)

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

New text: (Section 7.2.2)

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

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:

- SCTP MAY increment cwnd by 1*MTU.
- SCTP SHOULD increment cwnd by one 1*MTU once per RTT when the sender has cwnd or more bytes of data outstanding for the corresponding transport address.
- SCTP MUST NOT increment cwnd by more than 1*MTU per RTT.

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

Old text: (Section 7.2.2)

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

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

New text: (Section 7.2.2)

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

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

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

This text has been modified by multiple errata. It includes modifications from Section 3.12 and Section 3.22. It is in final form, and is not further updated in this document.

3.26.3. Solution Description

The basic guidelines for incrementing cwnd during the congestion avoidance phase are added into Section 7.2.2. The guidelines include the normative language and are aligned with [RFC5681].

The algorithm from Section 7.2.2 is improved to not allow increasing cwnd by more than 1*MTU per RTT.

3.27. Refresh of cwnd and ssthresh after Idle Period

3.27.1. Description of the Problem

[RFC4960] prescribes to adjust cwnd per RTO if the endpoint does not transmit data on a given transport address. In addition to that, it prescribes to set cwnd to the initial value after a sufficiently long idle period. The latter is excessive. Moreover, it is unclear what is a sufficiently long idle period.

[RFC4960] doesn't specify the handling of ssthresh in the idle case. If ssthresh is reduced due to a packet loss, ssthresh is never recovered. So traffic can end up in Congestion Avoidance all the time, resulting in a low sending rate and bad performance. The problem is even more serious for SCTP because in a multi-homed SCTP association traffic that switches back to the previously failed primary path will also lead to the situation where traffic ends up in Congestion Avoidance.

3.27.2. Text Changes to the Document

Old text: (Section 7.2.1)

- The initial cwnd before DATA transmission or after a sufficiently long idle period MUST be set to min(4*MTU, max (2*MTU, 4380 bytes)).

New text: (Section 7.2.1)

- The initial cwnd before DATA transmission MUST be set to min(4*MTU, max (2*MTU, 4380 bytes)).

Old text: (Section 7.2.1)
---

o When the endpoint does not transmit data on a given transport address, the cwnd of the transport address should be adjusted to max(cwnd/2, 4*MTU) per RTO.

New text: (Section 7.2.1)
---
o While the endpoint does not transmit data on a given transport address, the cwnd of the transport address SHOULD be adjusted to max(cwnd/2, 4*MTU) once per RTO. Before the first cwnd adjustment, the ssthresh of the transport address SHOULD be set to the cwnd.

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

3.27.3. Solution Description

A rule about cwnd adjustment after a sufficiently long idle period is removed.

The text is updated to describe the ssthresh handling. When the idle period is detected, the cwnd value is stored to the ssthresh value.

3.28. Window Updates After Receiver Window Opens Up

3.28.1. Description of the Problem

The sending of SACK chunks for window updates is only indirectly referenced in [RFC4960], Section 6.2, where it is stated that an SCTP receiver must not generate more than one SACK for every incoming packet, other than to update the offered window.

However, the sending of window updates when the receiver window opens up is necessary to avoid performance problems.

3.28.2. Text Changes to the Document

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:

- It states that a SACK should be sent to the source address of an incoming DATA. However, it is known that other SACK policies (e.g. sending SACKs always to the primary path) may be more beneficial in some situations.
- Initially it states that an endpoint should always transmit DATA chunks to the primary path. Then it states that the rule for transmittal of reply chunks should also be followed if the endpoint is bundling DATA chunks together with the reply chunk which contradicts with the first statement to always transmit DATA

chunks to the primary path. Some implementations were having problems with it and sent DATA chunks bundled with reply chunks to a different destination address than the primary path that caused many gaps.

3.29.2. Text Changes to the Document

Old text: (Section 6.4)

An endpoint SHOULD transmit reply chunks (e.g., SACK, HEARTBEAT ACK, etc.) to the same destination transport address from which it received the DATA or control chunk to which it is replying. This rule should also be followed if the endpoint is bundling DATA chunks together with the reply chunk.

However, when acknowledging multiple DATA chunks received in packets from different source addresses in a single SACK, the SACK chunk may be transmitted to one of the destination transport addresses from which the DATA or control chunks being acknowledged were received.

New text: (Section 6.4)

An endpoint SHOULD transmit reply chunks (e.g., INIT ACK, COOKIE ACK, HEARTBEAT ACK, etc.) in response to control chunks to the same destination transport address from which it received the control chunk to which it is replying.

The selection of the destination transport address for packets containing SACK chunks is implementation dependent. However, an endpoint SHOULD NOT vary the destination transport address of a SACK when it receives DATA chunks coming from the same source address.

When acknowledging multiple DATA chunks received in packets from different source addresses in a single SACK, the SACK chunk MAY be transmitted to one of the destination transport addresses from which the DATA or control chunks being acknowledged were received.

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

3.29.3. Solution Description

The SACK transmission policy is left implementation dependent but it is specified to not vary the destination address of a packet containing a SACK chunk unless there are reasons for it as it may negatively impact RTT measurement.

A confusing statement that prescribes to follow the rule for transmittal of reply chunks when the endpoint is bundling DATA chunks together with the reply chunk is removed.

3.30. Outstanding Data, Flightsize and Data In Flight Key Terms

3.30.1. Description of the Problem

[RFC4960] uses outstanding data, flightsize and data in flight key terms in formulas and statements but their definitions are not provided in Section 1.3. Furthermore, outstanding data does not include DATA chunks which are classified as lost but which have not been retransmitted yet and there is a paragraph in Section 6.1 of [RFC4960] where this statement is broken.

3.30.2. Text Changes to the Document

Old text: (Section 1.3)

- Congestion window (cwnd): An SCTP variable that limits the data, in number of bytes, a sender can send to a particular destination transport address before receiving an acknowledgement.

...

- Outstanding TSN (at an SCTP endpoint): A TSN (and the associated DATA chunk) that has been sent by the endpoint but for which it has not yet received an acknowledgement.

New text: (Section 1.3)

- Outstanding TSN (at an SCTP endpoint): A TSN (and the associated DATA chunk) that has been sent by the endpoint but for which it has not yet received an acknowledgement.

- Outstanding data (or Data outstanding or Data in flight): The total amount of the DATA chunks associated with outstanding TSNs. A retransmitted DATA chunk is counted once in outstanding data. A DATA chunk which is classified as lost but which has not been retransmitted yet is not in outstanding data.

- Flightsize: The amount of bytes of outstanding data to a particular destination transport address at any given time.

- Congestion window (cwnd): An SCTP variable that limits outstanding data, in number of bytes, a sender can send to a particular destination transport address before receiving an acknowledgement.

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

Old text: (Section 6.1)

C) When the time comes for the sender to transmit, before sending new DATA chunks, the sender MUST first transmit any outstanding DATA chunks that are marked for retransmission (limited by the current cwnd).

New text: (Section 6.1)

C) When the time comes for the sender to transmit, before sending new DATA chunks, the sender MUST first transmit any DATA chunks that are marked for retransmission (limited by the current cwnd).

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

3.30.3. Solution Description

Now Section 1.3, Key Terms, includes explanations of outstanding data, data in flight and flightsize key terms. Section 6.1 is corrected to properly use the outstanding data term.

3.31. CWND Degradation due to Max.Burst

3.31.1. Description of the Problem

Some implementations were experiencing a degradation of cwnd because of the Max.Burst limit. This was due to misinterpretation of the suggestion in [RFC4960], Section 6.1, on how to use the Max.Burst parameter when calculating the number of packets to transmit.

3.31.2. Text Changes to the Document

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 where successfully tested in all nine interoperability tests for SCTP, but the third one has never been successfully tested. Therefore, the Host Name Address parameter should be deprecated.

3.41.2. Text Changes to the Document

Old text: (Section 3.3.2)

Note 3: An INIT chunk MUST NOT contain more than one Host Name Address parameter. Moreover, the sender of the INIT MUST NOT combine any other address types with the Host Name Address in the INIT. The receiver of INIT MUST ignore any other address types if the Host Name Address parameter is present in the received INIT chunk.

New text: (Section 3.3.2)

Note 3: An INIT chunk MUST NOT contain the Host Name Address parameter. The receiver of an INIT chunk containing a Host Name Address parameter MUST send an ABORT and MAY include an Error Cause indicating an Unresolvable Address.

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:

- A short port name, consisting entirely of letters (A-Z and a-z), digits (0-9), and punctuation characters from "-_+./*" (not including the quotes).

- The port number that is requested for registration.

- A short English phrase describing the port's purpose.

- Name and contact information for the person or entity performing the registration, and possibly a reference to a document defining the port's use. Registrations coming from IETF working groups need only name the working group, but indicating a contact person is recommended.

Registrants are encouraged to follow these guidelines when submitting a registration.

- A port name SHOULD NOT be registered for more than one SCTP port

number.

- A port name registered for TCP MAY be registered for SCTP as well. Any such registration SHOULD use the same port number as the existing TCP registration.

- Concrete intent to use a port SHOULD precede port registration. For example, existing TCP ports SHOULD NOT be registered in advance of any intent to use those ports for SCTP.

New text: (Section 14.5)

SCTP services can use contact port numbers to provide service to unknown callers, as in TCP and UDP. IANA is therefore requested to open the existing Port Numbers registry for SCTP using the following rules, which we intend to mesh well with existing Port Numbers registration procedures. An IESG-appointed Expert Reviewer supports IANA in evaluating SCTP port allocation requests, according to the procedure defined in [RFC8126]. The details of this process are defined in [RFC6335].

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

3.44.3. Solution Description

[RFC6335] was integrated and the reference was updated to [RFC8126].

3.45. Integration of RFC 7053

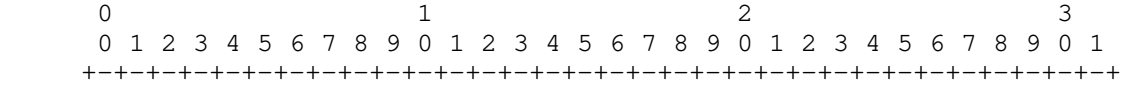
3.45.1. Description of the Problem

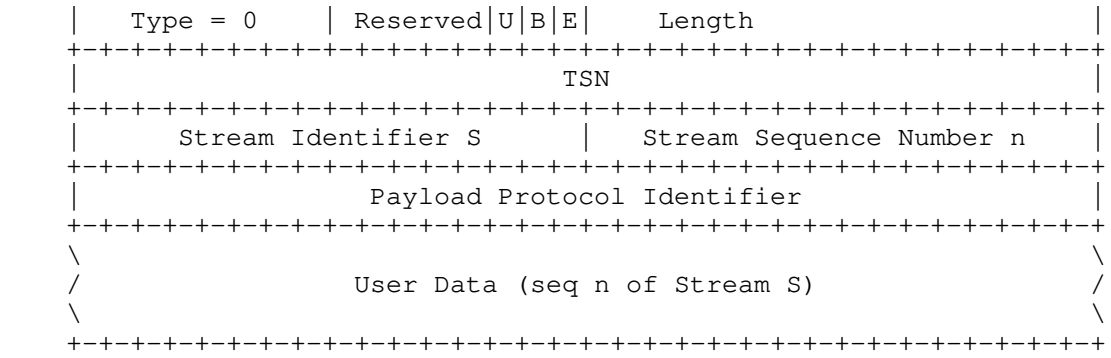
[RFC7053] updates [RFC4960] by adding the I bit to the DATA chunk. This should be integrated into the base specification.

3.45.2. Text Changes to the Document

Old text: (Section 3.3.1)

The following format MUST be used for the DATA chunk:

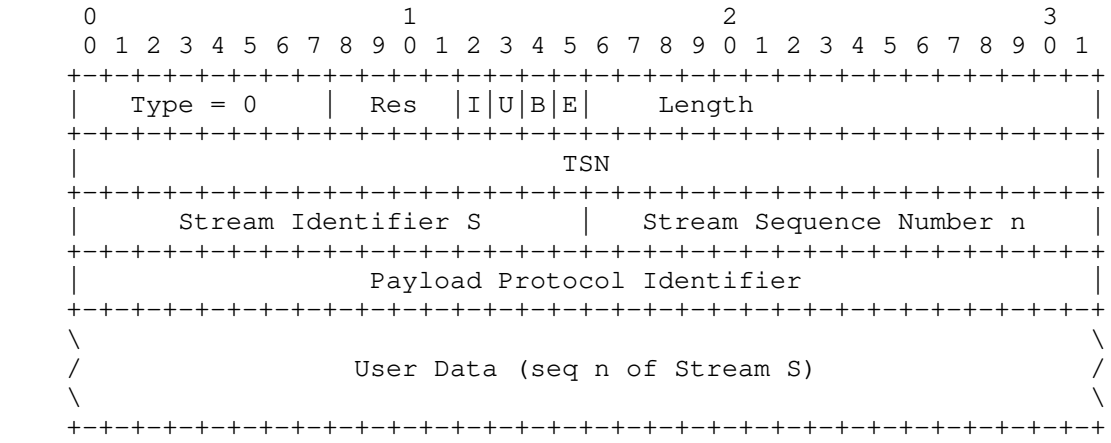




Reserved: 5 bits

Should be set to all '0's and ignored by the receiver.

New text: (Section 3.3.1)



Res: 4 bits

SHOULD be set to all '0's and ignored by the receiver.

I bit: 1 bit

The (I)mmediate Bit MAY be set by the sender, whenever the sender of a DATA chunk can benefit from the corresponding SACK chunk being sent back without delay. See [RFC7053] for a discussion about

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):

- The application requests to set the I bit of the last DATA chunk of a user message when providing the user message to the SCTP implementation (see Section 7).
- The sender is in the SHUTDOWN-PENDING state.
- The sending of a DATA chunk fills the congestion or receiver window.

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, 0x7EAEB2FAL,
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, 0x7EAEB2FAUL,
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, 0xCAA7A905UL, 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, Gorry Fairhurst, Benjamin Kaduk, Mirja Kuehlewind, Peter Lei, Gyula Marosi, Lionel Morand, Jeff Morriss, Karen E. E. Nielsen, Tom Petch, Kacheong Poon, Julien Pourtet, Irene Ruengeler, Michael Welzl, and Qiaobing Xie for their invaluable comments.

7. References

7.1. Normative References

[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: May 16, 2019

B. Briscoe
Independent

November 12, 2018

Propagating Explicit Congestion Notification Across IP Tunnel Headers Separated by a Shim
draft-ietf-tsvwg-rfc6040update-shim-07

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on May 16, 2019.

Table of Contents

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:

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

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

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

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

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

* if it is known that the tunnel egress does not support propagation of the ECN field (RFC 6040, RFC 4301 or the full functionality mode of RFC 3168)

* or if the behaviour of the egress is not known or an egress with unknown behaviour might be dynamically paired with the ingress.

* or if an IP header might be encapsulated within a non-IP header that the tunnel ingress is encapsulating, but the ingress does not inspect within the encapsulation.

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

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

* it MUST make propagation of the ECN field between inner and outer IP headers independent of any configuration of Diffserv codepoint propagation;

* it SHOULD be able to be configured to zero the outer ECN field.

"

There might be concern that the above "MUST" makes compliant equipment non-compliant at a stroke. However, any equipment that is still treating the former ToS octet (IPv4) or the former Traffic Class octet (IPv6) as a single 8-bit field is already non-compliant, and has been since 1998 when the upper 6 bits were separated off for the Diffserv field [RFC2474], [RFC3260]. For instance, copying the ECN field as a side-effect of copying the DSCP is a seriously unsafe bug that risks breaking the feedback loops that regulate load on a tunnel.

Permanently zeroing the outer ECN field is safe, but it is not sufficient to claim compliance with RFC 6040 because it does not meet the aim of introducing ECN support to tunnels (see Section 4.3 of [RFC6040]).

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

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

- PPTP (Point-to-Point Tunneling Protocol) [RFC2637];
- L2TP (Layer 2 Tunnelling Protocol), specifically L2TPv2 [RFC2661] and L2TPv3 [RFC3931], which not only includes all the L2-specific specializations of L2TP, but also derivatives such as the Keyed IPv6 Tunnel [RFC8159];
- GRE (Generic Routing Encapsulation) [RFC2784] and NVGRE (Network Virtualization using GRE) [RFC7637];
- GTP (GPRS Tunnelling Protocol), specifically GTPv1 [GTPv1], GTP v1 User Plane [GTPv1-U], GTP v2 Control Plane [GTPv2-C];
- Teredo [RFC4380];
- CAPWAP (Control And Provisioning of Wireless Access Points) [RFC5415];
- LISP (Locator/Identifier Separation Protocol) [RFC6830];
- AMT (Automatic Multicast Tunneling) [RFC7450];
- VXLAN (Virtual eXtensible Local Area Network) [RFC7348] and VXLAN-GPE [I-D.ietf-nvo3-vxlan-gpe];
- The Network Service Header (NSH [RFC8300]) for Service Function Chaining (SFC);
- Geneve [I-D.ietf-nvo3-geneve];
- GUE (Generic UDP Encapsulation) [I-D.ietf-intarea-gue];
- Direct tunnelling of an IP packet within a UDP/IP datagram (see Section 3.1.11 of [RFC8085]);
- TCP Encapsulation of IKE and IPsec Packets (see Section 12.5 of [RFC8229]).

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

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

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

NVGRE is not under the change control of the IETF, but it has been documented in an informational RFC [RFC7637]. NVGRE is a specific use-case of GRE (it re-purposes the key field from the initial specification of GRE [RFC1701] as a Virtual Subnet ID). Therefore the text that updates GRE in Section 5.1.2 below is also intended to update NVGRE.

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

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

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

VXLAN is not under the change control of the IETF but it has been documented in an informational RFC. In contrast, VXLAN-GPE (Generic Protocol Extension) is being documented under IETF change control. It is RECOMMENDED that VXLAN and VXLAN-GPE implementations comply

with RFC 6040 when the VXLAN header is inserted between (or removed from between) IP headers. The authors of any future update to these specifications are encouraged to add a requirement to comply with RFC 6040 as updated by the present specification.

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

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

Section 3.1.11 of the UDP usage guidelines [RFC8085] already explains that a tunnel that encapsulates an IP header directly within a UDP/IP datagram needs to follow RFC 6040 when propagating the ECN field between inner and outer IP headers. The requirements in Section 4 update RFC 6040 so, by reference, they automatically update RFC 8085 to add the important but previously unstated requirement that, if the UDP tunnel egress does not, or might not, support ECN propagation, a legacy UDP tunnel ingress has to clear the outer IP ECN field to 0b00, e.g. by configuration.

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

5.1. Specific Updates to Protocols under IETF Change Control

5.1.1. L2TP (v2 and v3) ECN Extension

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

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

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

Even though these shims are rather fat (particularly in the case of L2TPv3), they still fit the definition of a tightly coupled shim

header over an encapsulating header (Section 3.1), because all the headers encapsulating the L2 header are added (or removed) together. L2TPv2 and L2TPv3 are therefore within the scope of RFC 6040, as updated by Section 3 above.

L2TP maintainers are RECOMMENDED to implement the ECN extension to L2TPv2 and L2TPv3 defined in Section 5.1.1.2 below, in order to provide the benefits of ECN [RFC8087], whenever a node within an L2TP tunnel becomes the bottleneck for an end-to-end traffic flow.

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

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

> The operator of an LCCE that does not support the ECN Extension in Section 5.1.1.2 of RFCXXXX MUST follow the configuration requirements in Section 4 of RFCXXXX to ensure it clears the outer IP ECN field to 0b00 when the outer PSN header is IP (v4 or v6). {RFCXXXX refers to the present document so it will need to be inserted by the RFC Editor}

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

5.1.1.2. ECN Extension for L2TP (v2 or v3)

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

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

An LCCE can determine the remote LCCE's support for ECN either statically (by configuration) or by dynamic discovery during setup of each control connection between the LCCEs, using the Capability AVP defined in Section 5.1.1.2.1 below.

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

5.1.1.2.1. LCCE Capability AVP for ECN Capability Negotiation

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

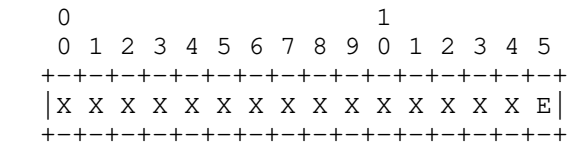


Figure 1: Value Field for the LCCE Capability Attribute

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 Capabiliy AVP is present, it indicates that the sender does not support ECN propagation. All the other bits are reserved. They MUST be cleared to zero when sent and ignored when received or forwarded.

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

If, on the other hand, the tunnel terminator does not support ECN it will ignore the ECN flag in the LCCE Capability AVP and send an SCCRP

to the tunnel initiator without a Capability AVP (or with a Capability AVP but with the ECN Capability flag cleared to zero). The tunnel initiator interprets the absence of the ECN Capability flag in the SCCRP as an indication that the tunnel terminator is incapable of supporting ECN. When encapsulating data packets for any sessions created by that control connection, the tunnel initiator will then use the compatibility mode of RFC 6040 to clear the ECN field of the outer IP header to 0b00.

If the tunnel terminator does not support this ECN extension, the network operator is still expected to configure it to comply with the safety provisions set out in Section 5.1.1.1 above, when it acts as an ingress LCCE.

5.1.2. GRE

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

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

GRE itself does not support dynamic set-up and configuration of tunnels. However, control plane protocols such as Mobile IPv4 (MIP4) [RFC5944], Mobile IPv6 (MIP6) [RFC6275], Proxy Mobile IP (PMIP) [RFC5845] and IKEv2 [RFC5996] are sometimes used to set up GRE tunnels dynamically.

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

Therefore, If the outer delivery protocol is IP (v4 or v6) the operator is obliged to follow the safe configuration requirements in Section 4 above. Section 5.1.2.1 below updates the base GRE specification with this requirement, to emphasize its importance.

Where the delivery protocol is some protocol other than IP that supports ECN, the appropriate ECN propagation specification will need to be followed, e.g Explicit Congestion Marking in MPLS [RFC5129].

Where no specification exists for ECN propagation by a particular PSN, [I-D.ietf-tsvwg-ecn-encap-guidelines] gives more general guidance on how to propagate ECN to and from protocols that encapsulate IP.

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

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

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

5.1.3. Teredo

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

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

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

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

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

> "5.1.3 Safe 'Non-ECN' Teredo Encapsulation
>
> A Teredo tunnel ingress implementation that does not support ECN propagation as defined in RFC 6040 or one of its compatible

predecessors (RFC 4301 or the full functionality mode of RFC 3168) MUST zero the ECN field in the outer IPv6 header."

5.1.4. AMT

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

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

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

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

An AMT tunnel is created dynamically (not manually), so the relay will need to determine the remote gateway's support for ECN using the ECN capability declaration defined in Section 5.1.4.2 below.

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

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

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

5.1.4.2. ECN Capability Declaration of an AMT Gateway

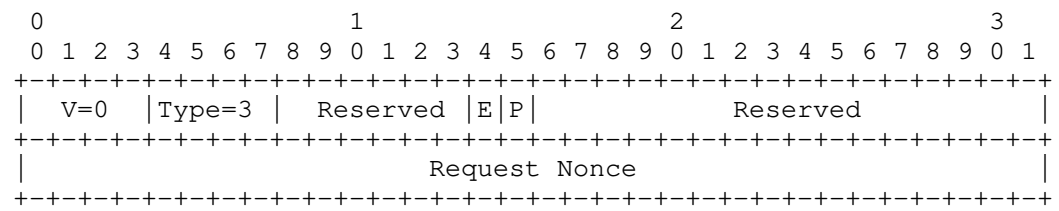


Figure 2: Updated AMT Request Message Format

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

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

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

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

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

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

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

- 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 5.1.4.1 above.

6. IANA Considerations

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

Attribute Type	Description	Reference
ZZ	ECN Capability	RFCXXXX

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

7. Security Considerations

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

8. Comments Solicited

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

9. Acknowledgements

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

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

10. References

10.1. Normative References

[I-D.ietf-tsvwg-ecn-encap-guidelines]
Briscoe, B., Kaippallimalil, J., and P. Thaler, "Guidelines for Adding Congestion Notification to Protocols that Encapsulate IP", draft-ietf-tsvwg-ecn-encap-guidelines-11 (work in progress), Novemeber 2018.

[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>.

10.2. 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.eastlake-sfc-nsh-ecn-support] Eastlake, D., Briscoe, B., and A. Malis, "Explicit Congestion Notification (ECN) and Congestion Feedback Using the Network Service Header (NSH)", draft-eastlake-sfc-nsh-ecn-support-02 (work in progress), October 2018.

[I-D.ietf-intarea-gue] Herbert, T. and O. Zia, "Generic UDP Encapsulation", draft-ietf-intarea-gue-06 (work in progress), August 2018.

[I-D.ietf-nvo3-geneve] Gross, J., Ganga, I., and T. Sridhar, "Geneve: Generic Network Virtualization Encapsulation", draft-ietf-nvo3-geneve-08 (work in progress), October 2018.

[I-D.ietf-nvo3-vxlan-gpe] Maino, F., Kreeger, L., and U. Elzur, "Generic Protocol Extension for VXLAN", draft-ietf-nvo3-vxlan-gpe-06 (work in progress), April 2018.

[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>.

[RFC5996] Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen, "Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 5996, DOI 10.17487/RFC5996, September 2010, <https://www.rfc-editor.org/info/rfc5996>.

[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>.

[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>.

Author's Address

Bob Briscoe
Independent
UK

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

TSVWG
Internet-Draft
Intended status: Standards Track
Expires: August 15, 2019

V. Roca
B. Teibi
INRIA

February 11, 2019

Sliding Window Random Linear Code (RLC) Forward Erasure Correction (FEC) Schemes for FECFRAME
draft-ietf-tsvwg-rlc-fec-scheme-12

Abstract

This document describes two fully-specified Forward Erasure Correction (FEC) Schemes for Sliding Window Random Linear Codes (RLC), one for RLC over the Galois Field (A.K.A. Finite Field) GF(2), a second one for RLC over the Galois Field GF(2^^8), each time with the possibility of controlling the code density. They can protect arbitrary media streams along the lines defined by FECFRAME extended to sliding window FEC codes, as defined in [fecframe-ext]. These sliding window FEC codes rely on an encoding window that slides over the source symbols, generating new repair symbols whenever needed. Compared to block FEC codes, these sliding window FEC codes offer key advantages with real-time flows in terms of reduced FEC-related latency while often providing improved packet erasure recovery capabilities.

Status of This Memo

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

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

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

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

Table of Contents

1. Introduction

Application-Level Forward Erasure Correction (AL-FEC) codes, or simply FEC codes, are a key element of communication systems. They are used to recover from packet losses (or erasures) during content delivery sessions to a potentially large number of receivers (multicast/broadcast transmissions). This is the case with the FLUTE/ALC protocol [RFC6726] when used for reliable file transfers over lossy networks, and the FECFRAME protocol when used for reliable continuous media transfers over lossy networks.

The present document only focuses on the FECFRAME protocol, used in multicast/broadcast delivery mode, in particular for contents that feature stringent real-time constraints: each source packet has a maximum validity period after which it will not be considered by the destination application.

1.1. Limits of Block Codes with Real-Time Flows

With FECFRAME, there is a single FEC encoding point (either a end-host/server (source) or a middlebox) and a single FEC decoding point per receiver (either a end-host (receiver) or middlebox). In this context, currently standardized AL-FEC codes for FECFRAME like Reed-Solomon [RFC6865], LDPC-Staircase [RFC6816], or Raptor/RaptorQ, are all linear block codes: they require the data flow to be segmented into blocks of a predefined maximum size.

To define this block size, it is required to find an appropriate balance between robustness and decoding latency: the larger the block size, the higher the robustness (e.g., in case of long packet erasure bursts), but also the higher the maximum decoding latency (i.e., the maximum time required to recover a lost (erased) packet thanks to FEC protection). Therefore, with a multicast/broadcast session where different receivers experience different packet loss rates, the block size should be chosen by considering the worst communication conditions one wants to support, but without exceeding the desired maximum decoding latency. This choice then impacts the FEC-related latency of all receivers, even those experiencing a good communication quality, since no FEC encoding can happen until all the source data of the block is available at the sender, which directly depends on the block size.

1.2. Lower Latency and Better Protection of Real-Time Flows with the Sliding Window RLC Codes

This document introduces two fully-specified FEC Schemes that do not follow the block code approach: the Sliding Window Random Linear Codes (RLC) over either Galois Fields (A.K.A. Finite Fields) GF(2) (the "binary case") or GF(2^^8), each time with the possibility of controlling the code density. These FEC Schemes are used to protect arbitrary media streams along the lines defined by FECFRAME extended to sliding window FEC codes [fecframe-ext]. These FEC Schemes, and more generally Sliding Window FEC codes, are recommended for instance, with media that feature real-time constraints sent within a multicast/broadcast session [Roca17].

The RLC codes belong to the broad class of sliding-window AL-FEC codes (A.K.A. convolutional codes) [RFC8406]. The encoding process is based on an encoding window that slides over the set of source packets (in fact source symbols as we will see in Section 3.2), this window being either of fixed size or variable size (A.K.A. an elastic window). Repair symbols are generated on-the-fly, by computing a random linear combination of the source symbols present in the current encoding window, and passed to the transport layer.

At the receiver, a linear system is managed from the set of received source and repair packets. New variables (representing source symbols) and equations (representing the linear combination carried by each repair symbol received) are added upon receiving new packets. Variables and the equations they are involved in are removed when they are too old with respect to their validity period (real-time constraints) . Lost source symbols are then recovered thanks to this linear system whenever its rank permits to solve it (at least partially).

The protection of any multicast/broadcast session needs to be dimensioned by considering the worst communication conditions one wants to support. This is also true with RLC (more generally any sliding window) code. However, the receivers experiencing a good to medium communication quality will observe a reduced FEC-related latency compared to block codes [Roca17] since an isolated lost source packet is quickly recovered with the following repair packet. On the opposite, with a block code, recovering an isolated lost source packet always requires waiting for the first repair packet to arrive after the end of the block. Additionally, under certain situations (e.g., with a limited FEC-related latency budget and with constant bitrate transmissions after FECFRAME encoding), sliding window codes can more efficiently achieve a target transmission quality (e.g., measured by the residual loss after FEC decoding) by sending fewer repair packets (i.e., higher code rate) than block codes.

1.3. Small Transmission Overheads with the Sliding Window RLC FEC Scheme

The Sliding Window RLC FEC Scheme is designed to limit the packet header overhead. The main requirement is that each repair packet header must enable a receiver to reconstruct the set of source symbols plus the associated coefficients used during the encoding process. In order to minimize packet overhead, the set of source symbols in the encoding window as well as the set of coefficients over GF(2^^m) (where m is 1 or 8, depending on the FEC Scheme) used in the linear combination are not individually listed in the repair packet header. Instead, each FEC Repair Packet header contains:

- the Encoding Symbol Identifier (ESI) of the first source symbol in the encoding window as well as the number of symbols (since this number may vary with a variable size, elastic window). These two pieces of information enable each receiver to reconstruct the set of source symbols considered during encoding, the only constraint being that there cannot be any gap;
- the seed and density threshold parameters used by a coding coefficients generation function (Section 3.6). These two pieces

of information enable each receiver to generate the same set of coding coefficients over GF(2^^m) as the sender;

Therefore, no matter the number of source symbols present in the encoding window, each FEC Repair Packet features a fixed 64-bit long header, called Repair FEC Payload ID (Figure 8). Similarly, each FEC Source Packet features a fixed 32-bit long trailer, called Explicit Source FEC Payload ID (Figure 6), that contains the ESI of the first source symbol (Section 3.2).

1.4. Document Organization

This fully-specified FEC Scheme follows the structure required by [RFC6363], section 5.6. "FEC Scheme Requirements", namely:

3. Procedures: This section describes procedures specific to this FEC Scheme, namely: RLC parameters derivation, ADUI and source symbols mapping, pseudo-random number generator, and coding coefficients generation function;
4. Formats and Codes: This section defines the Source FEC Payload ID and Repair FEC Payload ID formats, carrying the signalling information associated to each source or repair symbol. It also defines the FEC Framework Configuration Information (FFCI) carrying signalling information for the session;
5. FEC Code Specification: Finally this section provides the code specification.

2. Definitions and Abbreviations

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

This document uses the following definitions and abbreviations:

a^^b a to the power of b
GF(q) denotes a finite field (also known as the Galois Field) with q elements. We assume that q = 2^^m in this document
m defines the length of the elements in the finite field, in bits. In this document, m is equal to 1 or 8
ADU: Application Data Unit
ADUI: Application Data Unit Information (includes the F, L and padding fields in addition to the ADU)
E: size of an encoding symbol (i.e., source or repair symbol), assumed fixed (in bytes)

br_in: transmission bitrate at the input of the FECFRAME sender, assumed fixed (in bits/s)
br_out: transmission bitrate at the output of the FECFRAME sender, assumed fixed (in bits/s)
max_lat: maximum FEC-related latency within FECFRAME (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 instance, at session start, upon receiving new source ADUs, the ew_size progressively increases until it reaches its maximum value, ew_max_size. We have:

0 < ew_size <= 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 < cr <= 1. This is an input parameter that enables a FECFRAME sender to derive other internal parameters, as explained below. However, there is no need to communicate the cr parameter per see (it’s not required to process a repair symbol at a receiver). This code rate parameter can be static. However, in specific use-cases (e.g., with unicast transmissions in presence of a feedback mechanism that estimates the communication quality, out of scope of FECFRAME), the code rate may be adjusted dynamically.

Appendix C proposes non normative technics to derive those parameters, depending on the use-case specificities.

3.2. ADU, ADUI and Source Symbols Mappings

At a sender, an ADU coming from the application is not directly mapped to source symbols. When multiple source flows (e.g., media streams) are mapped onto the same FECFRAME instance, each flow is assigned its own Flow ID value (see below). This Flow ID is then prepended to each ADU before FEC encoding. This way, FEC decoding at a receiver also recovers this Flow ID and the recovered ADU can be assigned to the right source flow (note that the 5-tuple used to identify the right source flow of a received ADU is absent with a recovered ADU since it is not FEC protected).

Additionally, since ADUs are of variable size, padding is needed so that each ADU (with its flow identifier) contribute to an integral number of source symbols. This requires adding the original ADU length to each ADU before doing FEC encoding. Because of these requirements, an intermediate format, the ADUI, or ADU Information, is considered [RFC6363].

For each incoming ADU, an ADUI MUST created as follows. First of all, 3 bytes are prepended (Figure 1):

Flow ID (F) (8-bit field): this unsigned byte contains the integer identifier associated to the source ADU flow to which this ADU belongs. It is assumed that a single byte is sufficient, which implies that no more than 256 flows will be protected by a single FECFRAME session instance.

Length (L) (16-bit field): this unsigned integer contains the length of this ADU, in network byte order (i.e., big endian). This length is for the ADU itself and does not include the F, L, or Pad fields.

Then, zero padding is added to the ADU if needed:

Padding (Pad) (variable size field): this field contains zero padding to align the F, L, ADU and padding up to a size that is multiple of E bytes (i.e., the source and repair symbol length).

The data unit resulting from the ADU and the F, L, and Pad fields is called ADUI. Since ADUs can have different sizes, this is also the case for ADUIs. However, an ADUI always contributes to an integral number of source symbols.

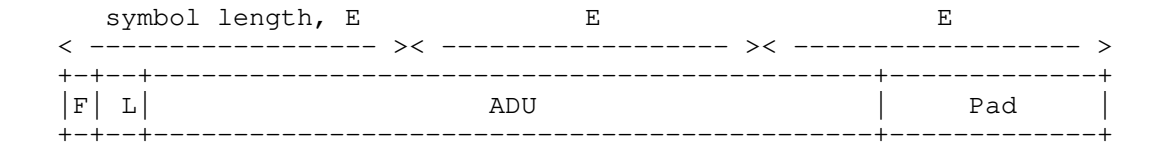


Figure 1: ADUI Creation example (here 3 source symbols are created for this ADUI).

Note that neither the initial 3 bytes nor the optional padding are sent over the network. However, they are considered during FEC encoding, and a receiver who lost a certain FEC Source Packet (e.g., the UDP datagram containing this FEC Source Packet when UDP is used as the transport protocol) will be able to recover the ADUI if FEC decoding succeeds. Thanks to the initial 3 bytes, this receiver will get rid of the padding (if any) and identify the corresponding ADU flow.

3.3. Encoding Window Management

Source symbols and the corresponding ADUs are removed from the encoding window:

- when the sliding encoding window has reached its maximum size, ew_max_size. In that case the oldest symbol MUST be removed before adding a new symbol, so that the current encoding window size always remains inferior or equal to the maximum size: ew_size <= ew_max_size;
- when an ADU has reached its maximum validity duration in case of a real-time flow. When this happens, all source symbols corresponding to the ADUI that expired SHOULD be removed from the encoding window;

Source symbols are added to the sliding encoding window each time a new ADU arrives, once the ADU-to-source symbols mapping has been performed (Section 3.2). The current size of the encoding window, ew_size, is updated after adding new source symbols. This process may require to remove old source symbols so that: ew_size <= ew_max_size.

Note that a FEC codec may feature practical limits in the number of source symbols in the encoding window (e.g., for computational complexity reasons). This factor may further limit the ew_max_size value, in addition to the maximum FEC-related latency budget (Section 3.1).

3.4. Source Symbol Identification

Each source symbol is identified by an Encoding Symbol ID (ESI), an unsigned integer. The ESI of source symbols MUST start with value 0 for the first source symbol and MUST be managed sequentially. Wrapping to zero happens after reaching the maximum value made possible by the ESI field size (this maximum value is FEC Scheme dependant, for instance, 2^32-1 with FEC Schemes XXX and YYY).

No such consideration applies to repair symbols.

3.5. Pseudo-Random Number Generator (PRNG)

In order to compute coding coefficients (see Section 3.6), the RLC FEC Schemes rely on the TinyMT32 PRNG defined in [tinymt32] with two additional functions defined in this section.

This PRNG MUST first be initialized with a 32-bit unsigned integer, used as a seed, with:

```
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 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]. 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 fulfill three validation criteria: the one described in [tinymt32] (for the TinyMT32 32-bit unsigned integer generator), and the two others detailed in Appendix A (for the mapping to 4-bit and 8-bit intervals). Because of the way the mapping functions work, it is unlikely that an implementation that fulfills the first criterion fails to fulfill the two others.

3.6. Coding Coefficients Generation Function

The coding coefficients, used during the encoding process, are generated at the RLC encoder by the generate_coding_coefficients() function each time a new repair symbol needs to be produced. The fraction of coefficients that are non zero (i.e., the density) is controlled by the DT (Density Threshold) parameter. DT has values between 0 (the minimum value) and 15 (the maximum value), and the average probability of having a non zero coefficient equals (DT + 1)

/ 16. In particular, when DT equals 15 the function guaranties that all coefficients are non zero (i.e., maximum density).

These considerations apply to both the RLC over GF(2) and RLC over GF(2^^8), the only difference being the value of the m parameter. With the RLC over GF(2) FEC Scheme (Section 5), m is equal to 1. With RLC over GF(2^^8) FEC Scheme (Section 4), m is equal to 8.

<CODE BEGINS>

```
/*
 * 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 table. This value is
 *                    equal to the current encoding window size.
 * (in) dt            integer between 0 and 15 (inclusive) that
 *                    controls the density. With value 15, all
 *                    coefficients are guaranteed to be non zero
 *                    (i.e. equal to 1 with GF(2) and equal to a
 *                    value in {1,... 255} with GF(2^^8)), otherwise
 *                    a fraction of them will be 0.
 * (in) m             Finite Field GF(2^^m) parameter. In this
 *                    document only values 1 and 8 are considered.
 * (out)              returns 0 in case of success, an error code
 *                    different than 0 otherwise.
 */
int generate_coding_coefficients (uint16_t  repair_key,
                                  uint8_t   cc_tab[],
                                  uint16_t  cc_nb,
                                  uint8_t   dt,
                                  uint8_t   m)
{
    uint32_t      i;
    tinymt32_t    s;    /* PRNG internal state */

    if (dt > 15) {
        return -1; /* error, bad dt parameter */
    }
    switch (m) {
    case 1:
        if (dt == 15) {
```

```
            /* all coefficients are 1 */
            memset(cc_tab, 1, cc_nb);
        } else {
            /* here coefficients are either 0 or 1 */
            tinymt32_init(&s, repair_key);
            for (i = 0 ; i < cc_nb ; i++) {
                cc_tab[i] = (tinymt32_rand16(&s) <= dt) ? 1 : 0;
            }
        }
        break;

    case 8:
        tinymt32_init(&s, repair_key);
        if (dt == 15) {
            /* coefficient 0 is avoided here in order to include
             * all the source symbols */
            for (i = 0 ; i < cc_nb ; i++) {
                do {
                    cc_tab[i] = (uint8_t) tinymt32_rand256(&s);
                } while (cc_tab[i] == 0);
            }
        } else {
            /* here a certain number of coefficients should be 0 */
            for (i = 0 ; i < cc_nb ; i++) {
                if (tinymt32_rand16(&s) <= dt) {
                    do {
                        cc_tab[i] = (uint8_t) tinymt32_rand256(&s);
                    } while (cc_tab[i] == 0);
                } else {
                    cc_tab[i] = 0;
                }
            }
        }
        break;

    default:
        return -2; /* error, bad parameter m */
    }
    return 0 /* success */
}
<CODE ENDS>
```

Figure 3: Coding Coefficients Generation Function Reference Implementation

3.7. Finite Fields Operations

3.7.1. Finite Field Definitions

The two RLC FEC Schemes specified in this document reuse the Finite Fields defined in [RFC5510], section 8.1. More specifically, the elements of the field GF(2^^m) are represented by polynomials with binary coefficients (i.e., over GF(2)) and degree lower or equal to m-1. The addition between two elements is defined as the addition of binary polynomials in GF(2), which is equivalent to a bitwise XOR operation on the binary representation of these elements.

With GF(2^^8), multiplication between two elements is the multiplication modulo a given irreducible polynomial of degree 8. The following irreducible polynomial MUST be used for GF(2^^8):

```
x^^8 + x^^4 + x^^3 + x^^2 + 1
```

With GF(2), multiplication corresponds to a logical AND operation.

3.7.2. 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:

```
repair[i] = cc_tab[0] * src_0[i] XOR cc_tab[1] * src_1[i] XOR ...
XOR cc_tab[ew_size - 1] * 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 FCCI needs to be shared (e.g., using SDP) between the FECFRAME sender and receiver instances in order to synchronize them. It includes a FEC Encoding ID, mandatory for any FEC Scheme specification, plus scheme-specific elements.

4.1.1.1. FEC Encoding ID

- FEC Encoding ID: the value assigned to this fully specified FEC Scheme MUST be XXXX, as assigned by IANA (Section 10).

When SDP is used to communicate the 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.

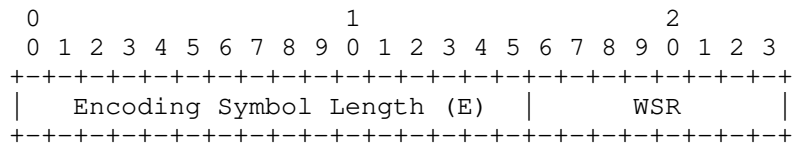


Figure 4: FSSI Encoding Format

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.

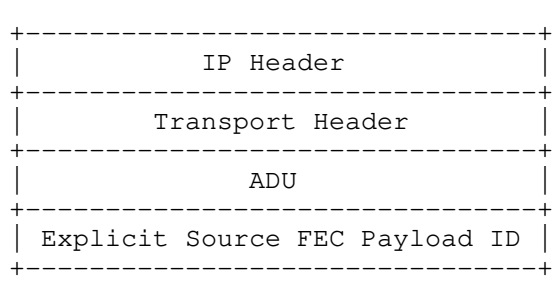


Figure 5: Structure of an FEC Source Packet with the Explicit Source FEC Payload ID

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.

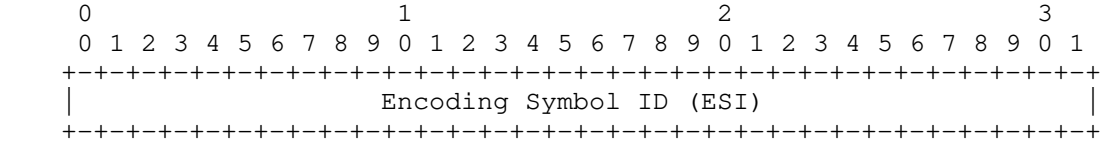


Figure 6: Source FEC Payload ID Encoding Format

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.

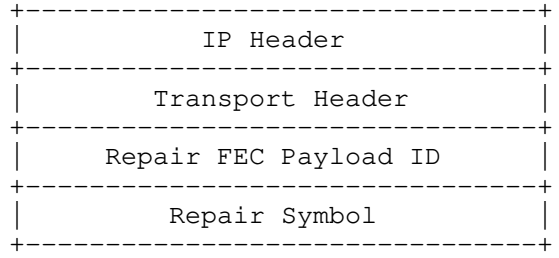


Figure 7: Structure of an FEC Repair Packet with the Repair FEC Payload ID

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;

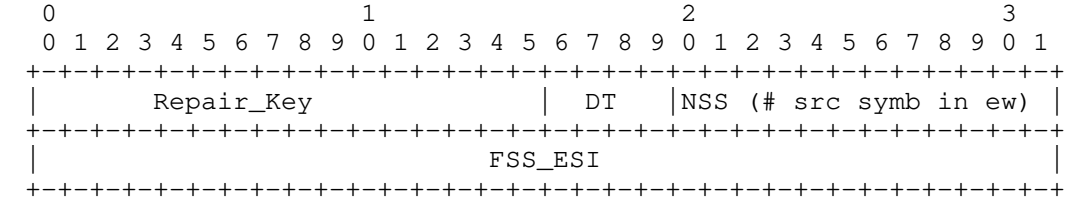


Figure 8: Repair FEC Payload ID Encoding Format

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

- 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.

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 Section 3.6), using the repair key and encoding window descriptions carried in the Repair FEC Payload ID. Whenever possible (i.e., when a sub-system covering one or more lost source symbols is of full rank), decoding is performed in order to recover lost source symbols. Gaussian elimination is one possible algorithm to solve this linear system. Each time an ADUI can be totally recovered, padding is removed (thanks to the Length field, L, of the ADUI) and the ADU is assigned to the corresponding application flow (thanks to the Flow ID field, F, of the ADUI). This ADU is finally passed to the corresponding upper application. Received FEC

Source Packets, containing an ADU, MAY be passed to the application either immediately or after some time to guaranty an ordered delivery to the application. This document does not mandate any approach as this is an operational and management decision.

With real-time flows, a lost ADU that is decoded after the maximum latency or an ADU received after this delay has no value to the application. This raises the question of deciding whether or not an ADU is late. This decision MAY be taken within the FECFRAME receiver (e.g., using the decoding window, see Section 3.1) or within the application (e.g., using RTP timestamps within the ADU). Deciding which option to follow and whether or not to pass all ADUs, including those assumed late, to the application are operational decisions that depend on the application and are therefore out of scope of this document. Additionally, Appendix D discusses a backward compatible optimization whereby late source symbols MAY still be used within the FECFRAME receiver in order to improve transmission robustness.

7. Implementation Status

Editor's notes: RFC Editor, please remove this section motivated by RFC 6982 before publishing the RFC. Thanks.

An implementation of the Sliding Window RLC FEC Scheme for FECFRAME exists:

- Organisation: Inria
- Description: This is an implementation of the Sliding Window RLC FEC Scheme limited to GF(2^^8). It relies on a modified version of our OpenFEC (http://openfec.org) FEC code library. It is integrated in our FECFRAME software (see [fecframe-ext]).
- Maturity: prototype.
- Coverage: this software complies with the Sliding Window RLC FEC Scheme.
- Licensing: proprietary.
- Contact: vincent.roca@inria.fr

8. Security Considerations

The FEC Framework document [RFC6363] provides a fairly comprehensive analysis of security considerations applicable to FEC Schemes. Therefore, the present section follows the security considerations section of [RFC6363] and only discusses specific topics.

8.1. Attacks Against the Data Flow

8.1.1. Access to Confidential Content

The Sliding Window RLC FEC Scheme specified in this document does not change the recommendations of [RFC6363]. To summarize, if confidentiality is a concern, it is RECOMMENDED that one of the solutions mentioned in [RFC6363] is used with special considerations to the way this solution is applied (e.g., is encryption applied before or after FEC protection, within the end-system or in a middlebox), to the operational constraints (e.g., performing FEC decoding in a protected environment may be complicated or even impossible) and to the threat model.

8.1.2. Content Corruption

The Sliding Window RLC FEC Scheme specified in this document does not change the recommendations of [RFC6363]. To summarize, it is RECOMMENDED that one of the solutions mentioned in [RFC6363] is used on both the FEC Source and Repair Packets.

8.2. Attacks Against the FEC Parameters

The FEC Scheme specified in this document defines parameters that can be the basis of attacks. More specifically, the following parameters of the FFCI may be modified by an attacker who targets receivers (Section 4.1.1.2):

- FEC Encoding ID: changing this parameter leads a receiver to consider a different FEC Scheme. The consequences are severe, the format of the Explicit Source FEC Payload ID and Repair FEC Payload ID of received packets will probably differ, leading to various malfunctions. Even if the original and modified FEC Schemes share the same format, FEC decoding will either fail or lead to corrupted decoded symbols. This will happen if an attacker turns value YYYY (i.e., RLC over GF(2)) to value XXXX (RLC over GF(2^^8)), an additional consequence being a higher processing overhead at the receiver. In any case, the attack results in a form of Denial of Service (DoS) or corrupted content.
- Encoding symbol length (E): setting this E parameter to a different value will confuse a receiver. If the size of a received FEC Repair Packet is no longer multiple of the modified E value, a receiver quickly detects a problem and SHOULD reject the packet. If the new E value is a sub-multiple of the original E value (e.g., half the original value), then receivers may not detect the problem immediately. For instance, a receiver may think that a received FEC Repair Packet contains more repair symbols (e.g., twice as many if E is reduced by half), leading to

malfunctions whose nature depends on implementation details. Here also, the attack always results in a form of DoS or corrupted content.

It is therefore RECOMMENDED that security measures be taken to guarantee the FFCI integrity, as specified in [RFC6363]. How to achieve this depends on the way the FFCI is communicated from the sender to the receiver, which is not specified in this document.

Similarly, attacks are possible against the Explicit Source FEC Payload ID and Repair FEC Payload ID. More specifically, in case of a FEC Source Packet, the following value can be modified by an attacker who targets receivers:

- Encoding Symbol ID (ESI): changing the ESI leads a receiver to consider a wrong ADU, resulting in severe consequences, including corrupted content passed to the receiving application;

And in case of a FEC Repair Packet:

- Repair Key: changing this value leads a receiver to generate a wrong coding coefficient sequence, and therefore any source symbol decoded using the repair symbols contained in this packet will be corrupted;
- DT: changing this value also leads a receiver to generate a wrong coding coefficient sequence, and therefore any source symbol decoded using the repair symbols contained in this packet will be corrupted. In addition, if the DT value is significantly increased, it will generate a higher processing overhead at a receiver. In case of very large encoding windows, this may impact the terminal performance;
- NSS: changing this value leads a receiver to consider a different set of source symbols, and therefore any source symbol decoded using the repair symbols contained in this packet will be corrupted. In addition, if the NSS value is significantly increased, it will generate a higher processing overhead at a receiver, which may impact the terminal performance;
- FSS_ESI: changing this value also leads a receiver to consider a different set of source symbols and therefore any source symbol decoded using the repair symbols contained in this packet will be corrupted.

It is therefore RECOMMENDED that security measures are taken to guarantee the FEC Source and Repair Packets as stated in [RFC6363].

8.3. When Several Source Flows are to be Protected Together

The Sliding Window RLC FEC Scheme specified in this document does not change the recommendations of [RFC6363].

8.4. Baseline Secure FEC Framework Operation

The Sliding Window RLC FEC Scheme specified in this document does not change the recommendations of [RFC6363] concerning the use of the IPsec/ESP security protocol as a mandatory to implement (but not mandatory to use) security scheme. This is well suited to situations where the only insecure domain is the one over which the FEC Framework operates.

8.5. Additional Security Considerations for Numerical Computations

In addition to the above security considerations, inherited from [RFC6363], the present document introduces several formulae, in particular in Appendix C.1. It is RECOMMENDED to check that the computed values stay within reasonable bounds since numerical overflows, caused by an erroneous implementation or an erroneous input value, may lead to hazardous behaviours. However, what "reasonable bounds" means is use-case and implementation dependent and is not detailed in this document.

Appendix C.2 also mentions the possibility of "using the timestamp field of an RTP packet header" when applicable. A malicious attacker may deliberately corrupt this header field in order to trigger hazardous behaviours at a FECFRAME receiver. Protection against this type of content corruption can be addressed with the above recommendations on a baseline secure operation. In addition, it is also RECOMMENDED to check that the timestamp value be within reasonable bounds.

9. Operations and Management Considerations

The FEC Framework document [RFC6363] provides a fairly comprehensive analysis of operations and management considerations applicable to FEC Schemes. Therefore, the present section only discusses specific topics.

9.1. Operational Recommendations: Finite Field GF(2) Versus GF(2^^8)

The present document specifies two FEC Schemes that differ on the Finite Field used for the coding coefficients. It is expected that the RLC over GF(2^^8) FEC Scheme will be mostly used since it warrants a higher packet loss protection. In case of small encoding windows, the associated processing overhead is not an issue (e.g., we

measured decoding speeds between 745 Mbps and 2.8 Gbps on an ARM Cortex-A15 embedded board in [Roca17] for an encoding window of size 18 or 23 symbols). Of course the CPU overhead will increase with the encoding window size, because more operations in the GF(2^^8) finite field will be needed.

The RLC over GF(2) FEC Scheme offers an alternative. In that case operations symbols can be directly XOR-ed together which warrants high bitrate encoding and decoding operations, and can be an advantage with large encoding windows. However, packet loss protection is significantly reduced by using this FEC Scheme.

9.2. Operational Recommendations: Coding Coefficients Density Threshold

In addition to the choice of the Finite Field, the two FEC Schemes define a coding coefficient density threshold (DT) parameter. This parameter enables a sender to control the code density, i.e., the proportion of coefficients that are non zero on average. With RLC over GF(2^^8), it is usually appropriate that small encoding windows be associated to a density threshold equal to 15, the maximum value, in order to warrant a high loss protection.

On the opposite, with larger encoding windows, it is usually appropriate that the density threshold be reduced. With large encoding windows, an alternative can be to use RLC over GF(2) and a density threshold equal to 7 (i.e., an average density equal to 1/2) or smaller.

Note that using a density threshold equal to 15 with RLC over GF(2) is equivalent to using an XOR code that computes the XOR sum of all the source symbols in the encoding window. In that case: (1) only a single repair symbol can be produced for any encoding window, and (2) the repair_key parameter becomes useless (the coding coefficients generation function does not rely on the PRNG).

10. IANA Considerations

This document registers two values in the "FEC Framework (FECFRAME) FEC Encoding IDs" registry [RFC6363] as follows:

- YYYY refers to the Sliding Window Random Linear Codes (RLC) over GF(2) FEC Scheme for Arbitrary Packet Flows, as defined in Section 5 of this document.
- XXXX refers to the Sliding Window Random Linear Codes (RLC) over GF(2^^8) FEC Scheme for Arbitrary Packet Flows, as defined in Section 4 of this document.

11. Acknowledgments

The authors would like to thank the three TSVWG chairs, Wesley Eddy, our shepherd, David Black and Gorry Fairhurst, as well as Spencer Dawkins, our responsible AD, and all those who provided comments, namely (alphabetical order) Alan DeKok, Jonathan Detchart, Russ Housley, Emmanuel Lochin, Marie-Jose Montpetit, and Greg Skinner. Last but not least, the authors are really grateful to the IESG members, in particular Benjamin Kaduk, Mirja Kuhlewind, Eric Rescorla, and Adam Roach for their highly valuable feedbacks that greatly contributed to improve this specification.

12. References

12.1. Normative References

[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>.

[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.

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 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.

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 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), and perform statistics on the number of occurrences of each of the 16 possible values across all sequences.

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 value would appear exactly 1 / 16 * 100 = 6.25% of times.

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 tests, 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 possible value 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:

```
dw_max_size = (max_lat * 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 compromized by choosing a too large value.

In a second configuration, the FECFRAME sender generates a fixed bitrate flow, equal to the CBR communication path bitrate equal to br_out (in bits/s), and this value is known by the FECFRAME sender,

as in [Roca17]. The maximum source flow bitrate needs to be such that, with the added repair flow overhead, the total transmission bitrate remains inferior or equal to br_out. We have:

dw_max_size = (max_lat * br_out * 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:

ew_max_size = dw_max_size * WSR / 255

In line with [Roca17], WSR = 191 is considered as a reasonable value (the resulting encoding to decoding window size ratio is then close to 0.75), but other values between 1 and 255 inclusive are possible, depending on the use-case.

The dw_max_size is computed by a FECFRAME sender but not explicitly communicated to a FECFRAME receiver. However, a FECFRAME receiver can easily evaluate the ew_max_size by observing the maximum Number of Source Symbols (NSS) value contained in the Repair FEC Payload ID of received FEC Repair Packets (Section 4.1.3). A receiver can then easily compute dw_max_size:

dw_max_size = max_NSS_observed * 255 / WSR

A receiver can then chose an appropriate linear system maximum size:

ls_max_size >= 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:

```
max_lat_for_encoding = max_lat * 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.

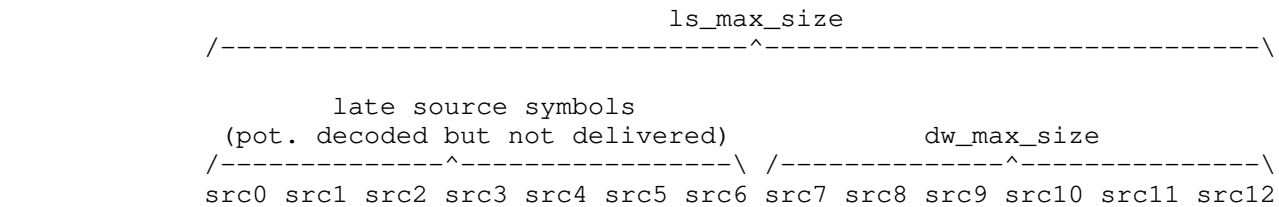


Figure 13: Relationship between parameters to decode beyond maximum latency.

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.

Table of Contents

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.

]

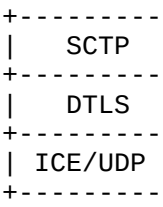


Figure 1: Basic stack diagram

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:

- A DTLS connection MUST be established before an SCTP association can be set up.
- 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.
- 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.
- 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.
- 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.
- 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, Gorry 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: June 7, 2018

X. Wei
Huawei Technologies
L.Zhu
Huawei Technologies
L.Deng
China Mobile

December 4, 2017

Tunnel Congestion Feedback
draft-ietf-tsvwg-tunnel-congestion-feedback-06

Abstract

This document describes a method to measure congestion on a tunnel segment based on recommendations from RFC 6040, "Tunneling of Explicit Congestion Notification", and to use IPFIX to communicate the congestion measurements from the tunnel's egress to a controller which can respond by modifying the traffic control policies at the tunnel's ingress.

Table of Contents

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.

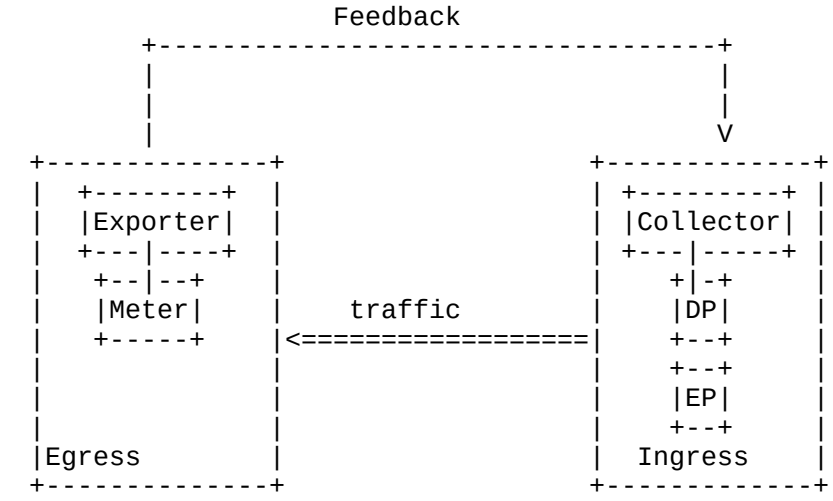


Figure 1: Feedback Model.

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:

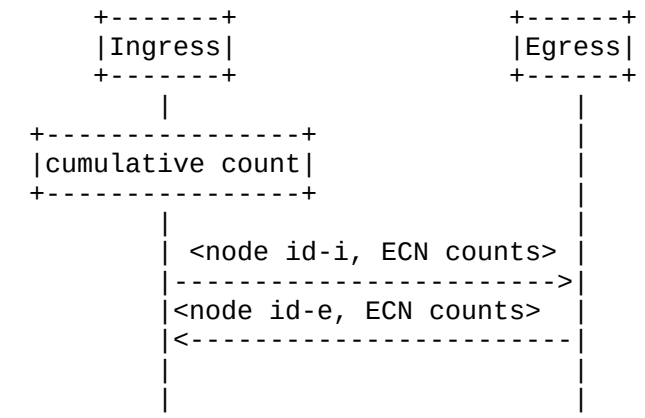


Figure 2: Procedure of Packet Loss Measurement

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

Statues: 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

Statues: current

Units: bytes

5.1.3 tunnelEcnEct1NectByteTotalCount

Description: The total number of bytes of incoming packets with ECT(1)|N-ECT ECN marking combination at the Observation Point since the Metering Process (re-)initialization for this Observation Point.

Abstract Data Type: unsigned64

Data Type Semantics: totalCounter

ElementId: TBD3

Statues: current

Units: bytes

5.1.4 tunnelEcnCeNectByteTotalCount

Description: The total number of bytes of incoming packets with CE|N-ECT ECN marking combination at the Observation Point since the Metering Process (re-)initialization for this Observation Point.

Abstract Data Type: unsigned64

Data Type Semantics: totalCounter

ElementId: TBD4

Statues: current

Units: bytes

5.1.5 tunnelEcnCeEct0ByteTotalCount

Description: The total number of bytes of incoming packets with CE|ECT(0) ECN marking combination at the Observation Point since the

Metering Process (re-)initialization for this Observation Point.

Abstract Data Type: unsigned64

Data Type Semantics: totalCounter

ElementId: TBD5

Statues: current

Units: bytes

5.1.6 tunnelEcnCeEct1ByteTotalCount

Description: The total number of bytes of incoming packets with CE|ECT(1) ECN marking combination at the Observation Point since the Metering Process (re-)initialization for this Observation Point.

Abstract Data Type: unsigned64

Data Type Semantics: totalCounter

ElementId: TBD6

Statues: current

Units: bytes

5.1.7 tunnelEcnEct0Ect0ByteTotalCount

Description: The total number of bytes of incoming packets with ECT(0)|ECT(0) ECN marking combination at the Observation Point since the Metering Process (re-)initialization for this Observation Point.

Abstract Data Type: unsigned64

Data Type Semantics: totalCounter

ElementId: TBD7

Statues: current

Units: bytes

5.1.8 tunnelEcnEct1Ect1PacketTotalCount

Description: The total number of bytes of incoming packets with ECT(1)|ECT(1) ECN marking combination at the Observation Point since

the Metering Process (re-)initialization for this Observation Point.

Abstract Data Type: unsigned64

Data Type Semantics: totalCounter

ElementId: TBD8

Statues: current

Units: bytes

5.1.9 tunnelEcnCEMarkedRatio

Description: The ratio of CE-marked Packet at the Observation Point.

Abstract Data Type: float32

ElementId: TBD8

Statues: current

6. Congestion Management

After tunnel ingress receives congestion level information, then congestion management actions could be taken based on the information, e.g. if the congestion level is higher than a predefined threshold, then action could be taken to reduce the congestion level.

The design of network side congestion management SHOULD take host side e2e congestion control mechanism into consideration, which means the congestion management needs to avoid the impacts on e2e congestion control. For instance, congestion management action must be delayed by more than a worst-case global RTT (e.g. 100ms), otherwise tunnel traffic management will not give normal e2e congestion control enough time to do its job, and the system could go unstable.

The detailed description of congestion management is out of scope of this document, as examples, congestion management such as circuit breaker [RFC8084] could be applied. Circuit breaker is an automatic mechanism to estimate congestion, and to terminate flow(s) when persistent congestion is detected to prevent network congestion collapse.

6.1 Example

This subsection provides an example of how the solution described in this document could work.

First of all, IPFIX template records are exchanged between ingress and egress to negotiate the format of data record, the example here is to measure the congestion level for the overall tunnel (caused by all the traffic in tunnel). After the negotiation is finished, ingress sends in-band message to egress, the message contains the number of each kind of ECN-marked packets (i.e. CE|CE, ECT|N-ECT and ECT|ECT) received until the sending of message.

After egress receives the message, the egress calculates CE-marked packet ratio and counts number of different kinds of ECN-marking packets received until receiving the message, then the egress sends a feedback message containing the counts together with the information in ingress's message to ingress.

Figure 3 to Figure 6 below show the example procedure between ingress and egress.

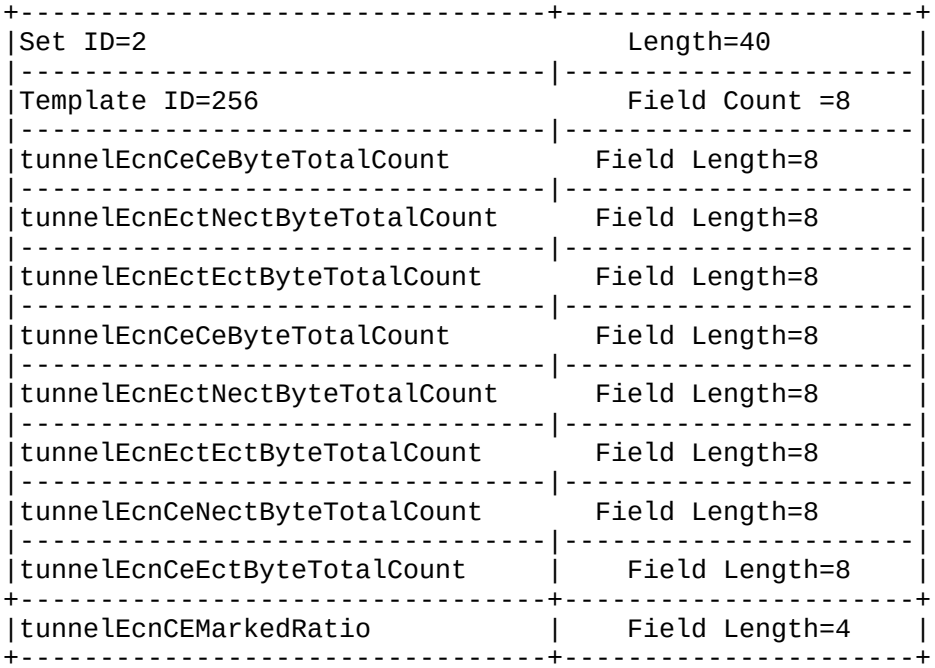


Figure 3: Template Record Sent From Egress to Ingress

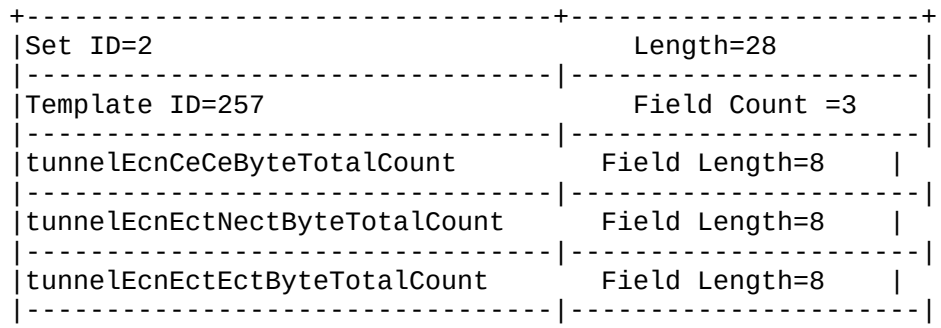


Figure 4: Template Record Sent From Ingress to Egress

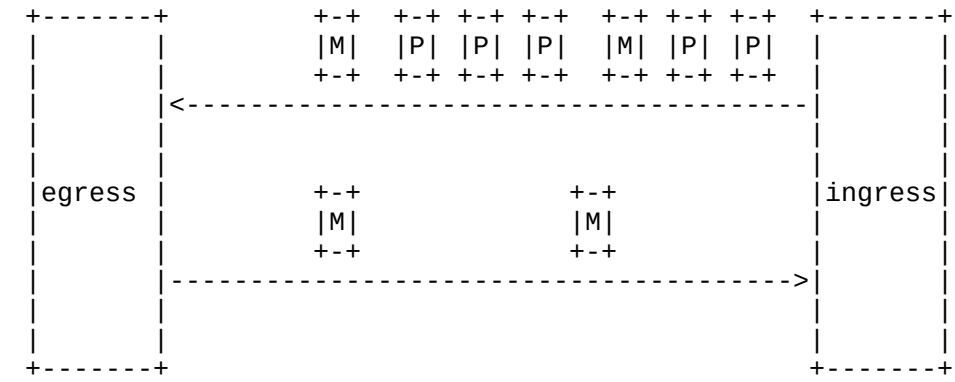


Figure 5 Traffic flow Between Ingress and Egress

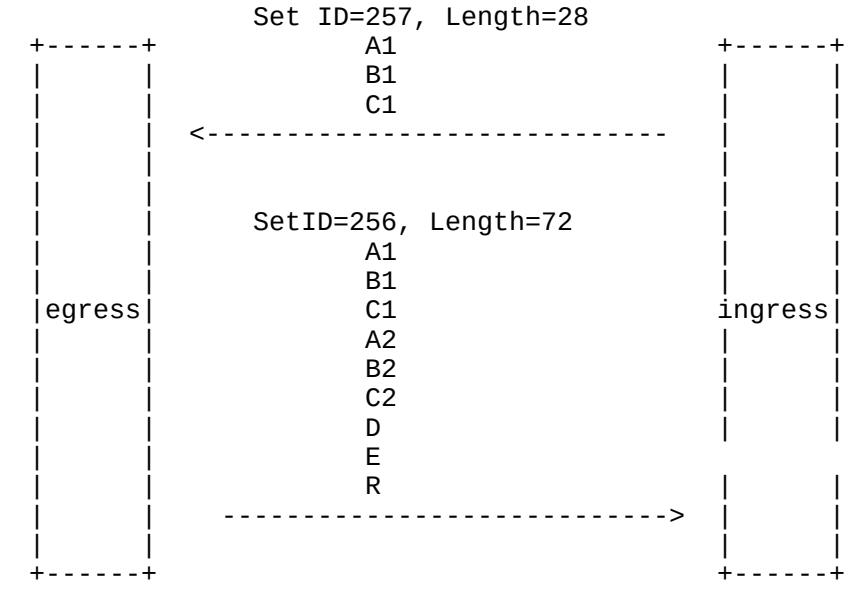


Figure 6: Message Between Ingress and Egress

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, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001, <http://www.rfc-editor.org/info/rfc3168>.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003, <http://www.rfc-editor.org/info/rfc3550>.

[RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, May 2004, <http://www.rfc-editor.org/info/rfc3758>.

[RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, March 2006, <http://www.rfc-editor.org/info/rfc4340>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, September 2007, <http://www.rfc-editor.org/info/rfc4960>.

[RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion

Notification", RFC 6040, November 2010, <http://www.rfc-editor.org/info/rfc6040>.

[RFC7011] Claise, B., Ed., Trammell, B., Ed., and P. Aitken, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information", STD 77, RFC 7011, September 2013, <http://www.rfc-editor.org/info/rfc7011>.

[RFC7013] Trammell, B. and B. Claise, "Guidelines for Authors and Reviewers of IP Flow Information Export (IPFIX) Information Elements", BCP 184, RFC 7013, September 2013, <http://www.rfc-editor.org/info/rfc7013>.

[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 Anthony Chan, Jake Holland, John Kaippallimalil and Vincent Roca for their careful reviews.

Authors' Addresses

Xinpeng Wei
Beiqing Rd. Z-park No.156, Haidian District,
Beijing, 100095, P. R. China
E-mail: weixinpeng@huawei.com

Zhu Lei
Beiqing Rd. Z-park No.156, Haidian District,
Beijing, 100095, P. R. China
E-mail:lei.zhu@huawei.com

Lingli Deng
Beijing, 100095, P. R. China
E-mail: denglingli@gmail.com

TSVWG
Internet Draft
Intended status: Standards Track
Intended updates: 768
Expires: January 2019

J. Touch

July 19, 2018

Transport Options for UDP
draft-ietf-tsvwg-udp-options-05.txt

Abstract

Transport protocols are extended through the use of transport header options. This document experimentally extends UDP by indicating the location, syntax, and semantics for UDP transport layer options.

Table of Contents

1. Introduction

Transport protocols use options as a way to extend their capabilities. TCP [RFC793], SCTP [RFC4960], and DCCP [RFC4340] include space for these options but UDP [RFC768] currently does not. This document defines an experimental extension to UDP that provides space for transport options including their generic syntax and semantics for their use in UDP's stateless, unreliable message protocol.

2. Conventions used in this document

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

In this document, these words will appear with that interpretation only when in ALL CAPS. Lowercase uses of these words are not to be interpreted as carrying significance described in RFC 2119.

In this document, the characters ">>" preceding an indented line(s) indicates a statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the portions of this RFC covered by these key words.

3. Background

Many protocols include a default header and an area for header options. These options enable the protocol to be extended for use in particular environments or in ways unforeseen by the original designers. Examples include TCP's Maximum Segment Size, Window Scale, Timestamp, and Authentication Options [RFC793][RFC5925][RFC7323].

These options are used both in stateful (connection-oriented, e.g., TCP [RFC793], SCTP [RFC4960], DCCP [RFC4340]) and stateless (connectionless, e.g., IPv4 [RFC791], IPv6 [RFC8200] protocols. In stateful protocols they can help extend the way in which state is managed. In stateless protocols their effect is often limited to individual packets, but they can have an aggregate effect on a sequence as well. One example of such uses is Substrate Protocol for User Datagrams (SPUD) [Tr16], and this document is intended to provide an out-of-band option area as an alternative to the in-band mechanism currently proposed [Hi15].

UDP is one of the most popular protocols that lacks space for options [RFC768]. The UDP header was intended to be a minimal

addition to IP, providing only ports and a data checksum for protection. This document experimentally extends UDP to provide a trailer area for options located after the UDP data payload.

4. The UDP Option Area

The UDP transport header includes demultiplexing and service identification (port numbers), a checksum, and a field that indicates the UDP datagram length (including UDP header). The UDP Length length field is typically redundant with the size of the maximum space available as a transport protocol payload (see also discussion in Section 9).

For IPv4, IP Total Length field indicates the total IP datagram length (including IP header), and the size of the IP options is indicated in the IP header (in 4-byte words) as the "Internet Header Length" (IHL), as shown in Figure 1 [RFC791]. As a result, the typical (and largest valid) value for UDP Length is:

```
UDP_Length = IPv4_Total_Length - IPv4_IHL * 4
```

For IPv6, the IP Payload Length field indicates the datagram after the base IPv6 header, which includes the IPv6 extension headers and space available for the transport protocol, as shown in Figure 2 [RFC8200]. Note that the Next HDR field in IPv6 might not indicate UDP (i.e., 17), e.g., when intervening IP extension headers are present. For IPv6, the lengths of any additional IP extensions are indicated within each extension [RFC8200], so the typical (and largest valid) value for UDP Length is:

```
UDP_Length = IPv6_Payload_Length - sum(extension header lengths)
```

In both cases, the space available for the UDP transport protocol data unit is indicated by IP, either completely in the base header (for IPv4) or adding information in the extensions (for IPv6). In either case, this document will refer to this available space as the "IP transport payload".

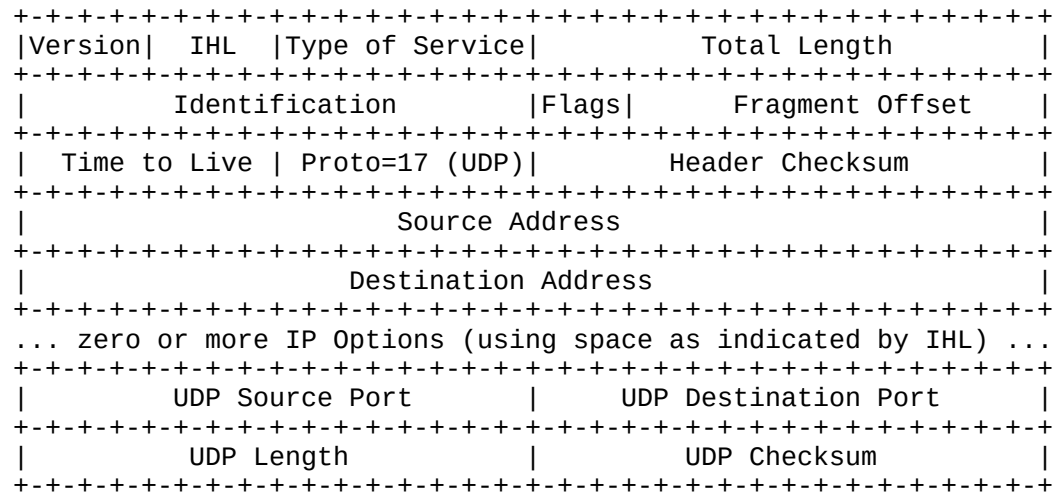


Figure 1 IPv4 datagram with UDP transport payload

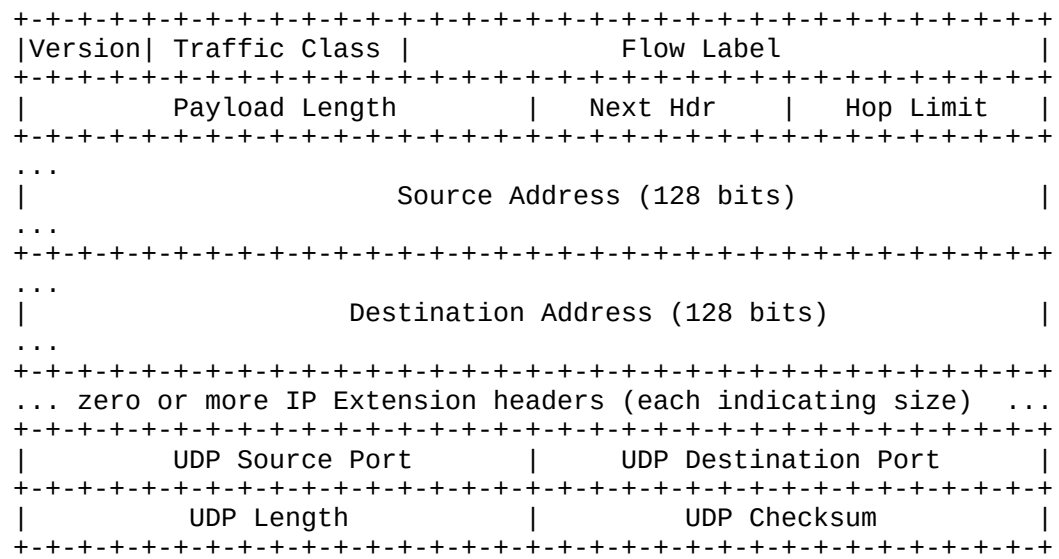


Figure 2 IPv6 datagram with UDP transport payload

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).

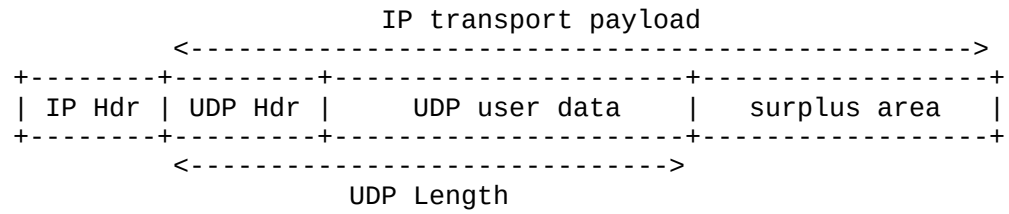


Figure 3 IP transport payload vs. UDP Length

In most cases, the IP transport payload and UDP Length point to the same location, indicating that there is no surplus area. It is important to note that this is not a requirement of UDP [RFC768] (discussed further in Section 9). UDP-Lite used the difference in these pointers to indicate the partial coverage of the UDP Checksum, such that the UDP user data, UDP header, and UDP pseudoheader (a subset of the IP header) are covered by the UDP checksum but additional user data in the surplus area is not covered [RFC3828]. This document uses the surplus area for UDP transport options.

The UDP option area is thus defined as the location between the end of the UDP payload and the end of the IP datagram as a trailing options area. This area can occur at any valid byte offset, i.e., it need not be 16-bit or 32-bit aligned. In effect, this document redefines the UDP "Length" field as a "trailer offset".

UDP options are defined using a TLV (type, length, and optional value) syntax similar to that of TCP [RFC793]. They are typically a minimum of two bytes in length as shown in Figure 4, excepting only the one byte options "No Operation" (NOP) and "End of Options List" (EOL) described below.

Figure 4 UDP option default format

>> UDP options MAY occur at any UDP length offset.

>> The UDP length MUST be at least as large as the UDP header (8) and no larger than the IP transport payload. Values outside this range MUST be silently discarded as invalid and logged where rate-limiting permits.

Others have considered using values of the UDP Length that is larger than the IP transport payload as an additional type of signal. Using

a value smaller than the IP transport payload is expected to be backward compatible with existing UDP implementations, i.e., to deliver the UDP Length of user data to the application and silently ignore the additional surplus area data. Using a value larger than the IP transport payload would either be considered malformed (and be silently dropped) or could cause buffer overruns, and so is not considered silently and safely backward compatible. Its use is thus out of scope for the extension described in this document.

>> UDP options MUST be interpreted in the order in which they occur in the UDP option area.

5. UDP Options

The following UDP options are currently defined:

Kind	Length	Meaning
0*	-	End of Options List (EOL)
1*	-	No operation (NOP)
2*	2	Option checksum (OCS)
3*	4	Alternate checksum (ACS)
4*	4	Lite (LITE)
5*	4	Maximum segment size (MSS)
6*	8/10	Fragmentation (FRAG)
7	10	Timestamps (TIME)
8	(varies)	Authentication and Encryption (AE)
9-126	(varies)	UNASSIGNED (assignable by IANA)
127-253		RESERVED
254	N(>=4)	RFC 3692-style experiments (EXP)
255		RESERVED

These options are defined in the following subsections.

>> An endpoint supporting UDP options MUST support those marked with a "*" above: EOL, NOP, OCS, ACS, LITE, FRAG, and MSS. This includes both recognizing and being able to generate these options if configured to do so.

>> All other options (without a "*") MAY be implemented, and their use SHOULD be determined either out-of-band or negotiated.

>> Receivers MUST silently ignore unknown options. That includes options whose length does not indicate the specified value.

>> Except for NOP, each option SHOULD NOT occur more than once in a single UDP datagram. If a non-NOP option occurs more than once, a

receiver MUST interpret only the first instance of that option and MUST ignore all others.

>> Only the OCS and AE options depend on the contents of the option area. AE is always computed as if the AE hash and OCS checksum are zero; OCS is always computed as if the OCS checksum is zero and after the AE hash has been computed. Future options MUST NOT be defined as having a value dependent on the contents of the option area. Otherwise, interactions between those values, OCS, and AE could be unpredictable.

Receivers cannot treat unexpected option lengths as invalid, as this would unnecessarily limit future revision of options (e.g., defining a new ACS that is defined by having a different length).

>> Option lengths MUST NOT exceed the IP length of the packet. If this occurs, the packet MUST be treated as malformed and dropped, and the event MAY be logged for diagnostics (logging SHOULD be rate limited).

>> Required options MUST come before other options. Each required option MUST NOT occur more than once (if they are repeated in a received segment, all except the first MUST be silently ignored).

The requirement that required options come before others is intended to allow for endpoints to implement DOS protection, as discussed further in Section 14.

5.1. End of Options List (EOL)

The End of Options List (EOL) option indicates that there are no more options. It is used to indicate the end of the list of options without needing to pad the options to fill all available option space.

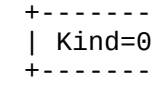


Figure 5 UDP EOL option format

>> When the UDP options do not consume the entire option area, the last non-NOP option SHOULD be EOL (vs. filling the entire option area with NOP values).

>> All bytes after EOL MUST be ignored by UDP option processing. As a result, there can only ever be one EOL option (even if other bytes were zero, they are ignored).

5.2. No Operation (NOP)

The No Operation (NOP) option is a one byte placeholder, intended to be used as padding, e.g., to align multi-byte options along 16-bit or 32-bit boundaries.

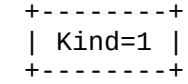


Figure 6 UDP NOP option format

>> If options longer than one byte are used, NOP options SHOULD be used at the beginning of the UDP options area to achieve alignment as would be more efficient for active (i.e., non-NOP) options.

>> Segments SHOULD NOT use more than three consecutive NOPs. NOPs are intended to assist with alignment, not other padding or fill.

[NOTE: Tom Herbert suggested we declare "more than 3 consecutive NOPs" a fatal error to reduce the potential of using NOPs as a DOS attack, but IMO there are other equivalent ways (e.g., using RESERVED or other UNASSIGNED values) and the "no more than 3" creates its own DOS vulnerability)

5.3. Option Checksum (OCS)

The Option Checksum (OCS) is an 8-bit ones-complement sum (Ones8) that covers all of the UDP options. OCS is 8-bits to allow the entire option to occupy a total of 16 bits. The primary purpose of OCS is to detect non-standard (i.e., non-option) uses of the surplus area.

OCS can be calculated by computing the 16-bit ones-complement sum and "folding over" the result (using carry wraparound). Note that OCS is direct, i.e., it is not negated or adjusted if zero (unlike the Internet checksum as used in IPv4, TCP, and UDP headers). OCS protects the option area from errors in a similar way that the UDP checksum protects the UDP user data.

Figure 7 UDP OCS option format

>> When present, the option checksum SHOULD occur as early as possible, preferably preceded by only NOP options for alignment and the LITE option if present.

OCS covers the entire UDP option area, including the Lite option as formatted before swapping (or relocation) for transmission (or, equivalently, after the swap/relocation after reception).

>> If the option checksum fails, all options MUST be ignored and any trailing surplus data (and Lite data, if used) silently discarded.

>> UDP data that is validated by a correct UDP checksum MUST be delivered to the application layer, even if the UDP option checksum fails, unless the endpoints have negotiated otherwise for this segment’s socket pair.

5.4. Alternate Checksum (ACS)

The Alternate Checksum (ACS) provides a stronger alternative to the checksum in the UDP header, using a 16-bit CRC of the conventional UDP payload only (excluding the IP pseudoheader, UDP header, and UDP options, and not include the LITE area). Because it does not include the IP pseudoheader or UDP header, it need not be updated by NATs when IP addresses or UDP ports are rewritten. Its purpose is to detect errors that the UDP checksum might not detect.

CRC-CCITT (polynomial x^16 + x^12 + x^5 + 1) has been chosen because of its ubiquity and use in other packet protocols, such as X.25, HDLC, and Bluetooth. The option contains FCS-16 as defined in Appendix C of [RFC1662], except that it is not inverted in the final step and that it is stored in the ACS option in network byte order.

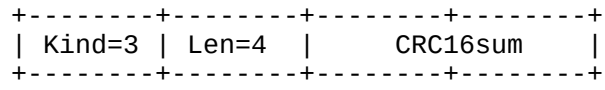


Figure 8 UDP ACS option format

When present, the ACS always contains a valid CRC checksum. There are no reserved values, including the value of zero. If the CRC is zero, this must indicate a valid checksum (i.e., it does not

indicate that the ACS is not used; instead, the option would simply not be included if that were the desired effect).

ACS does not protect the UDP pseudoheader; only the current UDP checksum provides that protection. ACS cannot provide that protection because it would need to be updated whenever the UDP pseudoheader changed, e.g., during NAT address and port translation; because this is not the case, ACS does not cover the pseudoheader.

5.5. Lite (LITE)

The Lite option (LITE) is intended to provide equivalent capability to the UDP Lite transport protocol [RFC3828]. UDP Lite allows the UDP checksum to cover only a prefix of the UDP data payload, to protect critical information (e.g., application headers) but allow potentially erroneous data to be passed to the user. This feature helps protect application headers but allows for application data errors. Some applications are impacted more by a lack of data than errors in data, e.g., voice and video.

>> When LITE is active, it MUST come first in the UDP options list.

LITE is intended to support the same API as for UDP Lite to allow applications to send and receive data that has a marker indicating the portion protected by the UDP checksum and the portion not protected by the UDP checksum.

LITE includes a 2-byte offset that indicates the length of the portion of the UDP data that is not covered by the UDP checksum.

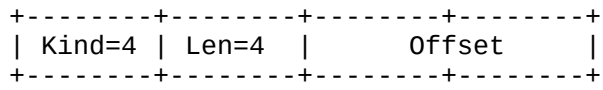


Figure 9 UDP LITE option format

At the sender, the option is formed using the following steps:

1. Create a LITE option, ordered as the first UDP option (Figure 10).

2. Calculate the location of the start of the options as an absolute offset from the start of the UDP header and place that length in the last two bytes of the LITE option.

3. If the LITE data area is 4 bytes or longer, swap all four bytes of the LITE option with the first 4 bytes of the LITE data area (Figure 11). If the LITE data area is 0-3 bytes long, slide the LITE option to the front of the LITE data area (i.e., placing the 0-3 bytes of LITE data after the LITE option).

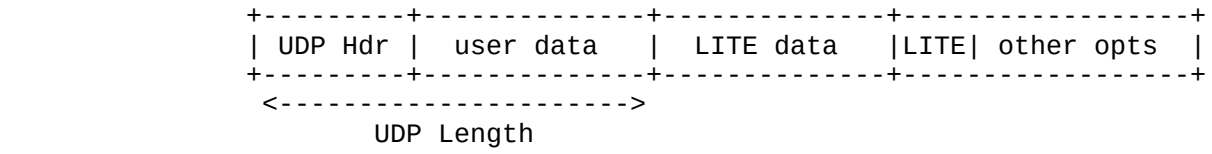


Figure 10 LITE option formation - LITE goes first

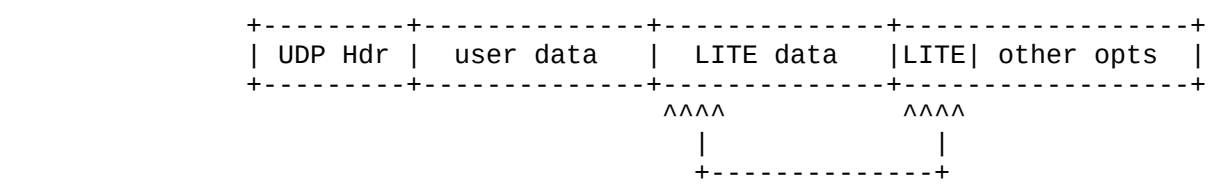


Figure 11 Before sending swap LITE option and front of LITE data

The resulting packet has the format shown in Figure 12. Note that the UDP length now points to the LITE option, and the LITE option points to the start of the option area.

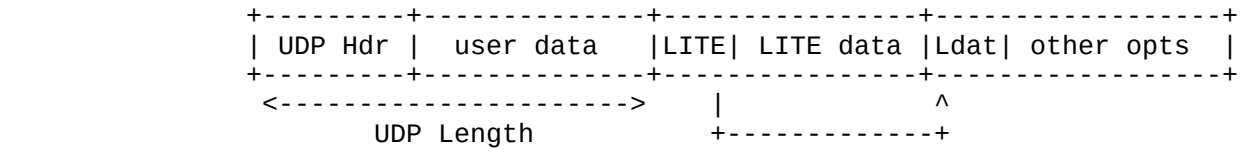


Figure 12 Lite option as sent

A legacy endpoint receiving this packet will discard the LITE option and everything that follows, including the lite data and remainder of the UDP options. The UDP checksum will protect only the user data, not the LITE option or lite data.

Receiving endpoints capable of processing UDP options will do the following:

1. Process options as usual. This will start at the LITE option.

2. When the LITE option is encountered, record its location as the start of the LITE data area and (if the LITE offset indicates a LITE data length of at least 4 bytes) swap the four bytes there with the four bytes at the location indicated inside the LITE option, which indicates the start of all of the options, including the LITE one (one past the end of the lite data area). If the LITE offset indicates a LITE data area of 0-3 bytes, then slide the LITE option forward that amount and slide the corresponding bytes after the LITE option to where the LITE option originally began. In either case, this restores the format of the option as it was prior to being sent, as per Figure 10.

3. Continue processing the remainder of the options, which are now in the format shown in Figure 11.

The purpose of this swap (or slide) is to support the equivalent of UDP Lite operation together with other UDP options without requiring the entire LITE data area to be moved after the UDP option area.

5.6. Maximum Segment Size (MSS)

The Maximum Segment Size (MSS, Kind = 3) is a 16-bit indicator of the largest UDP segment that can be received. As with the TCP MSS option [RFC793], the size indicated is the IP layer MTU decreased by the fixed IP and UDP headers only [RFC6691]. The space needed for IP and UDP options need to be adjusted by the sender when using the value indicated. The value transmitted is based on EMTU_R, the largest IP datagram that can be received (i.e., reassembled at the receiver) [RFC1122].

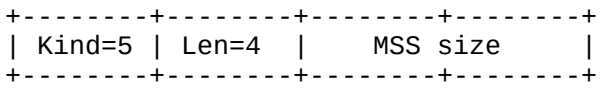


Figure 13 UDP MSS option format

The UDP MSS option MAY be used for path MTU discovery [RFC1191][RFC8201], but this may be difficult because of known issues with ICMP blocking [RFC2923] as well as UDP lacking automatic retransmission. It is more likely to be useful when coupled with IP source fragmentation to limit the largest reassembled UDP message, e.g., when EMTU_R is larger than the required minimums (576 for IPv4 [RFC791] and 1500 for IPv6 [RFC8200]).

5.7. Fragmentation (FRAG)

The Fragmentation option (FRAG) supports UDP fragmentation and reassembly, which can be used to transfer UDP messages larger than limited by the IP receive MTU (EMTU_R [RFC1122]). It is typically used with the UDP MSS option to enable more efficient use of large messages, both at the UDP and IP layers. FRAG is designed similar to the IPv6 Fragmentation Header [RFC8200], except that the UDP variant uses a 16-bit Offset measured in bytes, rather than IPv6's 13-bit Fragment Offset measured in 8-byte units. This UDP variant avoids creating reserved fields.

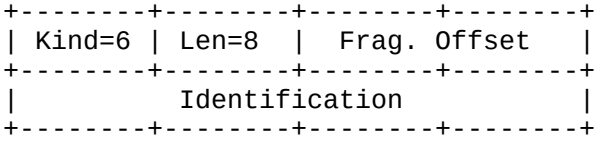


Figure 14 UDP non-terminal FRAG option format

The FRAG option also lacks a "more" bit, zeroed for the terminal fragment of a set. This is possible because the terminal FRAG option is indicated as a longer, 10-byte variant, which includes an Internet checksum over the entire reassembled UDP payload (omitting the IP pseudoheader and UDP header, as well as UDP options), as shown in Figure 15.

>> The reassembly checksum SHOULD be used, but MAY be unused in the same situations when the UDP checksum is unused (e.g., for transit tunnels or applications that have their own integrity checks [RFC8200]), and by the same mechanism (set the field to 0x0000).

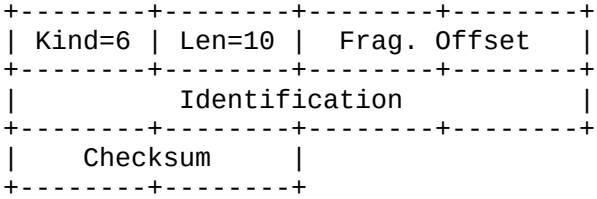


Figure 15 UDP terminal FRAG option format

>> During fragmentation, the UDP header checksum of each fragment needs to be recomputed based on each datagram's pseudoheader.

>> After reassembly is complete and validated using the checksum of the terminal FRAG option, the UDP header checksum of the resulting

datagram needs to be recomputed based on the datagram's pseudoheader.

The Fragment Offset is 16 bits and indicates the location of the UDP payload fragment in bytes from the beginning of the original unfragmented payload. The Len field indicates whether there are more fragments (Len=8) or no more fragments (Len=12).

>> The Identification field is a 32-bit value that MUST be unique over the expected fragment reassembly timeout.

>> The Identification field SHOULD be generated in a manner similar to that of the IPv6 Fragment ID [RFC8200].

>> UDP fragments MUST NOT overlap.

FRAG needs to be used with extreme care because it will present incorrect datagram boundaries to a legacy receiver, unless encoded as LITE data (see Section 5.7.1).

>> A host SHOULD indicate FRAG support by transmitting an unfragmented datagram using the Fragmentation option (e.g., with Offset zero and length 12, i.e., including the checksum area), except when encoded as LITE.

>> A host MUST NOT transmit a UDP fragment before receiving recent confirmation from the remote host, except when FRAG is encoded as LITE.

UDP fragmentation relies on a fragment expiration timer, which can be preset or could use a value computed using the UDP Timestamp option.

>> The default UDP reassembly SHOULD be no more than 2 minutes.

Implementers are advised to limit the space available for UDP reassembly.

>> UDP reassembly space SHOULD be limited to reduce the impact of DOS attacks on resource use.

>> UDP reassembly space limits SHOULD NOT be implemented as an aggregate, to avoid cross-socketpair DOS attacks.

>> Individual UDP fragments MUST NOT be forwarded to the user. The reassembled datagram is received only after complete reassembly,

checksum validation, and continued processing of the remaining options.

Any additional UDP options would follow the FRAG option in the final fragment, and would be included in the reassembled packet. Processing of those options would commence after reassembly.

>> UDP options MUST NOT follow the FRAG header in non-terminal fragments. Any data following the FRAG header in non-terminal fragments MUST be silently dropped. All other options that apply to a reassembled packet MUST follow the FRAG header in the terminal fragment.

5.7.1. Coupling FRAG with LITE

FRAG can be coupled with LITE to avoid impacting legacy receivers. Each fragment is sent as LITE un-checksummed data, where each UDP packet contains no legacy-compatible data. Legacy receivers interpret these as zero-payload packets, which would not affect the receiver unless the presence of the packet itself were a signal. The header of such a packet would appear as shown in Figure 16 and Figure 17.

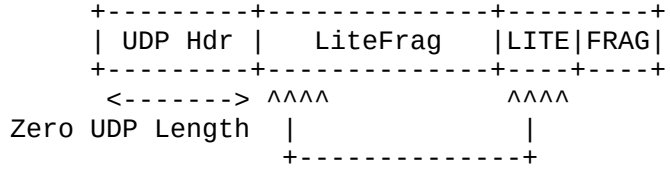


Figure 16 Preparing FRAG as Lite data

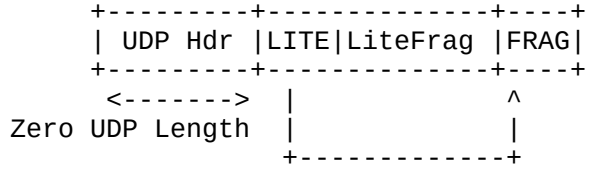


Figure 17 Lite option before transmission

When a packet is reassembled, it appears as a complete LITE data region. The UDP header of the reassembled packet is adjusted accordingly, so that the reassembled region now appears as conventional UDP user data, and processing of the UDP options continues, as with the non-LITE FRAG variant.

5.8. Timestamps (TIME)

The UDP Timestamp option (TIME) exchanges two four-byte timestamp fields. It serves a similar purpose to TCP's TS option [RFC7323], enabling UDP to estimate the round trip time (RTT) between hosts. For UDP, this RTT can be useful for establishing UDP fragment reassembly timeouts or transport-layer rate-limiting [RFC8085].

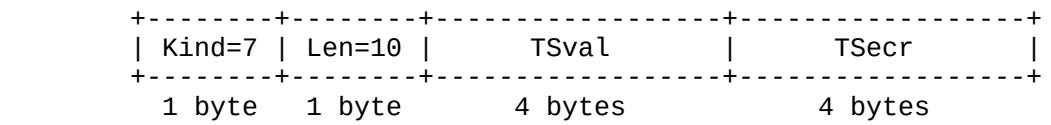


Figure 18 UDP TIME option format

TS Value (TSval) and TS Echo Reply (TSecr) are used in a similar manner to the TCP TS option [RFC7323]. On transmitted segments using the option, TS Value is always set based on the local "time" value. Received TSval and TSecr values are provided to the application, which can pass the TSval value to be used as TSecr on UDP messages sent in response (i.e., to echo the received TSval). A received TSecr of zero indicates that the TSval was not echoed by the transmitter, i.e., from a previously received UDP packet.

>> UDP MAY use an RTT estimate based on nonzero Timestamp values as a hint for fragmentation reassembly, rate limiting, or other mechanisms that benefit from such an estimate.

>> UDP SHOULD make this RTT estimate available to the user application.

5.9. Authentication and Encryption (AE)

The Authentication and Encryption option (AE) is intended to allow UDP to provide a similar type of authentication as the TCP Authentication Option (TCP-AO) [RFC5925]. It uses the same format as specified for TCP-AO, except that it uses a Kind of 8. UDP-AO supports NAT traversal in a similar manner as TCP-AO [RFC6978]. UDP-AO can also be extended to provide a similar encryption capability as TCP-AO-ENC, in a similar manner [To18ao]. For these reasons, the option is known as UDP-AE.

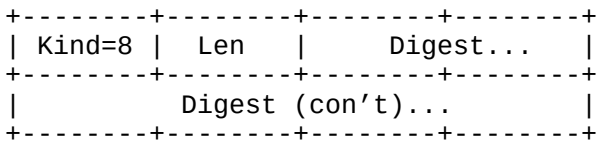


Figure 19 UDP non-terminal FRAG option format

Like TCP-AO, UDP-AE is not negotiated in-band. Its use assumes both endpoints have populated Master Key Tuples (MKTs), used to exclude non-protected traffic.

TCP-AO generates unique traffic keys from a hash of TCP connection parameters. UDP lacks a three-way handshake to coordinate connection-specific values, such as TCP's Initial Sequence Numbers (ISNs) [RFC793], thus UDP-AE's Key Derivation Function (KDF) uses zeroes as the value for both ISNs. This means that the UDP-AE reuses keys when socket pairs are reused, unlike TCP-AO.

UDP-AE can be configured to either include or exclude UDP options, the same way as can TCP-AO. When UDP options are covered, the OCS option area checksum and UDP-AE hash areas are zeroed before computing the UDP-AE hash. It is important to consider that options not yet defined might yield unpredictable results if not confirmed as supported, e.g., if they were to contain other hashes or checksums that depend on the option area contents. This is why such dependencies are not permitted except as defined for OCS and UDP-AE.

Similar to TCP-AO-NAT, UDP-AE can be configured to support NAT traversal, excluding one or both of the UDP ports [RFC6978].

5.10. Experimental (EXP)

The Experimental option (EXP) is reserved for experiments [RFC3692]. It uses a Kind value of 254. Only one such value is reserved because experiments are expected to use an Experimental ID (ExIDs) to differentiate concurrent use for different purposes, using UDP ExIDs registered with IANA according to the approach developed for TCP experimental options [RFC6994].

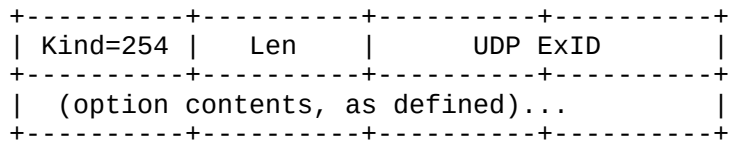


Figure 20 UDP EXP option format

>> 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]).

6. UDP API Extensions

UDP currently specifies an application programmer interface (API), summarized as follows (with Unix-style command as an example) [RFC768]:

- Method to create new receive ports
  - E.g., bind(handle, recvaddr(optional), recvport)
- Receive, which returns data octets, source port, and source address
  - E.g., recvfrom(handle, srcaddr, srcport, data)
- Send, which specifies data, source and destination addresses, and source and destination ports
  - E.g., sendto(handle, destaddr, destport, data)

This API is extended to support options as follows:

- Extend the method to create receive ports to include receive options that are required. Datagrams not containing these required options MUST be silently dropped and MAY be logged.
- Extend the receive function to indicate the options and their parameters as received with the corresponding received datagram.
- Extend the send function to indicate the options to be added to the corresponding sent datagram.

Examples of API instances for Linux and FreeBSD are provided in Appendix A, to encourage uniform cross-platform implementations.

7. Whose options are these?

UDP options are indicated in an area of the IP payload that is not used by UDP. That area is really part of the IP payload, not the UDP payload, and as such, it might be tempting to consider whether this is a generally useful approach to extending IP.

Unfortunately, the surplus area exists only for transports that include their own transport layer payload length indicator. TCP and SCTP include header length fields that already provide space for transport options by indicating the total length of the header area, such that the entire remaining area indicated in the network layer (IP) is transport payload. UDP-Lite already uses the UDP Length field to indicate the boundary between data covered by the transport checksum and data not covered, and so there is no remaining area where the length of the UDP-Lite payload as a whole can be indicated [RFC3828].

UDP options are intended for use only by the transport endpoints. They are no more (or less) appropriate to be modified in-transit than any other portion of the transport datagram.

UDP options are transport options. Generally, transport datagrams are not intended to be modified in-transit. However, the UDP option mechanism provides no specific protection against in-transit modification of the UDP header, UDP payload, or UDP option area, except as provided by the options selected (e.g., OCS, ACS, or AE).

8. UDP options LITE option vs. UDP-Lite

UDP-Lite provides partial checksum coverage, so that packets with errors in some locations can be delivered to the user [RFC3828]. It uses a different transport protocol number (136) than UDP (17) to interpret the UDP Length field as the prefix covered by the UDP checksum.

UDP (protocol 17) already defines the UDP Length field as the limit of the UDP checksum, but by default also limits the data provided to the application as that which precedes the UDP Length. A goal of UDP-Lite is to deliver data beyond UDP Length as a default, which is why a separate transport protocol number was required.

UDP options do not use or need a separate transport protocol number because the data beyond the UDP Length offset (surplus data) is not provided to the application by default. That data is interpreted exclusively within the UDP transport layer.

The LITE UDP options option supports a similar service to UDP-Lite. The main difference is that UDP-Lite provides the un-checksummed user data to the application by default, whereas the LITE UDP option can safely provide that service only between endpoints that negotiate that capability in advance. An endpoint that does not implement UDP options would silently discard this non-checksummed user data, along with the UDP options as well.

UDP-Lite cannot support UDP options, either as proposed here or in any other form, because the entire payload of the UDP packet is already defined as user data and there is no additional field in which to indicate a separate area for options. The UDP Length field in UDP-Lite is already used to indicate the boundary between user data covered by the checksum and user data not covered.

9. Interactions with Legacy Devices

It has always been permissible for the UDP Length to be inconsistent with the IP transport payload length [RFC768]. Such inconsistency has been utilized in UDP-Lite using a different transport number. There are no known systems that use this inconsistency for UDP [RFC3828]. It is possible that such use might interact with UDP options, i.e., where legacy systems might generate UDP datagrams that appear to have UDP options. The UDP OCS provides protection against such events and is stronger than a static "magic number".

UDP options have been tested as interoperable with Linux, macOS, and Windows Cygwin, and worked through NAT devices. These systems successfully delivered only the user data indicated by the UDP Length field and silently discarded the surplus area.

One reported embedded device passes the entire IP datagram to the UDP application layer. Although this feature could enable application-layer UDP option processing, it would require that conventional UDP user applications examine only the UDP payload. This feature is also inconsistent with the UDP application interface [RFC768] [RFC1122].

It has been reported that Alcatel-Lucent’s "Brick" Intrusion Detection System has a default configuration that interprets inconsistencies between UDP Length and IP Length as an attack to be reported. Note that other firewall systems, e.g., CheckPoint, use a default "relaxed UDP length verification" to avoid falsely interpreting this inconsistency as an attack.

(TBD: test with UDP checksum offload and UDP fragmentation offload)

10. Options in a Stateless, Unreliable Transport Protocol

There are two ways to interpret options for a stateless, unreliable protocol -- an option is either local to the message or intended to affect a stream of messages in a soft-state manner. Either interpretation is valid for defined UDP options.

It is impossible to know in advance whether an endpoint supports a UDP option.

>> UDP options MUST allow for silent failure on first receipt.

>> UDP options that rely on soft-state exchange MUST allow for message reordering and loss.

>> A UDP option MUST be silently optional until confirmed by exchange with an endpoint.

The above requirements prevent using any option that cannot be safely ignored unless that capability has been negotiated with an endpoint in advance for a socket pair. Legacy systems would need to be able to interpret the transport payload fragments as individual transport datagrams.

11. UDP Option State Caching

Some TCP connection parameters, stored in the TCP Control Block, can be usefully shared either among concurrent connections or between connections in sequence, known as TCP Sharing [RFC2140][To18cb]. Although UDP is stateless, some of the options proposed herein may have similar benefit in being shared or cached. We call this UCB Sharing, or UDP Control Block Sharing, by analogy.

[TBD: extend this section to indicate which options MAY vs. MUST NOT be shared and how, e.g., along the lines of To18cb]

12. Updates to RFC 768

This document updates RFC 768 as follows:

- This document defines the meaning of the IP payload area beyond the UDP length but within the IP length.

- This document extends the UDP API to support the use of options.

13. Multicast Considerations

UDP options are primarily intended for unicast use. Using these options over multicast IP requires careful consideration, e.g., to ensure that the options used are safe for different endpoints to interpret differently (e.g., either to support or silently ignore) or to ensure that all receivers of a multicast group confirm support for the options in use.

14. Security Considerations

The use of UDP packets with inconsistent IP and UDP Length fields has the potential to trigger a buffer overflow error if not properly handled, e.g., if space is allocated based on the smaller field and copying is based on the larger. However, there have been no reports of such vulnerability and it would rely on inconsistent use of the two fields for memory allocation and copying.

UDP options are not covered by DTLS (datagram transport-layer security). Despite the name, neither TLS [RFC5246] (transport layer security, for TCP) nor DTLS [RFC6347] (TLS for UDP) protect the transport layer. Both operate as a shim layer solely on the payload of transport packets, protecting only their contents. Just as TLS does not protect the TCP header or its options, DTLS does not protect the UDP header or the new options introduced by this document. Transport security is provided in TCP by the TCP Authentication Option (TCP-AO [RFC5925]) or in UDP by the Authentication Extension option (Section 5.9). Transport headers are also protected as payload when using IP security (IPsec) [RFC4301].

UDP options use the TLV syntax similar to that of TCP. This syntax is known to require serial processing and may pose a DOS risk, e.g., if an attacker adds large numbers of unknown options that must be parsed in their entirety. Implementations concerned with the potential for this vulnerability MAY implement only the required options and MAY also limit processing of TLVs. Because required options come first and at most once each (with the exception of NOPs, which should never need to come in sequences of more than three in a row), this limits their DOS impact. Note that when a packet’s options cannot be processed, it MUST be discarded; the packet and its options should always share the same fate.

15. IANA Considerations

Upon publication, IANA is hereby requested to create a new registry for UDP Option Kind numbers, similar to that for TCP Option Kinds. Initial values of this registry are as listed in Section 5.

Additional values in this registry are to be assigned by IESG Approval or Standards Action [RFC8126].

Upon publication, IANA is hereby requested to create a new registry for UDP Experimental Option Experiment Identifiers (UDP ExIDs) for use in a similar manner as TCP ExIDs [RFC6994]. This registry is initially empty. Values in this registry are to be assigned by IANA using first-come, first-served (FCFS) rules [RFC8126].

16. References

16.1. Normative References

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

[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.

[RFC1662] Simpson, W. Ed., "PPP in HDLC-like Framing," RFC 1662, Oct. 1994.

16.2. Informative References

[Hi15] 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.

[RFC1191] Mogul, J., S. Deering, "Path MTU discovery," RFC 1191, November 1990.

[RFC2140] Touch, J., "TCP Control Block Interdependence," RFC 2140, Apr. 1997.

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

[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.

[RFC4960] Stewart, R. (Ed.), "Stream Control Transmission Protocol", RFC 4960, September 2007.

[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.

[RFC5246] Dierks, T., E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, Aug. 2008.

[RFC5925] Touch, J., A. Mankin, R. Bonica, "The TCP Authentication Option," RFC 5925, June 2010.

[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.

[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.

[To18ao] Touch, J., "A TCP Authentication Option Extension for Payload Encryption", draft-touch-tcp-ao-encrypt, Jan. 2018.

[To18cb] Touch, J., M. Welzl, S. Islam, J. You, "TCP Control Block Interdependence," draft-touch-tcpm-2140bis, Jan. 2018.

[Tr16] Trammel, B. (Ed.), M. Kuelewind (Ed.), "Requirements for the design of a Substrate Protocol for User Datagrams (SPUD)," draft-trammell-spud-req-04, May 2016.

17. Acknowledgments

This work benefitted from feedback from Bob Briscoe, Ken Calvert, Ted Faber, Gorry Fairhurst, C. M. Heard (including the FRAG/LITE combination), Tom Herbert, and Mark Smith, as well as discussions on the IETF TSVWG and SPUD email lists.

This work is partly supported by USC/ISI’s Postel Center.

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Joe Touch

Manhattan Beach, CA 90266 USA

Phone: +1 (310) 560-0334
Email: touch@strayalpha.com

Appendix A. Implementation Information

The following information is provided to encourage interoperable API implementations.

System-level variables (sysctl):

Name	default	meaning
net.ipv4.udp_opt	0	UDP options available
net.ipv4.udp_opt_ocs	1	Default include OCS
net.ipv4.udp_opt_acs	0	Default include ACS
net.ipv4.udp_opt_lite	0	Default include LITE
net.ipv4.udp_opt_mss	0	Default include MSS
net.ipv4.udp_opt_time	0	Default include TIME
net.ipv4.udp_opt_frag	0	Default include FRAG
net.ipv4.udp_opt_ae	0	Default include AE

Socket options (sockopt), cached for outgoing datagrams:

Name	meaning
UDP_OPT	Enable UDP options (at all)
UDP_OPT_OCS	Enable UDP OCS option
UDP_OPT_ACS	Enable UDP ACS option
UDP_OPT_LITE	Enable UDP LITE option
UDP_OPT_MSS	Enable UDP MSS option
UDP_OPT_TIME	Enable UDP TIME option
UDP_OPT_FRAG	Enable UDP FRAG option
UDP_OPT_AE	Enable UDP AE option

Send/sendto parameters:

(TBD - currently using cached 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

Table of Contents

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 applicationsby 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:

- the Deterministic Network setup matches the needs of the Application
- 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
- the use of the network is optimized so as to ensure that every byte from the application can effectively be transported
- 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.

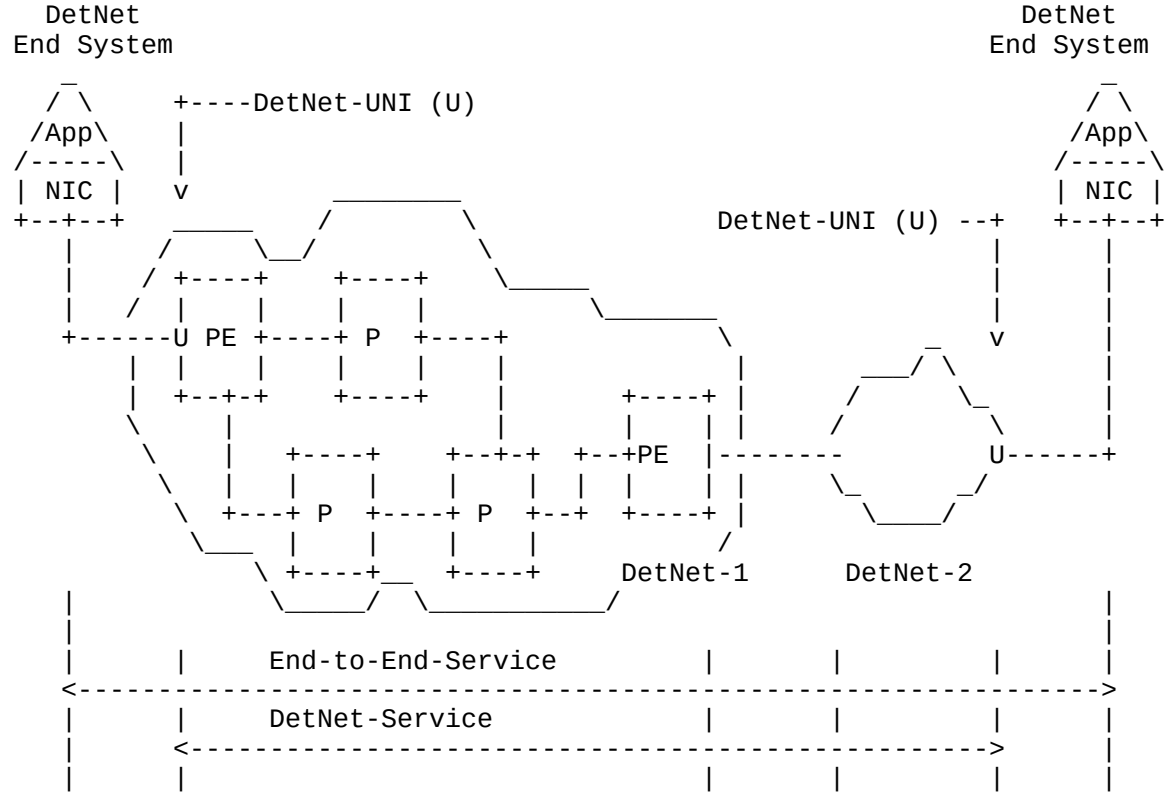


Figure 1: DetNet Service Reference Model (multi-domain)

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.

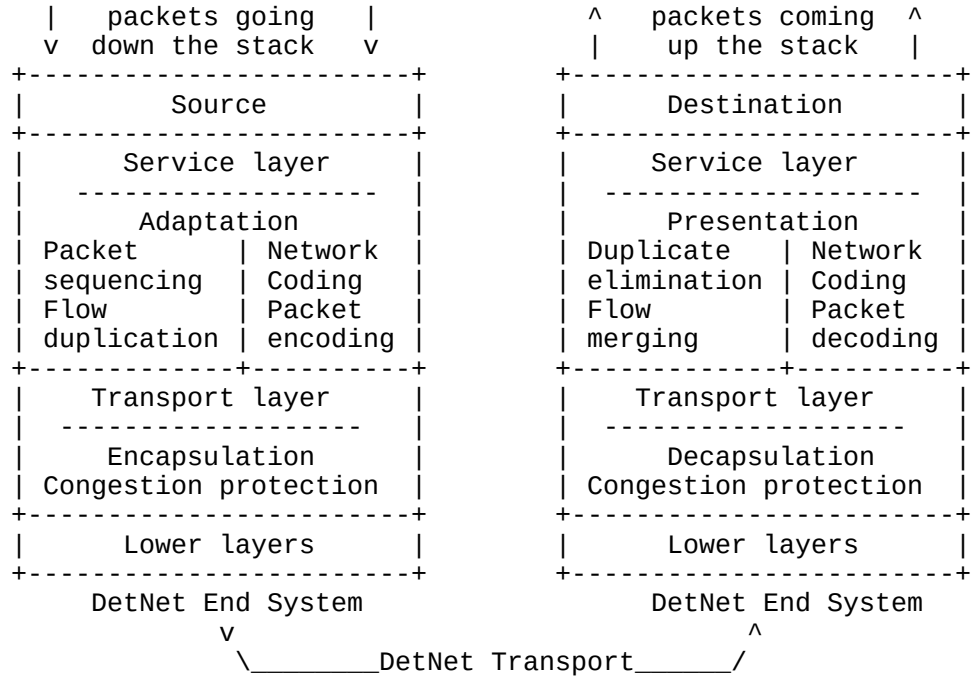


Figure 2: DetNet-Capable End-System Protocol Stack

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:

- 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.

- 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 Determistic End-to-End Service Flow (DEESF), or DetTrans Flow, to refer to an end-to-end application flow, and the term Determistic Service Flow (DSF), or DetNet Flow, to refer to a lower layer deterministic transport. This is illustrated in Figure 3.

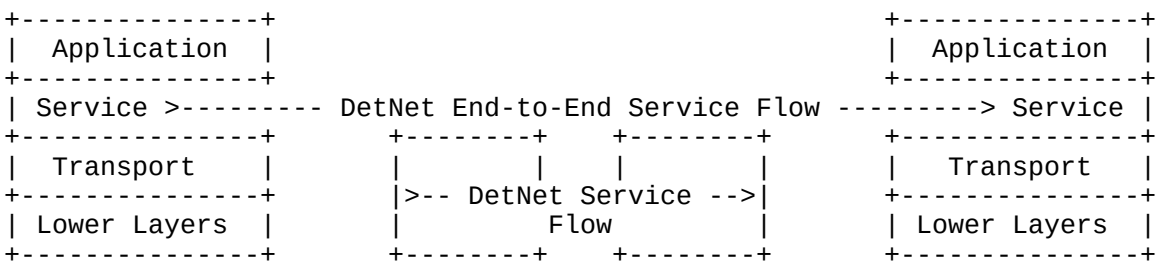


Figure 3: DetTrans vs. DetNet Flows

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 Determistic 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.

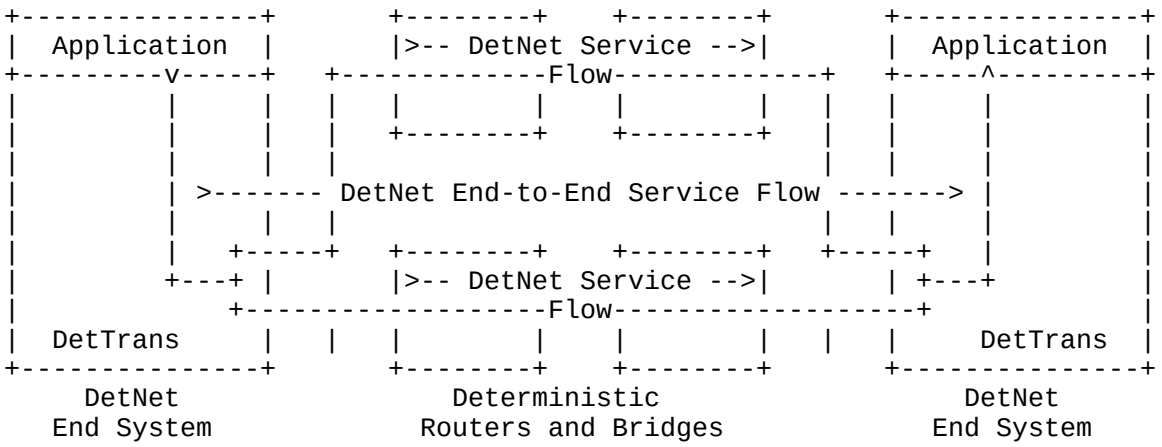


Figure 4: Load Sharing

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.

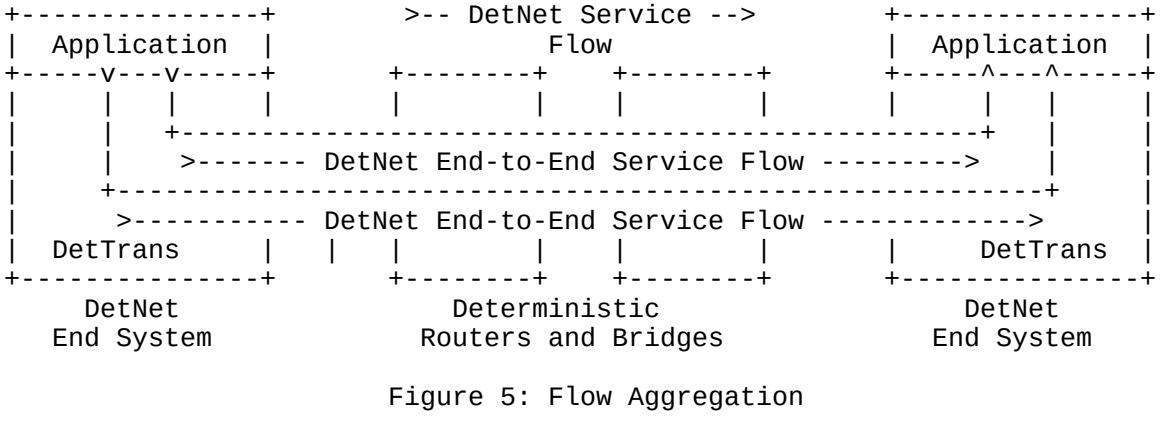


Figure 5: Flow Aggregation

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 segemnt 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 banswidth 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.

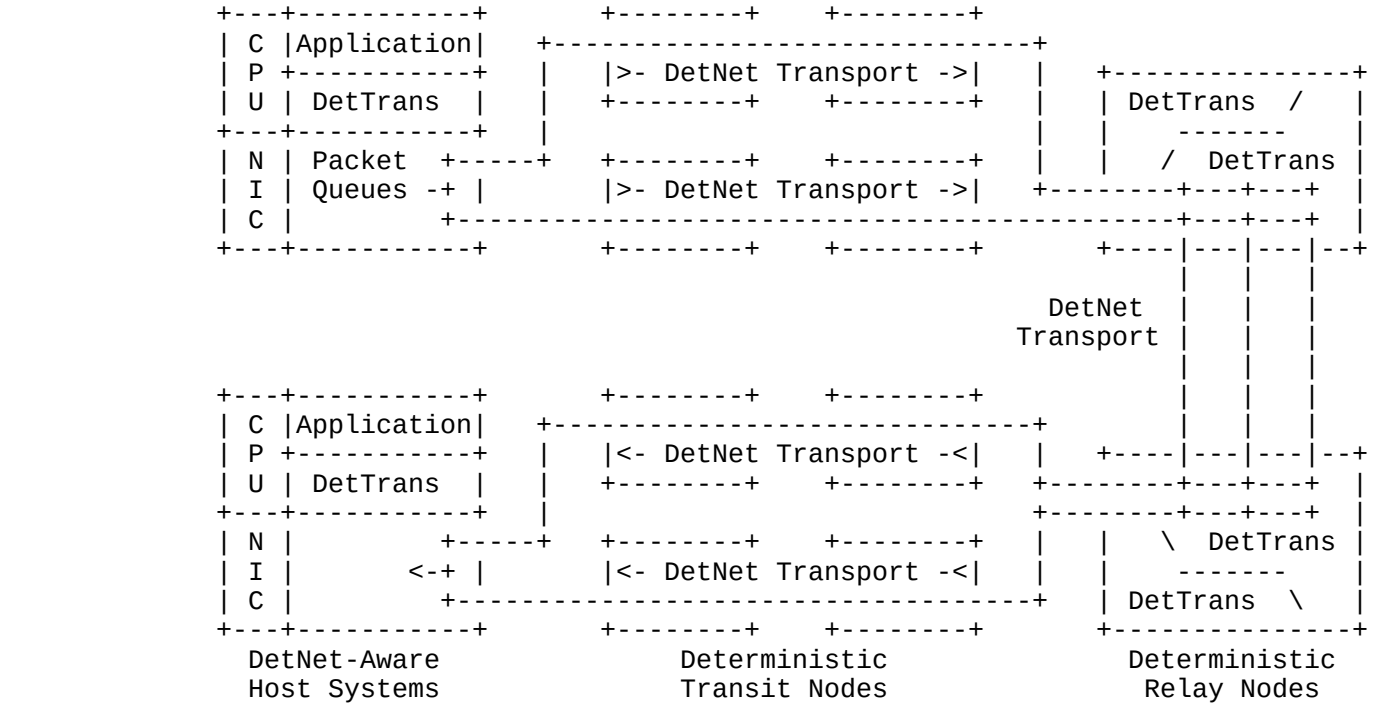


Figure 6: Intermediate Systems

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.

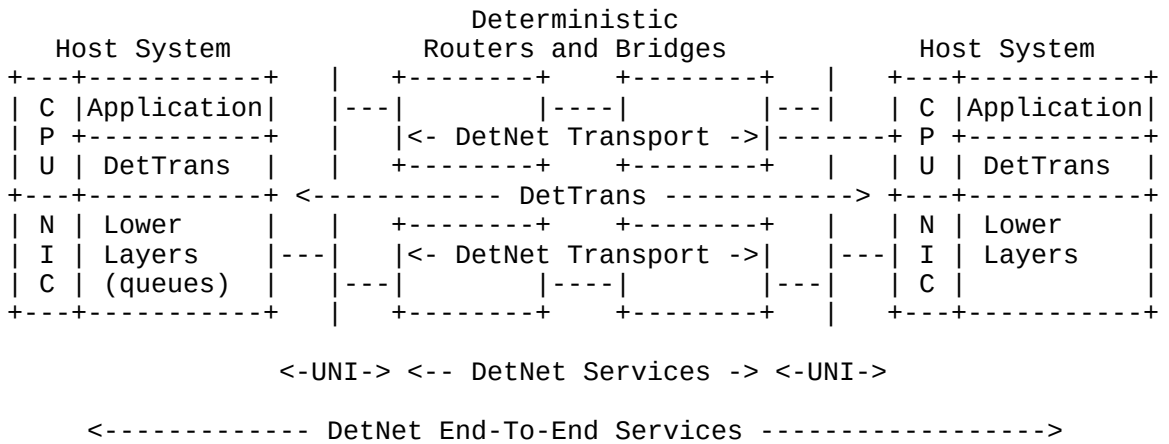


Figure 7: Example Physical Network

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.

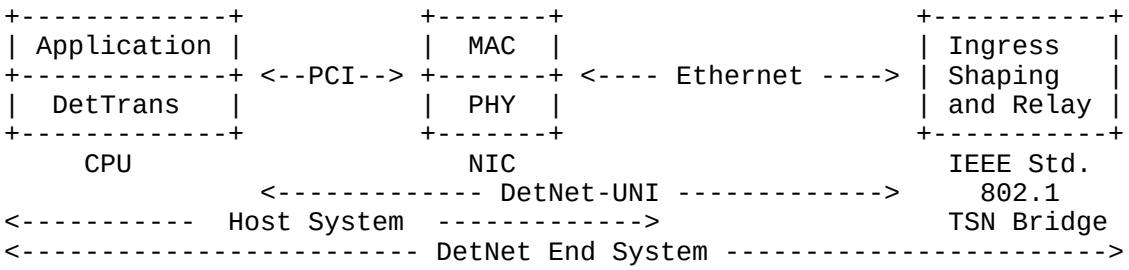
The DetNet UNI uses ICMPv6 to carry its protocol elements. Data Packets across the UNI are encapsulated in order to carry DetNet-UNI control information to identify the reason of a loss or a delay, and determine the action to be taken in case of a packet lost or delayed over the interface.

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.



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.

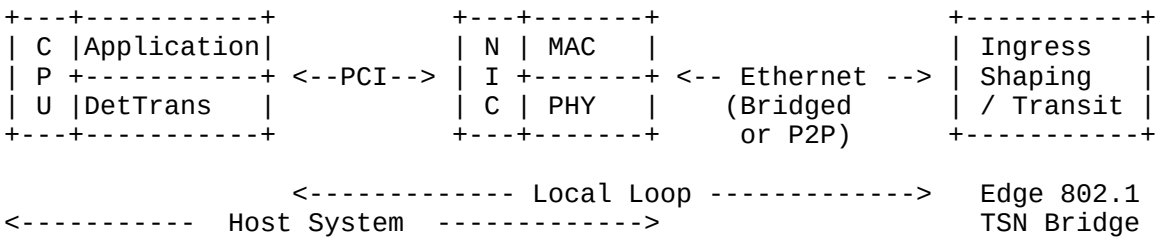


Figure 9: DetNet Unaware NIC

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.

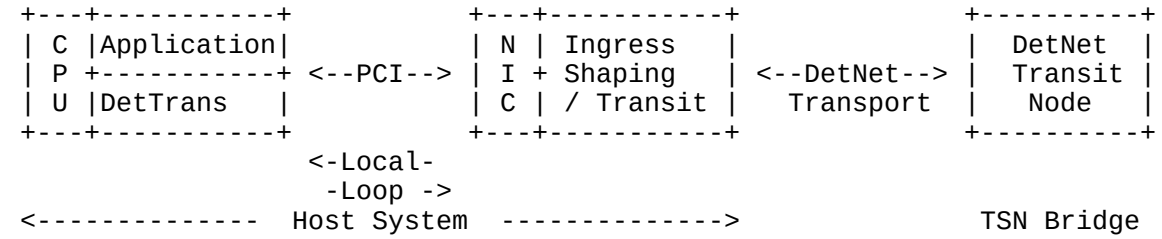


Figure 10: DetNet Capable NIC

If the NIC is DetNet-Aware and does not participates 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.

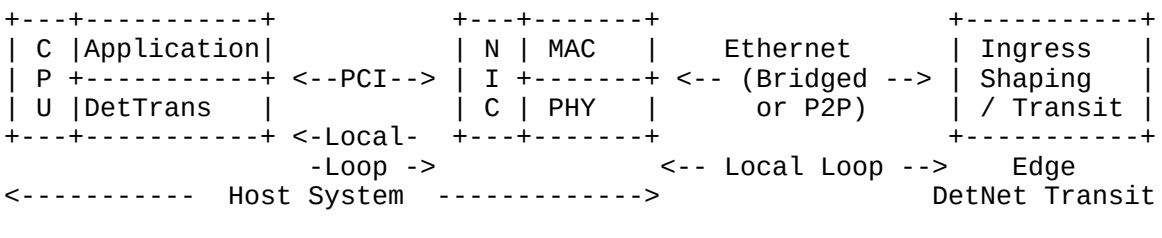


Figure 11: DetNet Capable NIC

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 is next time out.

5.2.3. Loss of a Control Message

The loss of a packet beween 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 Detrans 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>=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 of 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., Vizarreta, 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
Internet-Draft
Updates: 6951 (if approved)
Intended status: Standards Track
Expires: January 4, 2018

M. Tuexen
Muenster Univ. of Appl. Sciences
R. Stewart
Netflix, Inc.
July 3, 2017

Additional Considerations for UDP Encapsulation of Stream Control Transmission Protocol (SCTP) Packets
draft-tuexen-tsvwg-sctp-udp-encaps-cons-02.txt

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 for the handling of received packets for which this check is not possible.

This document updates RFC 6951 by specifying the handling of received packets where the verification tag can not be checked.

Table of Contents

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 automatically 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 for the handling of received packets where this verification tag check is not possible. This includes packets for which no association can be found and packets containing an INIT chunk, since the verification tag for these packets must be 0.

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. 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 receive packet was not encapsulated in UDP, the response packet MUST NOT be encapsulated in UDP.

Please not that in these cases a check of the of the verification tag is not possible.

4. Handling of SCTP Packets Containing an INIT Chunk Matching an Existing Association

SCTP packets containing an INIT chunk have the verification tag 0 in the common header. Therefore the verification 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.

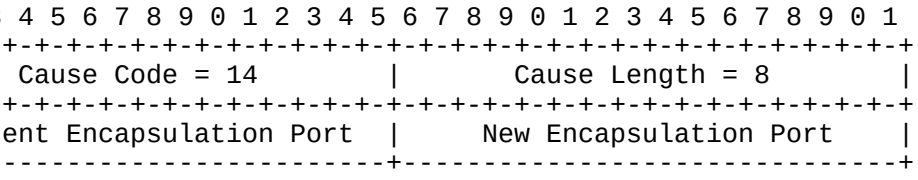
5. If the UDP encapsulation for outgoing packets towards the source address of the received SCTP packet is disabled and the received SCTP packet is not encapsulated, the processing defined in [RFC4960] MUST be performed. If a packet is sent in response, it MUST NOT be encapsulated.

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 bytes the following figure.



Cause Code: 2 bytes (unsigned integer)
This field MUST hold the IANA defined error cause code for the "Restart of an Association with New Encapsulation Port" error cause. The suggested value of this field for IANA is 14.

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 suggested value for the error cause code is 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:

Error Cause Codes

Value	Cause Code	Reference
14	Restart of an Association with New Encapsulation Port	[RFCXXXX]

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 an initial problem report.

The authors wish to thank Irene Ruengeler 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, <http://www.rfc-editor.org/info/rfc2119>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <http://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, <http://www.rfc-editor.org/info/rfc6951>.

Authors' Addresses

Michael Tuexen
Muenster University of Applied Sciences
Stegerwaldstrasse 39
48565 Steinfurt
Germany

Email: tuexen@fh-muenster.de

Randall R. Stewart
Netflix, Inc.
Chapin, SC 29036
United States

Email: randall@lakerest.net