LWIG Working Group
Internet-Draft
Intended status: Informational
Expires: April 11, 2019

C. Gomez
UPC
J. Crowcroft
University of Cambridge
M. Scharf
Hochschule Esslingen
October 8, 2018

TCP Usage Guidance in the Internet of Things (IoT)
draft-ietf-lwig-tcp-constrained-node-networks-04

Abstract

This document provides guidance on how to implement and use the Transmission Control Protocol (TCP) in Constrained-Node Networks (CNNs), which are a characterstic of the Internet of Things (IoT). Such environments require a lightweight TCP implementation and may not make use of optional functionality. This document explains a number of known and deployed techniques to simplify a TCP stack as well as corresponding tradeoffs. The objective is to help embedded developers with decisions on which TCP features to use.

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 11, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of

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

Table of Contents

1. Introduction

The Internet Protocol suite is being used for connecting Constrained-Node Networks (CNNs) to the Internet, enabling the so-called Internet of Things (IoT) [RFC7228]. In order to meet the requirements that stem from CNNs, the IETF has produced a suite of new protocols specifically designed for such environments (see e.g. [I-D.ietf-lwig-energy-efficient]). New IETF protocol stack components include the IPv6 over Low-power Wireless Personal Area Networks (6LoWPAN) adaptation layer, the IPv6 Routing Protocol for Low-power and lossy networks (RPL) routing protocol, and the Constrained Application Protocol (CoAP).

As of the writing, the main current transport layer protocols in IP-based IoT scenarios are UDP and TCP. However, TCP has been criticized (often, unfairly) as a protocol for the IoT. In fact, some TCP features are not optimal for IoT scenarios, such as relatively long header size, unsuitability for multicast, and always-confirmed data delivery. However, many typical claims on TCP unsuitability for IoT (e.g. a high complexity, connection-oriented approach incompatibility with radio duty-cycling, and spurious congestion control activation in wireless links) are not valid, can be solved, or are also found in well accepted IoT end-to-end reliability mechanisms (see [IntComp] for a detailed analysis).

At the application layer, CoAP was developed over UDP [RFC7252]. However, the integration of some CoAP deployments with existing infrastructure is being challenged by middleboxes such as firewalls, which may limit and even block UDP-based communications. This the main reason why a CoAP over TCP specification has been developed [RFC8323].

Other application layer protocols not specifically designed for CNNs are also being considered for the IoT space. Some examples include HTTP/2 and even HTTP/1.1, both of which run over TCP by default [RFC7230] [RFC7540], and the Extensible Messaging and Presence Protocol (XMPP) [RFC6120]. TCP is also used by non-IETF application-layer protocols in the IoT space such as the Message Queue Telemetry Transport (MQTT) and its lightweight variants.

TCP is a sophisticated transport protocol that includes optional functionality (e.g. TCP options) that may improve performance in some environments. However, many optional TCP extensions require

complex logic inside the TCP stack and increase the codesize and the RAM requirements. Many TCP extensions are not required for interoperability with other standard-compliant TCP endpoints. Given the limited resources on constrained devices, careful "tuning" of the TCP implementation can make an implementation more lightweight.

This document provides guidance on how to implement and use TCP in CNNs. The overarching goal is to offer simple measures to allow for lightweight TCP implementation and suitable operation in such environments. A TCP implementation following the guidance in this document is intended to be compatible with a TCP endpoint that is compliant to the TCP standards, albeit possibly with a lower performance. This implies that such a TCP client would always be able to connect with a standard-compliant TCP server, and a corresponding TCP server would always be able to connect with a standard-compliant TCP client.

This document assumes that the reader is familiar with TCP. A comprehensive survey of the TCP standards can be found in [RFC7414]. Similar guidance regarding the use of TCP in special environments has been published before, e.g., for cellular wireless networks [RFC3481].

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 [RFC2119].

3. Characteristics of CNNs relevant for TCP

3.1. Network and link properties

CNNs are defined in [RFC7228] as networks whose characteristics are influenced by being composed of a significant portion of constrained nodes. The latter are characterized by significant limitations on processing, memory, and energy resources, among others [RFC7228]. The first two dimensions pose constraints on the complexity and on the memory footprint of the protocols that constrained nodes can support. The latter requires techniques to save energy, such as radio duty-cycling in wireless devices [I-D.ietf-lwig-energy-efficient], as well as minimization of the number of messages transmitted/received (and their size).

[RFC7228] lists typical network constraints in CNN, including low achievable bitrate/throughput, high packet loss and high variability of packet loss, highly asymmetric link characteristics, severe penalties for using larger packets, limits on reachability over time,

etc. CNN may use wireless or wired technologies (e.g., Power Line Communication), and the transmission rates are typically low (e.g. below 1 Mbps).

For use of TCP, one challenge is that not all technologies in CNN may be aligned with typical Internet subnetwork design principles [RFC3819]. For instance, constrained nodes often use physical/link layer technologies that have been characterized as 'lossy', i.e., exhibit a relatively high bit error rate. Dealing with corruption loss is one of the open issues in the Internet [RFC6077].

3.2. Usage scenarios

There are different deployment and usage scenarios for CNNs. Some CNNs follow the star topology, whereby one or several hosts are linked to a central device that acts as a router connecting the CNN to the Internet. CNNs may also follow the multihop topology [RFC6606]. One key use case for the use of TCP is a model where constrained devices connect to unconstrained servers in the Internet. But it is also possible that both TCP endpoints run on constrained devices.

In constrained environments, there can be different types of devices [RFC7228]. For example, there can be devices with single combined send/receive buffer, devices with a separate send and receive buffer, or devices with a pool of multiple send/receive buffers. In the latter case, it is possible that buffers also be shared for other protocols.

When a CNN comprising one or more constrained devices and an unconstrained device communicate over the Internet using TCP, the communication possibly has to traverse a middlebox (e.g. a firewall, NAT, etc.). Figure 1 illustrates such scenario. Note that the scenario is asymmetric, as the unconstrained device will typically not suffer the severe constraints of the constrained device. The unconstrained device is expected to be mains-powered, to have high amount of memory and processing power, and to be connected to a resource-rich network.

Assuming that a majority of constrained devices will correspond to sensor nodes, the amount of data traffic sent by constrained devices (e.g. sensor node measurements) is expected to be higher than the amount of data traffic in the opposite direction. Nevertheless, constrained devices may receive requests (to which they may respond), commands (for configuration purposes and for constrained devices including actuators) and relatively infrequent firmware/software updates.

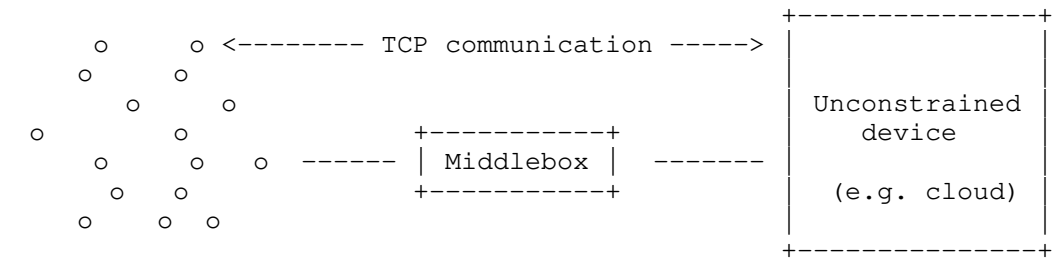

Figure 1: TCP communication between a constrained device and an unconstrained device, traversing a middlebox.

3.3. Communication and traffic patterns

IoT applications are characterized by a number of different communication patterns. The following non-comprehensive list explains some typical examples:

- Unidirectional transfers: An IoT device (e.g. a sensor) can send (repeatedly) updates to the other endpoint. Not in every case there is a need for an application response back to the IoT device.

- Request-response patterns: An IoT device receiving a request from the other endpoint, which triggers a response from the IoT device.

- Bulk data transfers: A typical example for a long file transfer would be an IoT device firmware update.

A typical communication pattern is that a constrained device communicates with an unconstrained device (cf. Figure 1). But it is also possible that constrained devices communicate amongst themselves.

4. TCP implementation and configuration in CNNs

This section explains how a TCP stack can deal with typical constraints in CNN. The guidance in this section relates to the TCP implementation and its configuration.

4.1. Path properties

4.1.1. Maximum Segment Size (MSS)

Some link layer technologies in the CNN space are characterized by a short data unit payload size, e.g. up to a few tens or hundreds of bytes. For example, the maximum frame size in IEEE 802.15.4 is 127 bytes. 6LoWPAN defined an adaptation layer to support IPv6 over IEEE 802.15.4 networks. The adaptation layer includes a fragmentation mechanism, since IPv6 requires the layer below to support an MTU of 1280 bytes [RFC2460], while IEEE 802.15.4 lacked fragmentation mechanisms. 6LoWPAN defines an IEEE 802.15.4 link MTU of 1280 bytes [RFC4944]. Other technologies, such as Bluetooth LE [RFC7668], ITU-T G.9959 [RFC7428] or DECT-ULE [RFC8105], also use 6LoWPAN-based adaptation layers in order to enable IPv6 support. These technologies do support link layer fragmentation. By exploiting this functionality, the adaptation layers that enable IPv6 over such technologies also define an MTU of 1280 bytes.

On the other hand, there exist technologies also used in the CNN space, such as Master Slave / Token Passing (TP) [RFC8163], Narrowband IoT (NB-IoT) [I-D.ietf-lpwan-overview] or IEEE 802.11ah [I-D.delcarpio-6lo-wlanah], that do not suffer the same degree of frame size limitations as the technologies mentioned above. The MTU for MS/TP is recommended to be 1500 bytes [RFC8163], the MTU in NB-IoT is 1600 bytes, and the maximum frame payload size for IEEE 802.11ah is 7991 bytes.

For the sake of lightweight implementation and operation, unless applications require handling large data units (i.e. leading to an IPv6 datagram size greater than 1280 bytes), it may be desirable to limit the MTU to 1280 bytes in order to avoid the need to support Path MTU Discovery [RFC1981].

An IPv6 datagram size exceeding 1280 bytes can be avoided by setting the TCP MSS not larger than 1220 bytes. (Note: IP version 6 is assumed.)

4.1.2. Explicit Congestion Notification (ECN)

Explicit Congestion Notification (ECN) [RFC3168] has a number of benefits that are relevant for CNNs. ECN allows a router to signal in the IP header of a packet that congestion is arising, for example when a queue size reaches a certain threshold. An ECN-enabled TCP receiver will echo back the congestion signal to the TCP sender by setting a flag in its next TCP ACK. The sender triggers congestion control measures as if a packet loss had happened. ECN can be incrementally deployed in the Internet. Guidance on configuration and usage of ECN is provided in [RFC7567]. The document [RFC8087] outlines the principal gains in terms of increased throughput,

reduced delay, and other benefits when ECN is used over a network path that includes equipment that supports Congestion Experienced (CE) marking.

ECN can reduce packet losses since congestion control measures can be applied earlier [RFC2884]. Less lost packets implies that the number of retransmitted segments decreases, which is particularly beneficial in CNNs, where energy and bandwidth resources are typically limited. Also, it makes sense to try to avoid packet drops for transactional workloads with small data sizes, which are typical for CNNs. In such traffic patterns, it is more difficult to detect packet loss without retransmission timeouts (e.g., as there may be no three duplicate ACKs). Any retransmission timeout slows down the data transfer significantly. When the congestion window of a TCP sender has a size of one segment, the TCP sender resets the retransmit timer, and the sender will only be able to send a new packet when the retransmit timer expires [RFC3168]. Effectively, the TCP sender reduces at that moment its sending rate from 1 segment per Round Trip Time (RTT) to 1 segment per RTO, which can result in a very low throughput. In addition to better throughput, ECN can also help reducing latency and jitter.

Given the benefits, more and more TCP stacks in the Internet support ECN, and it specifically makes sense to leverage ECN in controlled environments such as CNNs.

4.1.3. Explicit loss notifications

There has been a significant body of research on solutions capable of explicitly indicating whether a TCP segment loss is due to corruption, in order to avoid activation of congestion control mechanisms [ETEN] [RFC2757]. While such solutions may provide significant improvement, they have not been widely deployed and remain as experimental work. In fact, as of today, the IETF has not standardized any such solution.

4.2. TCP guidance for small windows and buffers

This section discusses TCP stacks that focus on transferring a single MSS. More general guidance is provided in Section 4.3.

4.2.1. Single-MSS stacks - benefits and issues

A TCP stack can reduce the RAM requirements by advertising a TCP window size of one MSS, and also transmit at most one MSS of unacknowledged data. In that case, both congestion and flow control implementation is quite simple. Such a small receive and send window may be sufficient for simple message exchanges in the CNN space.

However, only using a window of one MSS can significantly affect performance. A stop-and-wait operation results in low throughput for transfers that exceed the lengths of one MSS, e.g., a firmware download.

If CoAP is used over TCP with the default setting for NSTART in [RFC7252], a CoAP endpoint is not allowed to send a new message to a destination until a response for the previous message sent to that destination has been received. This is equivalent to an application-layer window size of 1. For this use of CoAP, a maximum TCP window of one MSS will be sufficient.

4.2.2. TCP options for single-MSS stacks

A TCP implementation needs to support options 0, 1 and 2 [RFC0793]. These options are sufficient for interoperability with a standard-compliant TCP endpoint, albeit many TCP stacks support additional options and can negotiate their use.

A TCP implementation for a constrained device that uses a single-MSS TCP receive or transmit window size may not benefit from supporting the following TCP options: Window scale [RFC1323], TCP Timestamps [RFC1323], Selective Acknowledgments (SACK) and SACK-Permitted [RFC2018]. Also other TCP options may not be required on a constrained device with a very lightweight implementation.

One potentially relevant TCP option in the context of CNNs is TCP Fast Open (TFO) [RFC7413]. As described in Section 5.3, TFO can be used to address the problem of traversing middleboxes that perform early filter state record deletion.

4.2.3. Delayed Acknowledgments for single-MSS stacks

TCP Delayed Acknowledgments are meant to reduce the number of transferred bytes within a TCP connection, but they may increase the time until a sender may receive an ACK. There can be interactions with stacks that use very small windows.

A device that advertises a single-MSS receive window should avoid use of delayed ACKs in order to avoid contributing unnecessary delay (of up to 500 ms) to the RTT [RFC5681], which limits the throughput and can increase the data delivery time.

A device that can send at most one MSS of data is significantly affected if the receiver uses delayed ACKs, e.g., if a TCP server or receiver is outside the CNN. One known workaround is to split the data to be sent into two segments of smaller size. A standard compliant TCP receiver will then immediately acknowledge the second

segment, which can improve throughput. This "split hack" works if the TCP receiver uses Delayed Acks, but the downside is the overhead of sending two IP packets instead of one.

4.2.4. RTO estimation for single-MSS stacks

The Retransmission Timeout (RTO) estimation is one of the fundamental TCP algorithms. There is a fundamental trade-off: A short, aggressive RTO behavior reduces wait time before retransmissions, but it also increases the probability of spurious timeouts. The latter lead to unnecessary waste of potentially scarce resources in CNNs such as energy and bandwidth. In contrast, a conservative timeout can result in long error recovery times and thus needlessly delay data delivery.

[RFC6298] describes the standard TCP RTO algorithm. If a TCP sender uses very small window size and cannot use Fast Retransmit/Fast Recovery or SACK, the Retransmission Timeout (RTO) algorithm has a larger impact on performance than for a more powerful TCP stack. In that case, RTO algorithm tuning may be considered, although careful assessment of possible drawbacks is recommended.

As an example, an adaptive RTO algorithm for CoAP over UDP has been defined [I-D.ietf-core-cocoa] that has been found to perform well in CNN scenarios [Commag].

4.3. General recommendations for TCP in CNNs

This section summarizes some widely used techniques to improve TCP, with a focus on their use in CNNs. The TCP extensions discussed here are useful in a wide range of network scenarios, including CNNs. This section is not comprehensive. A comprehensive survey of TCP extensions is published in [RFC7414].

4.3.1. Error recovery and congestion/flow control

Devices that have enough memory to allow larger TCP window size can leverage a more efficient error recovery using Fast Retransmit and Fast Recovery [RFC5681]. These algorithms work efficiently for window sizes of at least 5 MSS: If in a given TCP transmission of segments 1,2,3,4,5, and 6 the segment 2 gets lost, the sender should get an acknowledgement for segment 1 when 3 arrives and duplicate acknowledgements when 4, 5, and 6 arrive. It will retransmit segment 2 when the third duplicate ack arrives. In order to have segment 2, 3, 4, 5, and 6 sent, the window has to be at least five. With an MSS of 1220 byte, a buffer of the size of 5 MSS would require 6100 byte.

For bulk data transfers further TCP improvements may also be useful, such as limited transmit [RFC3042].

4.3.2. Selective Acknowledgments (SACK)

If a device with less severe memory and processing constraints can afford advertising a TCP window size of several MSSs, it makes sense to support the SACK option to improve performance. SACK allows a data receiver to inform the data sender of non-contiguous data blocks received, thus a sender (having previously sent the SACK-Permitted option) can avoid performing unnecessary retransmissions, saving energy and bandwidth, as well as reducing latency. SACK is particularly useful for bulk data transfers. The receiver supporting SACK will need to manage the reception of possible out-of-order received segments, requiring sufficient buffer space. SACK adds 8*n+2 bytes to the TCP header, where n denotes the number of data blocks received, up to 4 blocks. For a low number of out-of-order segments, the header overhead penalty of SACK is compensated by avoiding unnecessary retransmissions.

4.3.3. Delayed Acknowledgments

For certain traffic patterns, Delayed Acknowledgements may have a detrimental effect, as already noted in Section 4.2.3. Advanced TCP stacks may use heuristics to determine the maximum delay for an ACK. For CNNs, the recommendation depends on the expected communication patterns.

If a stack is able to deal with more than one MSS of data, it may make sense to use a small timeout or disable delayed ACKs when traffic over a CNN is expected to mostly be small messages with a size typically below one MSS. For request-response traffic between a constrained device and a peer (e.g. backend infrastructure) that uses delayed ACKs, the maximum ACK rate of the peer will be typically of one ACK every 200 ms (or even lower). If in such conditions the peer device is administered by the same entity managing the constrained device, it is recommended to disable delayed ACKs at the peer side.

In contrast, delayed ACKs allow to reduce the number of ACKs in bulk transfer type of traffic, e.g. for firmware/software updates or for transferring larger data units containing a batch of sensor readings.

Note that, in many scenarios, the peer that a constrained device communicates with will be a general purpose system that communicates with both constrained and unconstrained devices. Since delayed ACKs are often configured through system-wide parameters, delayed ACKs behavior at the peer will be the same regardless of the nature of the

endpoints it talks to. Such a peer will typically have delayed ACKs enabled.

5. TCP usage recommendations in CNNs

This section discusses how a TCP stack can be used by applications that are developed for CNN scenarios. These remarks are by and large independent of how TCP is exactly implemented.

5.1. TCP connection initiation

In the constrained device to unconstrained device scenario illustrated above, a TCP connection is typically initiated by the constrained device, in order for this device to support possible sleep periods to save energy.

5.2. Number of concurrent connections

TCP endpoints with a small amount of RAM may only support a small number of connections. Each TCP connection requires storing a number of variables in the Transmission Control Block (TCB). Depending on the internal TCP implementation, each connection may result in further overhead, and they may compete for scarce resources.

A careful application design may try to keep the number of concurrent connections as small as possible. A client can for instance limit the number of simultaneous open connections that it maintains to a given server. Multiple connections could for instance be used to avoid the "head-of-line blocking" problem in an application transfer. However, in addition to comsuming resources, using multiple connections can also cause undesirable side effects in congested networks. As example, the HTTP/1.1 specification encourages clients to be conservative when opening multiple connections [RFC7230].

Being conservative when opening multiple TCP connections is of particular importance in Constrained-Node Networks.

5.3. TCP connection lifetime

In order to minimize message overhead, it makes sense to keep a TCP connection open as long as the two TCP endpoints have more data to send. If applications exchange data rather infrequently, i.e., if TCP connections would stay idle for a long time, the idle time can result in problems. For instance, certain middleboxes such as firewalls or NAT devices are known to delete state records after an inactivity interval typically in the order of a few minutes [RFC6092]. The timeout duration used by a middlebox implementation may not be known to the TCP endpoints.

In CCNs, such middleboxes may e.g. be present at the boundary between the CCN and other networks. If the middlebox can be optimized for CCN use cases, it makes sense to increase the initial value for filter state inactivity timers to avoid problems with idle connections. Apart from that, this problem can be dealt with by different connection handling strategies, each having pros and cons.

One approach for infrequent data transfer is to use short-lived TCP connections. Instead of trying to maintain a TCP connection for long time, possibly short-lived connections can be opened between two endpoints, which are closed if no more data needs to be exchanged. For use cases that can cope with the additional messages and the latency resulting from starting new connections, it is recommended to use a sequence of short-lived connections, instead of maintaining a single long-lived connection.

This overhead could be reduced by TCP Fast Open (TFO) [RFC7413], which is an experimental TCP extension. TFO allows data to be carried in SYN (and SYN-ACK) segments, and to be consumed immediately by the receceiving endpoint. This reduces the overhead compared to the traditional three-way handshake to establish a TCP connection. For security reasons, the connection initiator has to request a TFO cookie from the other endpoint. The cookie, with a size of 4 or 16 bytes, is then included in SYN packets of subsequent connections. The cookie needs to be refreshed (and obtained by the client) after a certain amount of time. Nevertheless, TFO is more efficient than frequently opening new TCP connections with the traditional three-way handshake, as long as the cookie can be reused in subsequent connections.

Another approach is to use long-lived TCP connections with application-layer heartbeat messages. Various application protocols support such heartbeat messages. Periodic heartbeats requires transmission of packets, but they also allow aliveness checks at application level. In addition, they can prevent early filter state record deletion in middleboxes. In general, it makes sense realize aliveness checks at the highest protocol layer possible that is meaningful to the application, in order to maximize the depth of the aliveness check.

A TCP implementation may also be able to send "keep-alive" segments to test a TCP connection. According to [RFC1122], "keep-alives" are an optional TCP mechanism that is turned off by default, i.e., an application must explicitly enable it for a TCP connection. The interval between "keep-alive" messages must be configurable and it must default to no less than two hours. With this large timeout, TCP keep-alive messages are not very useful to avoid deletion of filter state records in middleboxes such as firewalls.

6. Security Considerations

Best current practise for securing TCP and TCP-based communication also applies to CNN. As example, use of Transport Layer Security (TLS) is strongly recommended if it is applicable.

There are also TCP options which can improve TCP security. Examples include the TCP MD5 signature option [RFC2385] and the TCP Authentication Option (TCP-AO) [RFC5925]. However, both options add overhead and complexity. The TCP MD5 signature option adds 18 bytes to every segment of a connection. TCP-AO typically has a size of 16-20 bytes.

For the mechanisms discussed in this document, the corresponding considerations apply. For instance, if TFO is used, the security considerations of [RFC7413] apply.

Constrained devices are expected to support smaller TCP window sizes than less limited devices. In such conditions, segment retransmission triggered by RTO expiration is expected to be relatively frequent, due to lack of (enough) duplicate ACKs, especially when a constrained device uses a single-MSS window size. For this reason, constrained devices running TCP may appear as particularly appealing victims of the so-called "shrew" Denial of Service (DoS) attack [shrew], whereby one or more sources generate a packet spike targetted to coincide with consecutive RTO-expiration-triggered retry attempts of a victim node. Note that the attack may be performed by Internet-connected devices, including constrained devices in the same CNN as the victim, as well as remote ones. Mitigation techniques include RTO randomization and attack blocking by routers able to detect shrew attacks based on their traffic pattern.

7. Acknowledgments

Carles Gomez has been funded in part by the Spanish Government (Ministerio de Educacion, Cultura y Deporte) through the Jose Castillejo grant CAS15/00336 and by European Regional Development Fund (ERDF) and the Spanish Government through project TEC2016-79988-P, AEI/FEDER, UE. Part of his contribution to this work has been carried out during his stay as a visiting scholar at the Computer Laboratory of the University of Cambridge.

The authors appreciate the feedback received for this document. The following folks provided comments that helped improve the document: Carsten Bormann, Zhen Cao, Wei Genyu, Ari Keranen, Abhijan Bhattacharyya, Andres Arcia-Moret, Yoshifumi Nishida, Joe Touch, Fred Baker, Nik Sultana, Kerry Lynn, Erik Nordmark, Markku Kojo, and

Hannes Tschofenig. Simon Brummer provided details, and kindly performed RAM and ROM usage measurements, on the RIOT TCP implementation. Xavi Vilajosana provided details on the OpenWSN TCP implementation. Rahul Jadhav provided details on the uIP TCP implementation.

8. Annex. TCP implementations for constrained devices

This section overviews the main features of TCP implementations for constrained devices. The survey is limited to open source stacks with small footprint. It is not meant to be all-encompassing. For more powerful embedded systems (e.g., with 32-bit processors), there are further stacks that comprehensively implement TCP. On the other hand, please be aware that this Annex is based on information available as of the writing.

8.1. uIP

uIP is a TCP/IP stack, targetted for 8 and 16-bit microcontrollers, which pioneered TCP/IP implementations for constrained devices. uIP has been deployed with Contiki and the Arduino Ethernet shield. A code size of ~5 kB (which comprises checksumming, IP, ICMP and TCP) has been reported for uIP [Dunk].

uIP uses same buffer both incoming and outgoing traffic, with has a size of a single packet. In case of a retransmission, an application must be able to reproduce the same user data that had been transmitted.

The MSS is announced via the MSS option on connection establishment and the receive window size (of one MSS) is not modified during a connection. Stop-and-wait operation is used for sending data. Among other optimizations, this allows to avoid sliding window operations, which use 32-bit arithmetic extensively and are expensive on 8-bit CPUs.

Contiki uses the "split hack" technique (see Section 4.2.3) to avoid delayed ACKs for senders using a single MSS.

8.2. lwIP

lwIP is a TCP/IP stack, targetted for 8- and 16-bit microcontrollers. lwIP has a total code size of ~14 kB to ~22 kB (which comprises memory management, checksumming, network interfaces, IP, ICMP and TCP), and a TCP code size of ~9 kB to ~14 kB [Dunk].

In contrast with uIP, lwIP decouples applications from the network stack. lwIP supports a TCP transmission window greater than a single

segment, as well as buffering of incoming and outcoming data. Other implemented mechanisms comprise slow start, congestion avoidance, fast retransmit and fast recovery. SACK and Window Scale have been recently added to lwIP.

8.3. RIOT

The RIOT TCP implementation (called GNRC TCP) has been designed for Class 1 devices [RFC 7228]. The main target platforms are 8- and 16-bit microcontrollers. GNRC TCP offers a similar function set as uIP, but it provides and maintains an independent receive buffer for each connection. In contrast to uIP, retransmission is also handled by GNRC TCP. GNRC TCP uses a single-MSS window size, which simplifies the implementation. The application programmer does not need to know anything about the TCP internals, therefore GNRC TCP can be seen as a user-friendly uIP TCP implementation.

The MSS is set on connections establishment and cannot be changed during connection lifetime. GNRC TCP allows multiple connections in parallel, but each TCB must be allocated somewhere in the system. By default there is only enough memory allocated for a single TCP connection, but it can be increased at compile time if the user needs multiple parallel connections.

The RIOT TCP implementation does not currently support classic POSIX sockets. However, it supports an interface that has been inspired by POSIX.

8.4. TinyOS

TinyOS was important as platform for early constrained devices. TinyOS has an experimental TCP stack that uses a simple nonblocking library-based implementation of TCP, which provides a subset of the socket interface primitives. The application is responsible for buffering. The TCP library does not do any receive-side buffering. Instead, it will immediately dispatch new, in-order data to the application and otherwise drop the segment. A send buffer is provided so that the TCP implementation can automatically retransmit missing segments. Multiple TCP connections are possible. Recently there has been little further work on the stack.

8.5. FreeRTOS

FreeRTOS is a real-time operating system kernel for embedded devices that is supported by 16- and 32-bit microprocessors. Its TCP implementation is based on multiple-MSS window size, although a 'Tiny-TCP' option, which is a single-MSS variant, can be enabled.

Delayed ACKs are supported, with a 20-ms Delayed ACK timer as a technique intended 'to gain performance'.

8.6. uC/OS

uC/OS is a real-time operating system kernel for embedded devices, which is maintained by Micrium. uC/OS is intended for 8-, 16- and 32-bit microprocessors. The uC/OS TCP implementation supports a multiple-MSS window size.

8.7. Summary

		uIP	lwIP orig	lwIP 2.0	RIOT	TinyOS	FreeRTOS	uC/OS
Memory	Code size(kB)	<5 (a)	~9 to ~14 (T1)	~40 (b)	<7 (T3)	N/A	<9.2 (T2)	N/A
TCP features	Win size(MSS)	1	Mult.	Mult.	1	Mult.	Mult.	Mult.
	Slow start	No	Yes	Yes	No	Yes	No	Yes
	Fast rec/retx	No	Yes	Yes	No	Yes	No	Yes
	Keep-alive	No	No	Yes	No	No	Yes	Yes
	Win. Scale	No	No	Yes	No	No	Yes	No
	TCP timest.	No	No	Yes	No	No	Yes	No
	SACK	No	No	Yes	No	No	Yes	No
	Del. ACKs	No	Yes	Yes	No	No	Yes	Yes
	Socket	No	No	Optional	(I)	Subset	Yes	Yes
	Concur. Conn.	Yes	Yes	Yes	Yes	Yes	Yes	Yes

(T1) = TCP-only, on x86 and AVR platforms
(T2) = TCP-only, on ARM Cortex-M platform
(T3) = TCP-only, on ARM Cortex-M0+ platform (NOTE: RAM usage for the same platform is ~2.5 kB for one TCP connection plus ~1.2 kB for each additional connection)
(a) = includes IP, ICMP and TCP on x86 and AVR platforms
(b) = the whole protocol stack on mbed
(I) = interface inspired by POSIX
Mult. = Multiple
N/A = Not Available

Figure 2: Summary of TCP features for differrent lightweight TCP implementations. None of the implementations considered in this Annex support ECN or TFO.

9. Annex. Changes compared to previous versions

RFC Editor: To be removed prior to publication

9.1. Changes between -00 and -01

- Changed title and abstract
- Clarification that communcation with standard-compliant TCP endpoints is required, based on feedback from Joe Touch
- Additional discussion on communication patters
- Numerous changes to address a comprehensive review from Hannes Tschofenig
- Reworded security considerations
- Additional references and better distinction between normative and informative entries
- Feedback from Rahul Jadhav on the uIP TCP implementation
- Basic data for the TinyOS TCP implementation added, based on source code analysis

9.2. Changes between -01 and -02

- Added text to the Introduction section, and a reference, on traditional bad perception of TCP for IoT
- Added sections on FreeRTOS and uC/OS
- Updated TinyOS section
- Updated summary table
- Reorganized Section 4 (single-MSS vs multiple-MSS window size), some content now also in new Section 5

9.3. Changes between -02 and -03

- Rewording to better explain the benefit of ECN
- Additional context information on the surveyed implementations
- Added details, but removed "Data size" raw, in the summary table
- Added discussion on shrew attacks

9.4. Changes between -03 and -04

- Addressing the remaining TODOs
- Alignment of the wording on TCP "keep-alives" with related discussions in the IETF transport area
- Added further discussion on delayed ACKs
- Removed OpenWSN subsection from the Annex

10. References

10.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[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>.

[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>.

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, DOI 10.17487/RFC2018, October 1996, <https://www.rfc-editor.org/info/rfc2018>.

[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>.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <https://www.rfc-editor.org/info/rfc2460>.

[RFC3042] Allman, M., Balakrishnan, H., and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, DOI 10.17487/RFC3042, January 2001, <https://www.rfc-editor.org/info/rfc3042>.

[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>.

[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>.

[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>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.

10.2. Informative References

[Commag] A. Betzler, C. Gomez, I. Demirkol, J. Paradells, "CoAP Congestion Control for the Internet of Things", IEEE Communications Magazine, June 2016.

[Dunk] A. Dunkels, "Full TCP/IP for 8-Bit Architectures", 2003.

[ETEN] R. Krishnan et al, "Explicit transport error notification (ETEN) for error-prone wireless and satellite networks", Computer Networks 2004.

[I-D.delcarpio-6lo-wlanah]
Vega, L., Robles, I., and R. Morabito, "IPv6 over 802.11ah", draft-delcarpio-6lo-wlanah-01 (work in progress), October 2015.

[I-D.ietf-core-coap-tcp-tls]
Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", draft-ietf-core-coap-tcp-tls-11 (work in progress), December 2017.

[I-D.ietf-core-cocoa]
Bormann, C., Betzler, A., Gomez, C., and I. Demirkol, "CoAP Simple Congestion Control/Advanced", draft-ietf-core-cocoa-03 (work in progress), February 2018.

[I-D.ietf-lpwan-overview]
Farrell, S., "LPWAN Overview", draft-ietf-lpwan-overview-10 (work in progress), February 2018.

[I-D.ietf-lwig-energy-efficient]
Gomez, C., Kovatsch, M., Tian, H., and Z. Cao, "Energy-Efficient Features of Internet of Things Protocols", draft-ietf-lwig-energy-efficient-08 (work in progress), October 2017.

[IntComp] C. Gomez, A. Arcia-Moret, J. Crowcroft, "TCP in the Internet of Things: from ostracism to prominence", IEEE Internet Computing, January-February 2018.

[RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, DOI 10.17487/RFC1981, August 1996, <https://www.rfc-editor.org/info/rfc1981>.

[RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, DOI 10.17487/RFC2385, August 1998, <https://www.rfc-editor.org/info/rfc2385>.

[RFC2757] Montenegro, G., Dawkins, S., Kojo, M., Magret, V., and N. Vaidya, "Long Thin Networks", RFC 2757, DOI 10.17487/RFC2757, January 2000, <https://www.rfc-editor.org/info/rfc2757>.

[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>.

[RFC3481] Inamura, H., Ed., Montenegro, G., Ed., Ludwig, R., Gurtov, A., and F. Khafizov, "TCP over Second (2.5G) and Third (3G) Generation Wireless Networks", BCP 71, RFC 3481, DOI 10.17487/RFC3481, February 2003, <https://www.rfc-editor.org/info/rfc3481>.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <https://www.rfc-editor.org/info/rfc4944>.

[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>.

[RFC6092] Woodyatt, J., Ed., "Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for Providing Residential IPv6 Internet Service", RFC 6092, DOI 10.17487/RFC6092, January 2011, <https://www.rfc-editor.org/info/rfc6092>.

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, DOI 10.17487/RFC6120, March 2011, <https://www.rfc-editor.org/info/rfc6120>.

[RFC6606] Kim, E., Kaspar, D., Gomez, C., and C. Bormann, "Problem Statement and Requirements for IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing", RFC 6606, DOI 10.17487/RFC6606, May 2012, <https://www.rfc-editor.org/info/rfc6606>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7414] Duke, M., Braden, R., Eddy, W., Blanton, E., and A. Zimmermann, "A Roadmap for Transmission Control Protocol (TCP) Specification Documents", RFC 7414, DOI 10.17487/RFC7414, February 2015, <https://www.rfc-editor.org/info/rfc7414>.

[RFC7428] Brandt, A. and J. Buron, "Transmission of IPv6 Packets over ITU-T G.9959 Networks", RFC 7428, DOI 10.17487/RFC7428, February 2015, <https://www.rfc-editor.org/info/rfc7428>.

[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>.

[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>.

[RFC7668] Nieminen, J., Savolainen, T., Isomaki, M., Patil, B., Shelby, Z., and C. Gomez, "IPv6 over BLUETOOTH(R) Low Energy", RFC 7668, DOI 10.17487/RFC7668, October 2015, <https://www.rfc-editor.org/info/rfc7668>.

[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>.

[RFC8105] Mariager, P., Petersen, J., Ed., Shelby, Z., Van de Logt, M., and D. Barthel, "Transmission of IPv6 Packets over Digital Enhanced Cordless Telecommunications (DECT) Ultra Low Energy (ULE)", RFC 8105, DOI 10.17487/RFC8105, May 2017, <https://www.rfc-editor.org/info/rfc8105>.

[RFC8163] Lynn, K., Ed., Martocci, J., Neilson, C., and S. Donaldson, "Transmission of IPv6 over Master-Slave/Token-Passing (MS/TP) Networks", RFC 8163, DOI 10.17487/RFC8163, May 2017, <https://www.rfc-editor.org/info/rfc8163>.

[RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <https://www.rfc-editor.org/info/rfc8323>.

[shrew] A. Kuzmanovic, E. Knightly, "Low-Rate TCP-Targeted Denial of Service Attacks", SIGCOMM'03 2003.

Authors' Addresses

Carles Gomez
UPC
C/Esteve Terradas, 7
Castelldefels 08860
Spain

Email: carlesgo@entel.upc.edu

Jon Crowcroft
University of Cambridge
JJ Thomson Avenue
Cambridge, CB3 0FD
United Kingdom

Email: jon.crowcroft@cl.cam.ac.uk

Michael Scharf
Hochschule Esslingen
Flandernstr. 101
Esslingen 73732
Germany

Email: michael.scharf@hs-esslingen.de

Network Working Group
Internet-Draft
Obsoletes: 5562 (if approved)
Intended status: Experimental
Expires: 4 June 2024

M. Bagnulo
UC3M
B. Briscoe, Ed.
Independent

2 December 2023

ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control Packets
draft-ietf-tcpm-generalized-ecn-15

Abstract

This document specifies an experimental modification to ECN when used with TCP. It allows the use of ECN in the IP header of the following TCP packets: SYNs, SYN/ACKs, pure ACKs, Window probes, FINs, RSTs and retransmissions. This specification obsoletes RFC5562, which described a different way to use ECN on SYN/ACKs alone.

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 4 June 2024.

Copyright Notice

Copyright (c) 2023 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction

RFC 3168 [RFC3168] specifies support of Explicit Congestion Notification (ECN) in IP (v4 and v6). By using the ECN capability, network elements (e.g. routers, switches) performing Active Queue Management (AQM) can use ECN marks instead of packet drops to signal congestion to the endpoints of a communication. This results in lower packet loss and increased performance. RFC 3168 also specifies support for ECN in TCP, but solely on data packets. For various reasons it precludes the use of ECN on TCP control packets (TCP SYN, TCP SYN-ACK, pure ACKs, Window probes) and on retransmitted packets. RFC 3168 is silent about the use of ECN on RST and FIN packets. RFC 5562 [RFC5562] is an experimental modification to ECN that enables ECN support for TCP SYN-ACK packets.

This document defines an experimental modification to ECN [RFC3168] that shall be called ECN++. It enables ECN support on all the aforementioned types of TCP packet. RFC 5562 (which was called ECN+) is obsoleted by the present specification, because it has the same goal of enabling ECT, but on only one type of control packet. The mechanisms proposed in this document have been defined conservatively and with safety in mind, possibly in some cases at the expense of performance.

ECN++ uses a sender-only deployment model. It works whether the two ends of the TCP connection use classic ECN feedback [RFC3168] or the updated scheme called Accurate ECN feedback (AccECN [I-D.ietf-tcpm-accurate-ecn]). {This is written assuming that AccECN will have been published as an RFC before ECN++, and that AccECN does indeed update RFC 3168, as intended at the time of writing. This note to be removed by the RFC Editor.}

Using ECN on initial SYN packets provides significant benefits, as we describe in the next subsection. However, only AccECN provides a way to feed back whether the SYN was CE marked, and RFC 3168 does not. Therefore, this spec recommends that implementers of ECN++ also implement AccECN. Conversely, if AccECN (or an equivalent safety mechanism) is not implemented with ECN++, this specification rules out ECN on the SYN.

ECN++ is designed for compatibility with a number of latency improvements to TCP such as TCP Fast Open (TFO [RFC7413]), initial window of 10 SMSS (IW10 [RFC6928]) and Low latency Low Loss Scalable Transport (L4S) [RFC9330], but they can all be implemented and deployed independently. [RFC8311] is a standards track procedural device that relaxes requirements in RFC 3168 and other standards track RFCs that would otherwise preclude the experimental modifications needed for ECN++ and other ECN experiments.

1.1. Motivation

The absence of ECN support on TCP control packets and retransmissions has a potential harmful effect. In any ECN deployment, non-ECN-capable packets suffer a penalty when they traverse a congested bottleneck. For instance, with a drop probability of 1%, 1% of connection attempts suffer a timeout of about 1 second before the SYN is retransmitted, which is highly detrimental to the performance of short flows. TCP control packets, particularly TCP SYNs and SYN-ACKs, are important for performance, so dropping them is best avoided.

Not using ECN on control packets can be particularly detrimental to performance in environments where the ECN marking level is high. For example, [judd-nsdi] shows that in a controlled private data centre (DC) environment where ECN is used (in conjunction with DCTCP [RFC8257]), the probability of being able to establish a new connection using a non-ECN SYN packet drops to close to zero even when there are only 16 ongoing TCP flows transmitting at full speed. The issue is that DCTCP exhibits a much more aggressive response to packet marking (which is why it is only applicable in controlled environments). This leads to a high marking probability for ECN-capable packets, and in turn a high drop probability for non-ECN

packets. Therefore non-ECN SYNs are dropped aggressively, rendering it nearly impossible to establish a new connection in the presence of even mild traffic load.

Finally, there are ongoing experimental efforts to promote the adoption of a slightly modified variant of DCTCP (and similar congestion controls) over the Internet to achieve low latency, low loss and scalable throughput (L4S) for all communications [RFC9330]. In such an approach, L4S packets identify themselves using an ECN codepoint [RFC9331]. With L4S, preventing TCP control packets from obtaining the benefits of ECN would not only expose them to the prevailing level of congestion loss, but it would also classify them into a different queue. Then only L4S data packets would be classified into the L4S queue that is expected to have lower latency, while the packets controlling and retransmitting these data packets would still get stuck behind the queue induced by non-L4S-enabled TCP traffic.

1.2. Experiment Goals

The goal of the experimental modifications defined in this document is to allow the use of ECN on all TCP packets. Experiments are expected in the public Internet as well as in controlled environments to understand the following issues:

* How SYNs, Window probes, pure ACKs, FINs, RSTs and retransmissions that carry the ECT(0), ECT(1) or CE codepoints are processed by the TCP endpoints and the network (including routers, firewalls and other middleboxes). In particular we would like to learn if these packets are frequently blocked or if these packets are usually forwarded and processed.

* The scale of deployment of the different flavours of ECN, including [RFC3168], [RFC5562], [RFC3540] and [I-D.ietf-tcpm-accurate-ecn].

* How much the performance of TCP communications is improved by allowing ECN marking of each packet type.

* To identify any issues (including security issues) raised by enabling ECN marking of these packets.

* To conduct the specific experiments identified in the text by the strings "EXPERIMENTATION NEEDED" or "MEASUREMENTS NEEDED".

The data gathered through the experiments described in this document, particularly under the first 2 bullets above, will help in the redesign of the final mechanism (if needed) for adding ECN support to the different packet types considered in this document.

Success criteria: The experiment will be a success if we obtain enough data to have a clearer view of the deployability and benefits of enabling ECN on all TCP packets, as well as any issues. If the results of the experiment show that it is feasible to deploy such changes; that there are gains to be achieved through the changes described in this specification; and that no other major issues may interfere with the deployment of the proposed changes; then it would be reasonable to adopt the proposed changes in a standards track specification that would update RFC 3168.

1.3. Document Structure

The remainder of this document is structured as follows. In Section 2, we present the terminology used in the rest of the document. In Section 3, we specify the modifications to provide ECN support to TCP SYNs, pure ACKs, Window probes, FINs, RSTs and retransmissions. We describe both the network behaviour and the endpoint behaviour. Section 5 discusses variations of the specification that will be necessary to interwork with a number of popular variants or derivatives of TCP. RFC 3168 provides a number of specific reasons why ECN support is not appropriate for each packet type. In Section 4, we revisit each of these arguments for each packet type to justify why it is reasonable to conduct this experiment.

2. Terminology

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

Pure ACK: A TCP segment with the ACK flag set and no data payload.

SYN: A TCP segment with the SYN (synchronize) flag set.

Window probe: Defined in [RFC9293], a window probe is a regular TCP segment, but with only one octet of new data that is sent to learn if the receive window is still zero.

FIN: A TCP segment with the FIN (finish) flag set.

RST: A TCP segment with the RST (reset) flag set.

Retransmission: A TCP segment that has been retransmitted by the TCP sender.

TCP client: The initiating end of a TCP connection. Also called the initiator.

TCP server: The responding end of a TCP connection. Also called the responder or listener.

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 endpoints support ECN. When this specification says the sender sets an ECT codepoint, by default it means ECT(0). Optionally, it could mean ECT(1), which has been redefined for use by L4S experiments [RFC8311] [RFC9331].

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 [RFC3168]. A node sets an increasing proportion of ECT packets to CE as the level of congestion increases.

3. Specification

The experimental ECN++ changes to the specification of TCP over ECN [RFC3168] defined here primarily alter the behaviour of the sending host for each half-connection. However, there are subsections for forwarding elements and receivers below, which recommend that they accept the new packets - they should do already, but might not. This will prompt implementers to check the receive side code while they are altering the send-side code. All changes can be deployed at each endpoint independently of others and independent of any network behaviour.

The feedback behaviour at the receiver depends on whether classic ECN TCP feedback [RFC3168] or Accurate ECN (AccECN) TCP feedback [I-D.ietf-tcpm-accurate-ecn] has been negotiated. Nonetheless, neither receiver feedback behaviour is altered by the present specification.

3.1. Network (e.g. Firewall) Behaviour

Previously the specification of ECN for TCP [RFC3168] required the sender to set not-ECT on TCP control packets and retransmissions. Some readers of RFC 3168 might have erroneously interpreted this as a requirement for firewalls, intrusion detection systems, etc. to check and enforce this behaviour. Section 4.3 of [RFC8311] updates RFC 3168 to remove this ambiguity. It requires firewalls or any intermediate nodes not to treat certain types of ECN-capable TCP segment differently (except potentially in one attack scenario). This is likely to only involve a firewall rule change in a fraction of cases (at most 0.4% of paths according to the tests reported in Section 4.2.2).

In case a TCP sender encounters a middlebox blocking ECT on certain TCP segments, the specification below includes behaviour to fall back to non-ECN. However, this loses the benefit of ECN on control packets. So operators are RECOMMENDED to alter their firewall rules to comply with the requirement referred to above (section 4.3 of [RFC8311]).

3.2. Sender Behaviour

For each type of control packet or retransmission, the following sections detail changes to the sender's behaviour in two respects: i) whether it sets ECT; and ii) its response to congestion feedback. Table 1 summarises these two behaviours for each type of packet, but the relevant subsection below should be referred to for the detailed behaviour. The subsection on the SYN is more complex than the others, because it has to include fall-back behaviour if the ECT packet appears not to have got through, and caching of the outcome to detect persistent failures.

TCP packet type	ECN field if AccECN f/b negotiated*	ECN field if RFC3168 f/b negotiated*	Congestion Response
SYN	ECT**	not-ECT	If AccECN, reduce IW
SYN-ACK	ECT	ECT	Reduce IW
Pure ACK	ECT	not-ECT	If AccECN, usual cwnd response and optionally [RFC5690]
W Probe	ECT	ECT	Usual cwnd response
FIN	ECT	ECT	None or optionally [RFC5690]
RST	ECT	ECT	N/A
Re-XMT	ECT	ECT	Usual cwnd response

W Probe and Re-XMT stand for Window Probe and Retransmission.
* For a SYN, "negotiated" means "requested".
** AccECN or equivalent safety (see Section 3.2.1.1.2)

Table 1: Summary of sender behaviour. In each case the relevant section below should be referred to for the detailed behaviour

It can be seen that we recommend against the sender setting ECT on the SYN if it is not requesting AccECN feedback. Therefore it is RECOMMENDED that the AccECN specification [I-D.ietf-tcpm-accurate-ecn] is implemented, along with the ECN++ experiment, because it is expected that ECT on the SYN will give the most significant performance gain, particularly for short flows.

Nonetheless, this specification also caters for the case where an ECN++ TCP sender is not using AccECN. This could be because it does not support AccECN or because its peer does not (AccECN can only be used if both ends of a connection support it).

Note that Table 1 does not imply any obligation to set any packet to ECT. ECN++ removes the restrictions that RFC 3168 places against setting ECT on these types of packets, and an implementation would normally be expected to take advantage of this, but it does not have

to. Therefore, an implementation of the ECN++ experiment would be compliant if, for instance, it set ECT on some types of control packets but not others. If it did not set ECT on any control packets or retransmissions, it would not be compliant.

3.2.1. SYN (Send)

3.2.1.1. Setting ECT on the SYN

With classic [RFC3168] ECN feedback, the SYN was not expected to be ECN-capable, so the flag provided to feed back congestion was put to another use (it is used in combination with other flags to indicate that the responder supports ECN). In contrast, Accurate ECN (AccECN) feedback [I-D.ietf-tcpm-accurate-ecn] provides a codepoint in the SYN-ACK for the responder to feed back whether the SYN arrived marked CE. Therefore the setting of the IP/ECN field on the SYN is specified separately for each case in the following two subsections.

3.2.1.1.1. ECN++ TCP Client also Supports AccECN

For the ECN++ experiment, if the SYN is requesting AccECN feedback, the TCP sender will also set ECT on the SYN. Section 4.3 of [RFC8311] allows an ECN experiment to use ECT on a SYN, by updating Section 6.1.1 of RFC 3168 (which prohibited it).

3.2.1.1.2. ECN++ TCP Client does not Support AccECN

If the SYN sent by a TCP initiator does not attempt to negotiate Accurate ECN feedback, or does not use an equivalent safety mechanism, it MUST NOT set ECT on a SYN. This follows section 6.1.1 of [RFC3168], because, in this case, it would not be safe to use the extra flexibility to set ECT on a SYN that [RFC8311] allows.

The only envisaged examples of "equivalent safety mechanisms" are: a) some future TCP ECN feedback protocol, perhaps evolved from AccECN, that feeds back CE marking on a SYN; b) setting the initial window to 1 SMSS. IW=1 is NOT RECOMMENDED because it could degrade performance, but might be appropriate for certain lightweight TCP implementations.

See Section 4.2 for discussion and rationale.

If the TCP initiator does not set ECT on the SYN, the rest of Section 3.2.1 does not apply.

3.2.1.2. Caching where to use ECT on SYNs

This subsection only applies if the ECN++ TCP client sets ECT on the SYN and supports AccECN.

Until AccECN servers become widely deployed, a TCP initiator that sets ECT on a SYN (which typically implies the same SYN also requests AccECN, as above) SHOULD also maintain a cache entry per server to record servers that it is not worth sending an ECT SYN to, e.g. because they do not support AccECN and therefore have no logic for congestion markings on the SYN. Mobile hosts MAY maintain a cache entry per access network to record 'non-ECT SYN' entries against proxies (see Section 4.2.3). This cache can be implemented as part of the shared state across multiple TCP connections, if it is following [RFC9040].

Subsequently the initiator will not set ECT on a SYN to such a server or proxy, but it can still always request AccECN support (because the response will state any earlier stage of ECN evolution that the server supports with no performance penalty). If a server subsequently upgrades to support AccECN, the initiator will discover this as soon as it next connects, then it can remove the server from its cache and subsequently always set ECT for that server.

The client can limit the size of its cache of 'non-ECT SYN' servers. Then, while AccECN is not widely deployed, it will only cache the 'non-ECT SYN' servers that are most used and most recently used by the client. As the client accesses servers that have been expelled from its cache, it will simply use ECT on the SYN by default.

Servers that do not support ECN as a whole do not need to be recorded separately from non-support of AccECN because the response to a request for AccECN immediately states which stage in the evolution of ECN the server supports (AccECN [I-D.ietf-tcpm-accurate-ecn], classic ECN [RFC3168] or no ECN).

The above strategy is named "optimistic ECT and cache failures". It is believed to be sufficient based on three measurement studies and assumptions detailed in Section 4.2.3. However, Section 4.2.3 gives two other strategies and the choice between them depends on the implementer's goals (e.g., see Section 5.3 if using L4S) and the deployment prevalence of ECN variants in the network and on servers, not to mention the prevalence of some significant bugs.

If the initiator times out without seeing a SYN-ACK, it will separately cache this fact (see fall-back in Section 3.2.1.4 for details).

3.2.1.3. SYN Congestion Response

As explained above, this subsection only applies if the ECN++ TCP client sets ECT on the initial SYN.

If the SYN-ACK returned to the TCP initiator confirms that the server supports AccECN, it will also be able to indicate whether or not the SYN was CE-marked. If the SYN was CE-marked, and if the initial window is greater than 1 MSS, then, the initiator MUST reduce its Initial Window (IW) and SHOULD reduce it to 1 SMSS (sender maximum segment size). The rationale is the same as that for the response to CE on a SYN-ACK (Section 4.3.2).

If the initiator has set ECT on the SYN and if the SYN-ACK shows that the server does not support feedback of a CE on the SYN (e.g. it does not support AccECN) and if the initial congestion window of the initiator is greater than 1 MSS, then the TCP initiator MUST conservatively reduce its Initial Window and SHOULD reduce it to 1 SMSS. A reduction to greater than 1 SMSS MAY be appropriate (see Section 4.2.1). Conservatism is necessary because the SYN-ACK cannot show whether the SYN was CE-marked.

If the TCP initiator (host A) receives a SYN from the remote end (host B) after it has sent a SYN to B, it indicates the (unusual) case of a simultaneous open. Host A will respond with a SYN-ACK. Host A will probably then receive a SYN-ACK in response to its own SYN, after which it can follow the appropriate one of the two paragraphs above.

In all the above cases, the initiator does not have to back off its retransmission timer as it would in response to a timeout following no response to its SYN [RFC6298], because both the SYN and the SYN-ACK have been successfully delivered through the network. Also, the initiator does not need to exit slow start or reduce ssthresh, which is not even required when a SYN is lost [RFC5681].

If an initial window of more than 3 segments is implemented (e.g. IW10 [RFC6928]), Section 5 gives additional recommendations.

3.2.1.4. Fall-Back Following No Response to an ECT SYN

As explained above, this subsection only applies if the ECN++ TCP client also sets ECT on the initial SYN.

An ECT SYN might be lost due to an over-zealous path element (or server) blocking ECT packets that do not conform to RFC 3168. Some evidence of this was found in a 2014 study [ecn-pam], but in a more recent study using 2017 data [Mandalari18] extensive measurements

found no case where ECT on TCP control packets was treated any differently from ECT on TCP data packets. Loss is commonplace for numerous other reasons, e.g. congestion loss at a non-ECN queue on the forward or reverse path, transmission errors, etc. Alternatively, the cause of the loss might be the associated attempt to negotiate AccECN, or possibly other unrelated options on the SYN.

Therefore, if the timer expires after the TCP initiator has sent the first ECT SYN, it SHOULD make one more attempt to retransmit the SYN with ECT set (backing off the timer as usual). If the retransmission timer expires again, it SHOULD retransmit the SYN with the not-ECT codepoint in the IP header, to expedite connection set-up. If other experimental fields or options were on the SYN, it will also be necessary to follow their specifications for fall-back too. It would make sense to coordinate all the strategies for fall-back in order to isolate the specific cause of the problem.

If the TCP initiator is caching failed connection attempts, it SHOULD NOT give up using ECT on the first SYN of subsequent connection attempts until it is clear that a blockage persistently and specifically affects ECT on SYNs. This is because loss is so commonplace for other reasons. Even if it does eventually decide to give up setting ECT on the SYN, it will probably not need to give up on AccECN on the SYN. In any case, if a cache is used, it SHOULD be arranged to expire so that the initiator will infrequently attempt to check whether the problem has been resolved.

Other fall-back strategies MAY be adopted where applicable (see Section 4.2.2 for suggestions, and the conditions under which they would apply).

3.2.2. SYN-ACK (Send)

3.2.2.1. Setting ECT on the SYN-ACK

For the ECN++ experiment, the TCP implementation will set ECT on SYN-ACKs. Section 4.3 of [RFC8311] allows an ECN experiment to use ECT on a SYN-ACK, by updating Section 6.1.1 of RFC 3168 (which prohibited it).

3.2.2.2. SYN-ACK Congestion Response

A host that sets ECT on SYN-ACKs MUST reduce its initial window in response to any congestion feedback, whether using classic ECN or AccECN (see Section 4.3.1). It SHOULD reduce it to 1 SMSS. This is different to the behaviour specified in an earlier experiment that set ECT on the SYN-ACK [RFC5562]. This is justified in Section 4.3.2.

The responder does not have to back off its retransmission timer because the ECN feedback proves that the network is delivering packets successfully and is not severely overloaded. Also the responder does not have to leave slow start or reduce ssthresh, which is not even required when a SYN-ACK has been lost.

The congestion response to CE-marking on a SYN-ACK for a server that implements either the TCP Fast Open experiment (TFO [RFC7413]) or experimentation with an initial window of more than 3 segments (e.g. IW10 [RFC6928]) is discussed in Section 5.

3.2.2.3. Fall-Back Following No Response to an ECT SYN-ACK

After the responder sends a SYN-ACK with ECT set, if its retransmission timer expires it SHOULD retransmit one more SYN-ACK with ECT set (and back-off its timer as usual). If the timer expires again, it SHOULD retransmit the SYN-ACK with not-ECT in the IP header. If other experimental fields or options were on the initial SYN-ACK, it will also be necessary to follow their specifications for fall-back. It would make sense to co-ordinate all the strategies for fall-back in order to isolate the specific cause of the problem.

This fall-back strategy attempts to use ECT one more time than the strategy for ECT SYN-ACKs in [RFC5562] (which is made obsolete, being superseded by the present specification). Other fall-back strategies MAY be adopted if found to be more effective, e.g. fall-back to not-ECT on the first retransmission attempt.

The server MAY cache failed connection attempts, e.g. per client access network. If the TCP server is caching failed connection attempts, it SHOULD NOT give up using ECT on the first SYN-ACK of subsequent connection attempts until it is clear that the blockage persistently and specifically affects ECT on SYN-ACKs. This is because loss is so commonplace for other reasons (see Section 3.2.1.4).

A client-based alternative to caching at the server is given in Section 4.3.3.

If either endpoint caches failed attempts, the cache SHOULD be arranged to expire so that the endpoint will infrequently attempt to check whether the problem has been resolved.

3.2.3. Pure ACK (Send)

A Pure ACK is an ACK packet that does not carry data, which includes the Pure ACK at the end of TCP's 3-way handshake.

For the ECN++ experiment, whether a TCP implementation sets ECT on a Pure ACK depends on whether or not Accurate ECN TCP feedback [I-D.ietf-tcpm-accurate-ecn] has been successfully negotiated for a particular TCP connection, as specified in the following two subsections.

3.2.3.1. Pure ACK without AccECN Feedback

If AccECN has not been successfully negotiated for a connection, ECT MUST NOT be set on Pure ACKs by either end.

3.2.3.2. Pure ACK with AccECN Feedback

For the ECN++ experiment, a host can only set ECT on outgoing Pure ACKs if it satisfies the following three conditions:

* it MUST have successfully negotiated AccECN feedback for the connection;

* it MUST have successfully negotiated SACK [RFC2018] for the connection;

* when it checks whether incoming pure ACKs are duplicates it MUST apply the additional test in Section 3.3.3.1.

If the host satisfies all these requirements, it can then set ECT on a pure ACK. Section 4.3 of [RFC8311] allows an ECN experiment to use ECT on a pure ACK, by updating Sections 5.2 and 6.1.4 of RFC 3168 (which prohibited it).

> MEASUREMENTS NEEDED: Measurements are needed to learn how the deployed base of network elements and RFC 3168 servers react to pure ACKs marked with the ECT(0)/ECT(1)/CE codepoints, i.e. whether they are dropped, codepoint cleared or processed and the congestion indication fed back on a subsequent packet.

See Section 3.3.3 for the implications if a host receives a CE-marked Pure ACK.

3.2.3.2.1. Pure ACK Congestion Response

This subsection only applies for a host that is setting ECT on outgoing pure ACKs, which is conditional on it satisfying the three conditions in Section 3.2.3.2.

A host that sets ECT on pure ACKs SHOULD respond to the congestion signal resulting from pure ACKs being marked with the CE codepoint. The specific response will need to be defined as an update to each congestion control specification. Possible responses to congestion feedback include reducing the congestion window (CWND) and/or regulating the pure ACK rate (see Section 4.4.2.1).

Note that, in comparison, TCP Congestion Control [RFC5681] does not require a TCP to detect or respond to loss of pure ACKs at all; it requires no reduction in congestion window or ACK rate.

3.2.4. Window Probe (Send)

For the ECN++ experiment, the TCP sender will set ECT on window probes. Section 4.3 of [RFC8311] allows an ECN experiment to use ECT on a window probe, by updating Section 6.1.6 of RFC 3168 (which prohibited it).

A window probe contains a single octet, so it is no different from a regular TCP data segment. Therefore a TCP receiver will feed back any CE marking on a window probe as normal (either using classic ECN feedback or AccECN feedback). The sender of the probe will then reduce its congestion window as normal.

A receive window of zero indicates that the receiving application is not consuming data fast enough and does not imply anything about network congestion. Once the receive window opens, the congestion window might become the limiting factor, so it is correct that CE-marked probes reduce the congestion window. This complements cwnd validation [RFC7661], which reduces cwnd as more time elapses without having used available capacity. However, CE-marking on window probes does not reduce the rate of the probes themselves. This is unlikely to present a problem, given the duration between window probes doubles [RFC1122] as long as the receiver is advertising a zero window (currently minimum 1 second, maximum at least 1 minute [RFC6298]).

> MEASUREMENTS NEEDED: Measurements are needed to learn how the deployed base of network elements and servers react to Window probes marked with the ECT(0)/ECT(1)/CE codepoints, i.e. whether they are dropped, codepoint cleared or processed.

3.2.5. FIN (Send)

A TCP implementation can set ECT on a FIN.

See Section 3.3.4 for the implications if a host receives a CE-marked FIN.

A congestion response to a CE-marking on a FIN is not required.

After sending a FIN, the endpoint will not send any more data in the connection. Therefore, even if the FIN-ACK indicates that the FIN was CE-marked (whether using classic or AccECN feedback), reducing the congestion window will not affect anything.

After sending a FIN, a host might send one or more pure ACKs. If it is using one of the techniques in Section 3.2.3 to regulate the delayed ACK ratio for pure ACKs, it could equally be applied after a FIN. But this is not required.

> MEASUREMENTS NEEDED: Measurements are needed to learn how the deployed base of network elements and servers react to FIN packets marked with the ECT(0)/ECT(1)/CE codepoints, i.e. whether they are dropped, codepoint cleared or processed.

3.2.6. RST (Send)

A TCP implementation can set ECT on a RST.

See Section 3.3.5 for the implications if a host receives a CE-marked RST.

A congestion response to a CE-marking on a RST is not required (and actually not possible).

> MEASUREMENTS NEEDED: Measurements are needed to learn how the deployed base of network elements and servers react to RST packets marked with the ECT(0)/ECT(1)/CE codepoints, i.e. whether they are dropped, codepoint cleared or processed.

Implementers SHOULD ensure that RST packets are always sent out with the same ECN field regardless of the TCP state machine. Otherwise the ECN field could reveal internal TCP state. For instance, the ECN field on a RST ought not to reveal any distinction between a non-listening port, a recently in-use port, and a closed session port.

3.2.7. Retransmissions (Send)

For the ECN++ experiment, the TCP sender will set ECT on retransmitted segments. Section 4.3 of [RFC8311] allows an ECN experiment to use ECT on a retransmission, by updating Sections 6.1.5 of RFC 3168 (which prohibited it).

See Section 3.3.6 for the implications if a host receives a CE-marked retransmission.

If the TCP sender receives feedback that a retransmitted packet was CE-marked, it will react as it would to any feedback of CE-marking on a data packet.

> MEASUREMENTS NEEDED: Measurements are needed to learn how the deployed base of network elements and servers react to retransmissions marked with the ECT(0)/ECT(1)/CE codepoints, i.e. whether they are dropped, codepoint cleared or processed.

3.2.8. General Fall-back for any Control Packet or Retransmission

Extensive measurements in fixed and mobile networks [Mandalari18] have found no evidence of blockages due to ECT being set on any type of TCP control packet.

In case traversal problems arise in future, fall-back measures have been specified above, but only for the cases regarding the initial packet of a half-connection (SYN or SYN-ACK) where ECT is persistently failing to get through.

Fall-back measures for blockage of ECT on other TCP control packets MAY be implemented. However they are not specified here given the lack of any evidence they will be needed. Section 4.9 justifies this advice in more detail.

3.3. Receiver Behaviour

The present ECN++ specification primarily concerns the behaviour for sending TCP control packets or retransmissions. Below are a few changes to the receive side of an implementation that are recommended while updating its send side. Nonetheless, where deployment is concerned, ECN++ is still a sender-only deployment, because it does not depend on receivers complying with any of these recommendations.

3.3.1. Receiver Behaviour for Any TCP Control Packet or Retransmission

RFC8311 is a standards track update to RFC 3168 in order to (amongst other things) "...allow the use of ECT codepoints on SYN packets, pure acknowledgement packets, window probe packets, and retransmissions of packets..., provided that the changes from RFC 3168 are documented in an Experimental RFC in the IETF document stream."

Section 4.3 of RFC 8311 amends every statement in RFC 3168 that precludes the use of ECT on control packets and retransmissions to add "unless otherwise specified by an Experimental RFC in the IETF document stream". The present specification is such an Experimental RFC. Therefore, In order for the present RFC 8311 experiment to be useful, TCP receivers will need to satisfy the following requirements:

* Any TCP implementation SHOULD accept receipt of any valid TCP control packet or retransmission irrespective of its IP/ECN field. If any existing implementation does not, it SHOULD be updated to do so.

* A TCP implementation taking part in the experiments proposed here MUST accept receipt of any valid TCP control packet or retransmission irrespective of its IP/ECN field.

The following sections give further requirements specific to each type of control packet.

These measures are derived from the robustness principle of "... be liberal in what you accept from others", not only to ensure compatibility with the present experimental specification, but also any future protocol changes that allow ECT on any TCP packet.

3.3.2. SYN (Receive)

RFC 3168 negotiates the use of ECN for the connection end-to-end using the ECN flags in the TCP header. RFC 3168 originally said that "A host MUST NOT set ECT on SYN ... packets." but it was silent as to what a TCP server ought to do if it receives a SYN packet with a non-zero IP/ECN field anyway.

For the avoidance of doubt, the normative statements for all TCP control packets in Section 3.3.1 are interpreted for the specific case when a SYN is received as follows:

* Any TCP server implementation SHOULD accept receipt of a valid SYN that requests ECN support for the connection, irrespective of the IP/ECN field of the SYN. If any existing implementation does not, it SHOULD be updated to do so.

* A TCP implementation taking part in the ECN++ experiment MUST accept receipt of a valid SYN, irrespective of its IP/ECN field.

* If the SYN is CE-marked and the server has no logic to feed back a CE mark on a SYN-ACK (e.g. it does not support AccECN), it has to ignore the CE-mark (the client detects this case and behaves conservatively in mitigation – see Section 3.2.1.3).

Rationale: At the time of the writing, some implementations of TCP servers (see Section 4.2.2.2) assume that, if a host receives a SYN with a non-zero IP/ECN field, it must be due to network mangling, and they disable ECN for the rest of the connection. Section 4.2.2.2 cites a measurement study run in 2017 that found no occurrence of this type of network mangling. However, a year earlier, when ECN was enabled on connections from Apple clients, there was a case of a whole network that re-marked the ECN field of every packet to CE (it was rapidly fixed).

When ECN was not allowed on SYNs, it made sense to look for a non-zero ECN field on the SYN to detect this type of network mangling. But now that ECN is being allowed on a SYN, detection needs to be more nuanced. A server needs to disable the test on the SYN alone for AccECN SYNs (which was done for Linux RFC 3168 servers in 2019 [relax-strict-ecn]) and for RFC 3168 SYNs it needs to watch for three or four packets all set to CE at the start of a flow. If such mangling is indeed now so rare, it would also be preferable to log each case detected and manually report it to the responsible network, so that the problem will eventually be eliminated.

3.3.3. Pure ACK (Receive)

For the avoidance of doubt, the normative statements for all TCP control packets in Section 3.3.1 are interpreted for the specific case when a Pure ACK is received as follows:

* Any TCP implementation SHOULD accept receipt of a pure ACK with a non-zero IP-ECN field, as now allowed by [RFC8311], which updated section 6.1.4 of [RFC3168] (which previously required the ECN field to be cleared to zero on all pure ACKs).

* A TCP implementation taking part in the ECN++ experiment MUST accept receipt of a pure ACK with a non-zero ECN field.

The question of whether and how the receiver of pure ACKs is required to feed back any CE marks on them is outside the scope of the present specification because it is a matter for the relevant feedback specification ([RFC3168] or [I-D.ietf-tcpm-accurate-ecn]). AccECN feedback mandates counting of CE marks on any control packets including pure ACKs. Whereas RFC 3168 is silent on this point, so feedback of CE-markings might be implementation specific (see Section 4.4.2.1).

3.3.3.1. Additional DupACK Check

A TCP implementation that is sending ECN-capable pure ACKs MUST add the following check to all its algorithms that detect duplicate ACKs (defined in Section 2 of [RFC5681]):

* If there is no SACK option on an incoming pure ACK (ECN-capable or not) despite SACK having been negotiated, it is not counted as a duplicate ACK.

SACK blocks are known to be stripped on some paths through the public Internet, or some implementations are known to negotiate SACK but never send SACK blocks. Therefore, if a TCP implementation is sending ECN-capable pure ACKs, it SHOULD check for these pathologies (unless it is intended solely for an environment known to be clear of them), as follows:

* If an incoming pure ACK appears to be a duplicate, but:

  - it does not carry any SACK blocks (despite SACK having been negotiated)

  - and AccECN [I-D.ietf-tcpm-accurate-ecn] has been negotiated but the ACE field has increased by less than 3 (after allowing for wrap)

  then the implementation can deem that SACK blocks are being stripped. In this case, for the rest of the connection, it SHOULD:

  - either disable the sending of ECN-capable pure ACKs;

  - or replace any instance of the above additional DupACK check with heuristics such as the following examples:

    o If the ACE field [I-D.ietf-tcpm-accurate-ecn] has incremented by at least 3, an incoming ACK is not counted as a duplicate ACK;

- if the Timestamp option (TSopt) is available [RFC7323], and the echoed timestamp (TSecr) increments relative to the previous ACK, an incoming ACK is not counted as a Duplicate ACK.

An additional DupACK check is one of the mandatory conditions specified in Section 3.2.3.2 for sending ECN-capable pure ACKs (the others are AccECN mode and SACK-negotiated mode).

See Section 4.4.4 for rationale.

3.3.4. FIN (Receive)

The normative statements for all TCP control packets in Section 3.3.1 apply for the specific case when a FIN is received, with 'valid' defined as follows:

The TCP data receiver MUST ignore the CE codepoint on incoming FINs that fail any validity check. The validity check in section 5.2 of [RFC5961] is RECOMMENDED.

3.3.5. RST (Receive)

The normative statements for all TCP control packets in Section 3.3.1 apply for the specific case when a RST is received, with 'valid' defined as follows:

The "challenge ACK" approach to checking the validity of RSTs (section 3.2 of [RFC5961]) is RECOMMENDED at the data receiver.

3.3.6. Retransmissions (Receive)

The normative statements for all TCP control packets in Section 3.3.1 apply for the specific case when a retransmission is received, with 'valid' defined as follows:

The TCP data receiver MUST ignore the CE codepoint on incoming segments that fail any validity check. The validity check in section 5.2 of [RFC5961] is RECOMMENDED. This will effectively mitigate an attack that uses spoofed data packets to fool the receiver into feeding back spoofed congestion indications to the sender, which in turn would be fooled into continually reducing its congestion window.

4. Rationale

This section is informative, not normative. It presents counter-arguments against the justifications in the RFC series for disabling ECN on TCP control segments and retransmissions. It also gives rationale for why ECT is safe on control segments that have not, so far, been mentioned in the RFC series (FINs and RSTs). First it addresses over-arching arguments used for most packet types, then it addresses the specific arguments for each packet type in turn.

4.1. The Reliability Argument

Section 5.2 of RFC 3168 states:

> "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 [at a subsequent node] in the network would be detected by the end nodes and interpreted as an indication of congestion."

We believe this argument is misplaced. TCP does not deliver most control packets reliably. So it is more important to allow control packets to be ECN-capable, which greatly improves reliable delivery of the control packets themselves (see motivation in Section 1.1). ECN also improves the reliability and latency of delivery of any congestion notification on control packets, particularly where TCP does not detect the loss of certain types of control packet anyway. Both these points outweigh by far the concern that a CE marking applied to a control packet by one node might subsequently be dropped by another node.

The principle to determine whether a packet can be ECN-capable ought to be "do no extra harm", meaning that the reliability of a congestion signal's delivery ought to be no worse with ECN than without.

It will help to first compare with the case of a reliably delivered packet (e.g. a SYN or data packet) that is made ECN-capable. If it is CE-marked at two buffers in succession, it is not discarded by the first buffer so it goes on to help congest the second. But it delivers only one congestion signal. Similarly, if instead it is marked at the first buffer and dropped at the second, it still helps congest the second buffer, but it still delivers only one congestion signal (the loss).

Some non-ECN TCP control packets (e.g. pure ACKs or FINs) certainly do not reliably deliver a congestion signal if they are discarded. But, making such control packets ECN-capable upgrades their ability

to deliver a congestion signal from a buffer with ECN support. However, as before, they still cannot reliably deliver a loss signal from a non-ECN buffer. This includes the case where one congested buffer CE-marks such a packet, then a second congested buffer without ECN support discards it.

Thus ECN is always more and never less reliable for delivery of congestion notification.

4.2. SYNs

RFC 5562 presents two arguments against ECT marking of SYN packets (quoted verbatim):

> "First, when the TCP SYN packet is sent, there are no guarantees that the other TCP endpoint (node B in Figure 2) is ECN-Capable, or that it would be able to understand and react if the ECN CE codepoint was set by a congested router.
>
> Second, the ECN-Capable codepoint in TCP SYN packets could be misused by malicious clients to "improve" the well-known TCP SYN attack. By setting an ECN-Capable codepoint in TCP SYN packets, a malicious host might be able to inject a large number of TCP SYN packets through a potentially congested ECN-enabled router, congesting it even further."

The first point actually describes two subtly different issues. So below three arguments are countered in turn.

4.2.1. Argument 1a: Unrecognized CE on the SYN

This argument certainly applied at the time RFC 5562 was written, when no ECN responder mechanism had any logic to recognize a CE marking on a SYN and, even if logic were added, there was no field in the SYN-ACK to feed it back. The problem was that, during the 3WHS, the flag in the TCP header for ECN feedback (called Echo Congestion Experienced) had been overloaded to negotiate the use of ECN itself.

The accurate ECN (AccECN) protocol [I-D.ietf-tcpm-accurate-ecn] has since been designed to solve this problem. Two features are important here:

1. An AccECN server uses the 3 AccECN flags in the TCP header of the SYN-ACK to respond to the client. 4 of the possible 8 codepoints provide enough space for the server to feed back which of the 4 IP/ECN codepoints was on the incoming SYN (including CE of course).

2. If any of these 4 codepoints are in the SYN-ACK, it confirms that the server supports AccECN and, if another codepoint is returned, it confirms that the server does not support AccECN.

This still does not seem to allow a client to set ECT on a SYN, it only finds out whether the server would have supported it afterwards. The trick the client uses for ECN++ is to set ECT on the SYN optimistically then, if the SYN-ACK reveals that the server wouldn't have understood CE on the SYN, the client responds conservatively as if the SYN was marked with CE.

The recommended conservative congestion response is to reduce the initial window, which does not affect the performance of very popular protocols such as HTTP, since it is currently extremely rare for an HTTP client to send more than one packet as its initial request anyway (for data on HTTP/1 & HTTP/2 request sizes see Fig 3 in [Manzoor17]). Any clients that do frequently use a larger initial window for their first message to the server can cache which servers will not understand ECT on a SYN (see Section 4.2.3 below). If caching is not practical, such clients could reduce the initial window to say IW2 or IW3.

> EXPERIMENTATION NEEDED: Experiments will be needed to determine any better strategy for reducing IW in response to congestion on a SYN, when the server does not support congestion feedback on the SYN-ACK (whether cached or discovered explicitly).

4.2.2. Argument 1b: ECT Considered Invalid on the SYN

Given that prior to [RFC8311] ECT-marked SYN packets were prohibited, it cannot be assumed they will be accepted, by TCP middleboxes or servers.

4.2.2.1. ECT on SYN Considered Invalid by Middleboxes

According to a study using 2014 data [ecn-pam] from a limited range of fixed vantage points, for the top 1M Alexa web sites, adding the ECN capability to SYNs increased connection establishment failures by about 0.4%.

From a wider range of fixed and mobile vantage points, a more recent study in Jan-May 2017 [Mandalari18] found no occurrences of blocking of ECT on SYNs. However, in more than half the mobile networks tested it found wiping of the ECN codepoint at the first hop.

> MEASUREMENTS NEEDED: As wiping at the first hop is remedied, measurements will be needed to check whether SYNs with ECT are sometimes blocked deeper into the path.

Silent failures introduce a retransmission timeout delay (default 1 second) at the initiator before it attempts any fall back strategy (whereas explicit RSTs can be dealt with immediately). Ironically, making SYNs ECN-capable is intended to avoid the timeout when a SYN is lost due to congestion. Fortunately, if there is any discard of ECN-capable SYNs due to policy, it will occur predictably, not randomly like congestion. So the initiator should be able to avoid it by caching paths or servers that do not support ECN-capable SYNs (see the last paragraph of Section 3.2.1.2).

4.2.2.2. ECT on SYN Considered Invalid by Servers

A study conducted in Nov 2017 [Kuehlewind18] found that, of the 82% of the Alexa top 50k web servers that supported ECN, 84% disabled ECN if the IP/ECN field on the SYN was ECT0, CE or either. Given most web servers use Linux, this behaviour can most likely be traced to a patch contributed in May 2012 that was first distributed in v3.5 of the Linux kernel [strict-ecn]. The comment says "RFC3168 : 6.1.1 SYN packets must not have ECT/ECN bits set. If we receive a SYN packet with these bits set, it means a network is playing bad games with TOS bits. In order to avoid possible false congestion notifications, we disable TCP ECN negociation." Of course, some of the 84% might be due to similar code in other OSs.

For brevity we shall call this the "over-strict" ECN test, because it is over-conservative with what it accepts, contrary to Postel's robustness principle. A robust protocol will not usually assume network mangling without comparing with the value originally sent, and one packet is not sufficient to make an assumption with such irreversible consequences anyway.

Ironically, networks rarely seem to alter the IP/ECN field on a SYN from zero to non-zero anyway. In a study conducted in Jan-May 2017 over millions of paths from vantage points in a few dozen mobile and fixed networks [Mandalari18], no such transition was observed. With such a small or non-existent incidence of this sort of network mangling, it would be preferable to report any residual problem paths so that they can be fixed.

Whatever, the widespread presence of this 'over-strict' test proves that RFC 5562 was correct to expect that ECT would be considered invalid on SYNs. Nonetheless, it is not an insurmountable problem - the over-strict test in Linux was patched in Apr 2019 [relax-strict-ecn] and caching can work round it where previous versions of Linux are running. The prevalence of these "over-strict" ECN servers makes it challenging to cache them all. However, Section 4.2.3 below explains how a cache of limited size can alleviate this problem for a client's most popular sites.

For the future, [RFC8311] updates RFC 3168 to clarify that the IP/ECN field does not have to be zero on a SYN if documented in an experimental RFC such as the present ECN++ specification.

4.2.3. Caching Strategies for ECT on SYNs

Given the server handling of ECN on SYNs outlined in Section 4.2.2.2 above, an initiator might combine AccECN with three candidate strategies for setting ECT on a SYN and caching the outcome:

(S1): Pessimistic ECT and cache successes: The initiator always requests AccECN, but by default without ECT on the SYN. Then it caches those servers that confirm that they support AccECN as 'ECT SYN OK'. On a subsequent connection to any server that supports AccECN, the initiator can then set ECT on the SYN. When connecting to other servers (non-ECN or classic ECN) it will not set ECT on the SYN, so it will not fail the 'over-strict' ECN test.

Longer term, as servers upgrade to AccECN, the initiator is still requesting AccECN, so it will add them to the cache and use ECT on subsequent SYNs to those servers. However, assuming it has to cap the size of the cache, the client will not have the benefit of ECT SYNs to those less frequently used AccECN servers expelled from its cache.

(S2): Optimistic ECT: The initiator always requests AccECN and by default sets ECT on the SYN. Then, if the server response shows it has no AccECN logic (so it cannot feed back a CE mark), the initiator conservatively behaves as if the SYN was CE-marked, by reducing its initial window. Two caching sub-strategies are feasible:

a. No cache.

b. Cache failures: The optimistic ECT strategy can be improved by caching solely those servers that do not support AccECN as 'ECT SYN NOK'. This would include non-ECN servers and all Classic ECN servers whether 'over-strict' or not. On subsequent connections to these non-AccECN servers, the initiator will still request AccECN but not set ECT on the SYN. Then, the connection can still fall back to Classic ECN, if the server supports it, and the initiator can use its full initial window (if it has enough request data to need it).

Longer term, as servers upgrade to AccECN, the initiator will remove them from the cache and use ECT on subsequent SYNs to that server.

Where an access network operator mediates Internet access via a proxy that does not support AccECN, the optimistic ECT strategy will always fail. This scenario is more likely in mobile networks. Therefore, a mobile host could cache lack of AccECN support per attached access network operator. Whenever it attached to a new operator, it could check a well-known AccECN test server and, if it found no AccECN support, it would add a cache entry for the attached operator. It would only use ECT when neither network nor server were cached. It would only populate its per server cache when not attached to a non-AccECN proxy.

(S3): ECT by configuration: In a controlled environment, the administrator can make sure that servers support ECN-capable SYN packets. Examples of controlled environments are single-tenant DCs, and possibly multi-tenant DCs if it is assumed that each tenant mostly communicates with its own VMs.

For unmanaged environments like the public Internet, pragmatically the choice is between strategies (S1), (S2A) and (S2B). The normative specification for ECT on a SYN in Section 3.2.1 recommends the "optimistic ECT and cache failures" strategy (S2B) but the choice depends on the implementer’s motivation for using ECN++, and the deployment prevalence of different technologies and bug-fixes.

* The "pessimistic ECT and cache successes" strategy (S1) suffers from exposing the initial SYN to the prevailing loss level, even if the server supports ECT on SYNs, but only on the first connection to each AccECN server. If AccECN becomes widely deployed on servers, SYNs to those AccECN servers that are less frequently used by the client and therefore don’t fit in the cache will not benefit from ECN protection at all.

* The "optimistic ECT without a cache" strategy (S2A) is the simplest. It would satisfy the goal of an implementer who is solely interested in low latency using AccECN and ECN++ and is not concerned about fall-back to Classic ECN.

* The "optimistic ECT and cache failures" strategy (S2B) exploits ECT on SYNs from the very first attempt. But if the server turns out to be ’over-strict’ it will disable ECN for the connection, but only for the first connection if it’s one of the client’s more popular servers that fits in the cache. If the server turns out

not to support AccECN, the initiator has to conservatively limit its initial window, but again only for the first connection if it's one of the client's more popular servers (and anyway this rarely makes any difference when most client requests fit in a single packet).

Note that, if AccECN deployment grows, storage for 'caching successes' (S1) starts off small then grows, while with 'caching failures' (S2B) it is large at first, then shrinks. At half-way, the size of the cache has to be capped with either approach, so the default behaviour for all the servers that do not fit in the cache is as important as the behaviour for the popular servers that do fit.

MEASUREMENTS NEEDED: Measurements are needed to determine which strategy would be sufficient for any particular client, whether a particular client would need different strategies in different circumstances and how many occurrences of problems would be masked by how few cache entries.

Another strategy would be to send a not-ECT SYN a short delay (below the typical lowest RTT) after an ECT SYN and only accept the non-ECT connection if it returned first. This would reduce the performance penalty for those deploying ECT SYN support. However, this 'happy eyeballs' approach becomes complex when multiple optional features are all tried on the first SYN (or on multiple SYNs), so it is not recommended.

4.2.4. Argument 2: DoS Attacks

[RFC5562] says that ECT SYN packets could be misused by malicious clients to augment "the well-known TCP SYN attack". It goes on to say "a malicious host might be able to inject a large number of TCP SYN packets through a potentially congested ECN-enabled router, congesting it even further."

We assume this is a reference to the TCP SYN flood attack (see https://en.wikipedia.org/wiki/SYN_flood), which is an attack against a responder end point. We assume the idea of this attack is to use ECT to get more packets through an ECN-enabled router in preference to other non-ECN traffic so that they can go on to use the SYN flooding attack to inflict more damage on the responder end point. This argument could apply to flooding with any type of packet, but we assume SYNs are singled out because their source address is easier to spoof, whereas floods of other types of packets are easier to block.

Mandating Not-ECT in an RFC does not stop attackers using ECT for flooding. Nonetheless, if a standard says SYNs are not meant to be ECT it would make it legitimate for firewalls to discard them.

However this would negate the considerable benefit of ECT SYNs for compliant transports and seems unnecessary because RFC 3168 already provides the means to address this concern. In section 7, RFC 3168 says "During periods where ... the potential packet marking rate would be high, our recommendation is that routers drop packets rather then set the CE codepoint..." and this advice is repeated in [RFC7567] (section 4.2.1). This makes it harder for flooding packets to gain from ECT.

[ecn-overload] showed that ECT can only slightly augment flooding attacks relative to a non-ECT attack. It was hard to overload the link without causing the queue to grow, which in turn caused the AQM to disable ECN and switch to drop, thus negating any advantage of using ECT. This was true even with the switch-over point set to 25% drop probability (i.e. the arrival rate was 133% of the link rate).

4.3. SYN-ACKs

The proposed approach in Section 3.2.2 for experimenting with ECN-capable SYN-ACKs is effectively identical to the scheme called ECN+ [ECN-PLUS]. In 2005, the ECN+ paper demonstrated that it could reduce the average Web response time by an order of magnitude. It also argued that adding ECT to SYN-ACKs did not raise any new security vulnerabilities.

4.3.1. Possibility of Unrecognized CE on the SYN-ACK

The feedback behaviour by the initiator in response to a CE-marked SYN-ACK from the responder depends on whether classic ECN feedback [RFC3168] or AccECN feedback [I-D.ietf-tcpm-accurate-ecn] has been negotiated. In either case no change is required to RFC 3168 or the AccECN specification.

Some classic ECN client implementations might ignore a CE-mark on a SYN-ACK, or even ignore a SYN-ACK packet entirely if it is set to ECT or CE. This is a possibility because an RFC 3168 implementation would not necessarily expect a SYN-ACK to be ECN-capable. This issue already came up when the IETF first decided to experiment with ECN on SYN-ACKs [RFC5562] and it was decided to go ahead without any extra precautionary measures. This was because the probability of encountering the problem was believed to be low and the harm if the problem arose was also low (see Appendix B of RFC 5562).

4.3.2. Response to Congestion on a SYN-ACK

The IETF has already specified an experiment with ECN-capable SYN-ACK packets [RFC5562]. It was inspired by the ECN+ paper, but it specified a much more conservative congestion response to a CE-marked SYN-ACK, called ECN+/TryOnce. This required the server to reduce its initial window to 1 segment (like ECN+), but then the server had to send a second SYN-ACK and wait for its ACK before it could continue with its initial window of 1 SMSS. The second SYN-ACK of this 5-way handshake had to carry no data, and had to disable ECN, but no justification was given for these last two aspects.

The present ECN++ experimental specification obsoletes RFC 5562 because it uses the ECN+ congestion response, not ECN+/TryOnce. First we argue against the rationale for ECN+/TryOnce given in sections 4.4 and 6.2 of [RFC5562]. It starts with a rather too literal interpretation of the requirement in RFC 3168 that says TCP's response to a single CE mark has to be "essentially the same as the congestion control response to a *single* dropped packet." TCP's response to a dropped initial (SYN or SYN-ACK) packet is to wait for the retransmission timer to expire (currently 1s). However, this long delay assumes the worst case between two possible causes of the loss: a) heavy overload; or b) the normal capacity-seeking behaviour of other TCP flows. When the network is still delivering CE-marked packets, it implies that there is an AQM at the bottleneck and that it is not overloaded. This is because an AQM under overload will disable ECN (as recommended in section 7 of RFC 3168 and repeated in section 4.2.1 of RFC 7567). So scenario (a) can be ruled out. Therefore, TCP's response to a CE-marked SYN-ACK can be similar to its response to the loss of _any_ packet, rather than backing off as if the special _initial_ packet of a flow has been lost.

How TCP responds to the loss of any single packet depends what it has just been doing. But there is not really a precedent for TCP's response when it experiences a CE mark having sent only one (small) packet. If TCP had been adding one segment per RTT, it would have halved its congestion window, but it hasn't established a congestion window yet. If it had been exponentially increasing it would have exited slow start, but it hasn't started exponentially increasing yet so it hasn't established a slow-start threshold.

Therefore, we have to work out a reasoned argument for what to do. If an AQM is CE-marking packets, it implies there is already a queue and it is probably already somewhere around the AQM's operating point - it is unlikely to be well below and it might be well above. So, the more data packets that the client sends in its IW, the more likely at least one will be CE marked, leading it to exit slow-start early. On the other hand, it is highly unlikely that the SYN-ACK

itself pushed the AQM into congestion, so it will be safe to introduce another single segment immediately (1 RTT after the SYN-ACK). Therefore, starting to probe for capacity with a slow start from an initial window of 1 segment seems appropriate to the circumstances. This is the approach adopted in Section 3.2.2.

> EXPERIMENTATION NEEDED: Experiments will be needed to check the above reasoning and determine any better strategy for reducing IW in response to congestion on a SYN-ACK (or a SYN).

4.3.3. Fall-Back if ECT SYN-ACK Fails

Section 3.2.2.3 describes how a server could cache failed connection attempts. As an alternative, the server could rely on the client to cache failed attempts (on the basis that the client would cache a failure whether ECT was blocked on the SYN or the SYN-ACK). This strategy cannot be used if the SYN does not request AccECN support.

It works as follows. If a server would rather not maintain its own cache, when it receives a SYN that requests AccECN support but is set to not-ECT, the server replies with a SYN-ACK also set to not-ECT. This gives the client the power to disable ECT on both the SYN and SYN-ACK, if it has cached knowedge that there were previous problems.

If a middlebox only blocks ECT on SYNs, not SYN-ACKs, this strategy might disable ECN on a SYN-ACK when it did not need to, but at least it saves the server from maintaining a cache. However, a client cannot rely on all non-caching servers suppressing ECT on SYN/ACKs when they might not need to.

Therefore, a more practical way for a client to cache failures on behalf of the server would be for the client to initially fall-back to Not-ECT on the SYN after multiple timeouts, then if that doesn't resolve the problem, the client could disable ECN completely at the TCP layer, and if the connection then works, it could cache to disable ECN in future. With this approach, it is still preferable but optional for the server to also cache failure to deliver ECN-capable SYN-ACKs.

4.4. Pure ACKs

4.4.1. Arguments against ECT on Pure ACKS

After the general reliability argument already quoted in Section 4.1, Section 5.2 of RFC 3168 goes on to use ECT marking of pure ACKs as a specific example of the reliability argument:

> "Transport protocols such as TCP do not necessarily detect all packet drops, such as the drop of a "pure" ACK packet; for example, TCP does not reduce the arrival rate of subsequent ACK packets in response to an earlier dropped ACK packet. Any proposal for extending ECN-Capability to such packets would have to address issues such as the case of an ACK packet that was marked with the CE codepoint but was later dropped in the network. We believe that this aspect is still the subject of research, so this document specifies that at this time, "pure" ACK packets MUST NOT indicate ECN-Capability."

Later on, in section 6.1.4 it reads:

> "For the current generation of TCP congestion control algorithms, pure acknowledgement packets (e.g., packets that do not contain any accompanying data) MUST be sent with the not-ECT codepoint. Current TCP receivers have no mechanisms for reducing traffic on the ACK-path in response to congestion notification. Mechanisms for responding to congestion on the ACK-path are areas for current and future research. (One simple possibility would be for the sender to reduce its congestion window when it receives a pure ACK packet with the CE codepoint set). For current TCP implementations, a single dropped ACK generally has only a very small effect on the TCP's sending rate."

4.4.2. Counter-arguments for ECT on Pure ACKs

The first argument above is a specific instance of the reliability argument for the case of pure ACKs. This has already been addressed by countering the general reliability argument in Section 4.1.

The second argument says that ECN ought not to be enabled on Pure ACKs unless there is a mechanism to respond to it. Although the above passage from RFC 3168 envisages the possibility of ECN on pure ACKs in the future, it is silent on how its ECN feedback mechanisms would be used if CE markings did arrive on pure ACKs. In contrast, the position of AccECN with respect to the three parts of a congestion response mechanism is as follows:

Detection: The AccECN spec requires the receiver of any TCP packets including pure ACKs to count any CE marks on them (whether or not it sends ECN-capable control packets itself).

Feedback: The AccECN Data Receiver continually feeds back a count of the number of CE-marked packets (including pure ACKs) that it has received.

Even if the receiver of a CE-mark on a pure ACK does not feed it back immediately, it still includes it within subsequent feedback, for instance when it later sends a data segment. Even if an AccECN host has no data outstanding, it is still required to send an 'increment-triggered' pure ACK after every 'n' CE marks it receives, where 'n' is at least 3.

Congestion response: Once an AccECN Data Sender receives feedback about CE-markings on pure ACKs, it will be able to reduce the congestion window (cwnd) and/or the ACK rate. The specific response to congestion feedback is out of scope of both AccECN and the present spec, because it would be defined in a base TCP congestion control spec [RFC5681] [RFC9438] or a variant. Nonetheless, in order to decide whether the present ECN++ experimental spec should allow a host to set ECT on pure ACKs, we only need to know whether a congestion response would be feasible - we do not have to standardize it. Possible responses are discussed in the following subsections,

4.4.2.1. Congestion Window Response to CE-Marked Pure ACKs

This subsection explores issues that congestion control designers will need to consider when defining a cwnd response to CE-marked Pure ACKs.

A CE-mark on a Pure ACK does not mean that only Pure ACKs are causing congestion. It only means that the marked Pure ACK is part of an aggregate that is collectively causing a bottleneck queue to randomly CE-mark a fraction of the packets. A CE-mark on a Pure ACK might be due to data packets in other flows through the same bottleneck, due to data packets interspersed between Pure ACKs in the same half-connection, or just due to the rate of Pure ACKs alone. (RFC 3168 only considered the last possibility, which led to the argument that standardization of ECN-enabled Pure ACKs had to be deferred, because ACK congestion control was a research issue.)

If a host has been sending a mix of Pure ACKs and data, it doesn't need to work out whether a particular CE mark was on a Pure ACK or not; it just needs to respond to congestion feedback as a whole by reducing its congestion window (cwnd), which limits the data it can launch into flight through the congested bottleneck. If a host is solely receiving data and sending only Pure ACKs, reducing cwnd will have no immediate effect (the next subsection addresses that). Nonetheless, reducing cwnd at one moment would limit its rate if it was given something to send at a later moment.

When a host is sending data as well as Pure ACKs, it would not be right for CE-marks on Pure ACKs and on data packets to induce the same reduction in cwnd. A possible way to address this issue would be to weight the response by the size of the marked packets (assuming the congestion control supports a weighted response, e.g. [RFC8257]). For instance, one could calculate the fraction of CE-marked bytes (headers and data) over each round trip (say) as follows:

> (CE-marked header bytes + CE-marked data bytes) / (all header bytes + all data bytes)

Even if the exact header size is not known, header bytes could be calculated by multiplying a packet count by a nominal header size, which is possible with AccECN feedback, because it gives a count of CE-marked packets (as well as CE-marked bytes). The above simple aggregate calculation caters for the full range of scenarios; from all Pure ACKs to just a few interspersed with data packets.

Note that any mechanism that reduces cwnd due to CE-marked Pure ACKs would need to be integrated with the congestion window validation mechanism [RFC7661], which already conservatively reduces cwnd over time because cwnd becomes stale if it is not used to fill the pipe.

4.4.2.2. ACK Rate Response to CE-Marked Pure ACKs

Reducing the congestion window will have little effect if the bottleneck is congested mostly by unresponsive pure ACKs. This could leave little or no capacity for data transfers that would be responsive to the congestion.

Since RFC 3168 was published, experimental Acknowledgement Congestion Control (AckCC) techniques have been documented in [RFC5690] (informational), which describes how two new TCP options could allow any pair of TCP endpoints to regulate the delayed ACK ratio in response to lost or CE-marked pure ACKs. However, this spec did not ask IANA to actually allocate any option numbers, because the intention was to describe the scheme and document the unresolved complications.

AckCC addressed three main problems, namely that TCP had: i) no mechanism to feed back loss or CE-marking of pure ACKs; ii) consequently, no mechanism to allow ECT to be set on pure ACKs; and iii) no mechanism to regulate the ACK rate. A combination of AccECN and the present specification addresses the first two problems, at least for ECN marking. So, with the addition of an ACK rate mechanism, it might now be possible to design an ECN-specific ACK congestion control scheme along similiar lines to RFC 5690. However, such a mechanism is out of scope of the present document.

Setting aside the unfinished nature of RFC 5690, the need for AckCC has not been conclusively demonstrated. It has been argued that the Internet has survived so far with no mechanism to even detect loss of pure ACKs. However, it has also been argued that ECN is not the same as loss. Packet discard can naturally thin the ACK load to whatever the bottleneck can support, whereas ECN marking does not (it queues the ACKs instead). Nonetheless, RFC 3168 (section 7) recommends that an AQM switches over from ECN marking to discard when the marking probability becomes high. Therefore discard can still be relied on to thin out ECN-enabled pure ACKs as a last resort.

4.4.3. Summary: Enabling ECN on Pure ACKs

In the case when AccECN has been negotiated, it provides a feasible congestion response mechanism for Pure ACKs, so the arguments for ECT on pure ACKs outweigh those against. ECN is always more and never less reliable for delivery of congestion notification. A cwnd reduction needs to be considered by congestion control designers as a response to congestion on pure ACKs. Separately, AckCC (or an improved variant exploiting AccECN) could optionally be used to regulate the spacing between pure ACKs. However, it is not clear whether AckCC is justified. If it is not, packet discard will still act as the "congestion response of last resort" by thinning out the ACK traffic. In contrast, not setting ECT on pure ACKs is certainly detrimental to performance, because when a pure ACK is lost it can prevent the release of new data.

In the case when Classic ECN has been negotiated, the argument for ECT on pure ACKs is less clear-cut. Some of the installed base of RFC 3168 implementations might happen to (unintentionally) provide a feedback mechanism to support a cwnd response. For those that did not, setting ECT on pure ACKs would be better for the flow's own performance than not setting it. However, where there was no feedback mechanism, setting ECT could do slightly more harm than not setting it. AckCC could provide a complementary response mechanism, because it is designed to work with RFC 3168 ECN, but it is incomplete. In summary, a congestion response mechanism for Pure ACKs is unlikely to be feasible with the installed base of classic ECN.

Section 3.2.3 of this specification uses a safe approach where it allows ECT on Pure ACKs if AccECN feedback has been negotiated, but not with classic RFC 3168 ECN feedback. Allowing hosts to set ECT on Pure ACKs without a feasible response mechanism could result in risk. It would certainly improve the flow's own performance, but it would slightly increase potential harm to others. Morevoer, if would set an undesirable precedent for setting ECT on packets with no mechanism to respond to any resulting congestion signals.

4.4.4. Pure ACKs: DupACK Tests

This section justifies the requirement for a host to use the additional test for incoming duplicate pure ACKs in Section 3.3.3.1, which is one of the mandatory conditions for a host to set ECT on its outgoing pure ACKs. See Section 3.2.3.2 for all three conditions, the other two being to have successfully negotiated SACK and AccECN feedback.

The AccECN spec [I-D.ietf-tcpm-accurate-ecn] mandates the 'increment-triggered ACK' rule where, "an AccECN Data Receiver MUST emit an ACK if 'n' CE marks have arrived since the previous ACK." The value of 'n' depends on whether there is newly delivered data to acknowledge. If there is not, "'n' MUST be no less than 3".

Use of ECN-capable pure ACKs by the ECN++ experiment combined with congestion at an ECN AQM at the bottleneck of the ACK path can cause AccECN to acknowledge ACKs that carry new CE information, by the above rule. This leads to repetition of the ACK of a segment, which is an exception to the requirement in the last paragraph of Sec 4.2 of [RFC5681].

This exception is justified because, although there is no new data for the receiver to acknowledge, there is new IP-ECN information to feed back. When such new IP-ECN information arrives on an ACK, it is not possible to feed it back without also repeating the acknowledgement of the latest data segment (if ECN++ had been part of TCP from the start, a logical approach would have been to clear the ACK flag, but nowadays such an unorthodox segment would be rejected). Therefore, to distinguish these ACKs of ACKs from genuine duplicate ACKs, it was necessary to introduce the test for the absence of SACK.

To justify adding this test, we use the same scenario as in Section 3.2.2.5.1 of the AccECN spec [I-D.ietf-tcpm-accurate-ecn] with volleys of unidirectional data initially from host A to B then from B to A. In response to the first volley, B emits ECN-capable pure ACKs as feedback (as per the present ECN++ spec). If the ACK stream from B experiences congestion at an ECN-enabled buffer, some of these ACKs will be CE-marked once they arrive at A.

Host A would normally inform B about the congested ACK path by piggybacking AccECN feedback on the data packets from A to B. However, once A stops sending data, it still needs to inform B about any CE-marked ACKs continuing to arrive from B in the subsequent round trip (so that B has up to date congestion information, in case it starts sending data). AccECN's 'increment-triggered ACK' rule ensures that A emits a pure ACK at least every third incoming CE mark. However, as each 'increment-triggered ACK' from A arrives at B, it will not only feed back CE markings, but it will also repeatedly acknowledge whatever the last sequence number was from B.

Then, if SACK had not been negotiated for the connection, and if B started sending its volley of new data packets, the ACKs from A could imply to B that its first new data packet had been lost and A was then emitting duplicate ACKs triggered by the rest of B's volley arriving. Then, as well as detecting CE marking, B could falsely detect loss, leading to spurious retransmission and potentially incorrect congestion response. Similarly, false detection of duplicate ACKs could confuse other algorithms, such as Limited Transmit, Fast Recovery, PRR, etc.

This is why Section 3.3.3.1 requires B to negotiate SACK and to use lack of SACK options as an additional check for a duplicate ACK.

4.5. Window Probes

Section 6.1.6 of RFC 3168 presents only the reliability argument for prohibiting ECT on Window probes:

> "If a window probe packet is dropped in the network, this loss is not detected by the receiver. Therefore, the TCP data sender MUST NOT set either an ECT codepoint or the CWR bit on window probe packets.
>
> However, because window probes use exact sequence numbers, they cannot be easily spoofed in denial-of-service attacks. Therefore, if a window probe arrives with the CE codepoint set, then the receiver SHOULD respond to the ECN indications."

The reliability argument has already been addressed in Section 4.1.

Allowing ECT on window probes could considerably improve performance because, once the receive window has reopened, if a window probe is lost the sender will stall until the next window probe reaches the receiver, which might be after the maximum retransmission timeout (at least 1 minute [RFC6928]).

On the bright side, RFC 3168 at least specifies the receiver behaviour if a CE-marked window probe arrives, so changing the behaviour ought to be less painful than for other packet types.

4.6. FINs

RFC 3168 is silent on whether a TCP sender can set ECT on a FIN. A FIN is considered as part of the sequence of data, and the rate of pure ACKs sent after a FIN could be controlled by a CE marking on the FIN. Therefore there is no reason not to set ECT on a FIN.

4.7. RSTs

RFC 3168 is silent on whether a TCP sender can set ECT on a RST. The host generating the RST message does not have an open connection after sending it (either because there was no such connection when the packet that triggered the RST message was received or because the packet that triggered the RST message also triggered the closure of the connection).

Moreover, the receiver of a CE-marked RST message can either: i) accept the RST message and close the connection; ii) emit a so-called challenge ACK in response (with suitable throttling) [RFC5961] and otherwise ignore the RST (e.g. because the sequence number is in-window but not the precise number expected next); or iii) discard the RST message (e.g. because the sequence number is out-of-window). In the first two cases there is no point in echoing any CE mark received because the sender closed its connection when it sent the RST. In the third case, given the RST is deemed invalid, any CE marking on it could also be invalid, so it makes sense to discard the CE signal as well as the RST.

Although a congestion response following a CE-marking on a RST does not appear to make sense, the following factors have been considered before deciding whether the sender ought to set ECT on a RST message:

* As explained above, a congestion response by the sender of a CE-marked RST message is not possible;

* So the only reason for the sender setting ECT on a RST would be to improve the reliability of the message's delivery;

* RST messages are used to both mount and mitigate attacks:

  - Spoofed RST messages are used by attackers to terminate ongoing connections, although the mitigations in RFC 5961 have considerably raised the bar against off-path RST attacks;

  - Legitimate RST messages allow endpoints to inform their peers to eliminate existing state that correspond to non existing connections, liberating resources e.g. in DoS attacks scenarios;

* AQMs are advised to disable ECN marking during persistent overload, so:

  - it is harder for an attacker to exploit ECN to intensify an attack;

  - it is harder for a legitimate user to exploit ECN to more reliably mitigate an attack

* Prohibiting ECT on a RST would deny the benefit of ECN to legitimate RST messages, but not to attackers who can disregard RFCs;

* If ECT were prohibited on RSTs

  - it would be easy for security middleboxes to discard all ECN-capable RSTs;

  - However, unlike a SYN flood, it is already easy for a security middlebox (or host) to distinguish a RST flood from legitimate traffic [RFC5961], and even if a some legitimate RSTs are accidentally removed as well, legitimate connections still function.

So, on balance, it has been decided that it is worth experimenting with ECT on RSTs. During experiments, if the ECN capability on RSTs is found to open a vulnerability that is hard to close, this decision can be reversed, before it is specified for the standards track.

4.8. Retransmitted Packets.

RFC 3168 says the sender "MUST NOT" set ECT on retransmitted packets. The rationale for this consumes nearly 2 pages of RFC 3168, so the reader is referred to section 6.1.5 of RFC 3168, rather than quoting it all here. There are essentially three arguments, namely: reliability; DoS attacks; and unnecessary retransmissions. We address them in order below.

The reliability argument has already been addressed in Section 4.1.

Protection against DoS attacks is not afforded by prohibiting ECT on retransmitted packets. An attacker can set ECT or CE on spoofed retransmissions whether or not it is prohibited by an RFC.

Protection against the DoS attack described in section 6.1.5 of RFC 3168 is solely afforded by the requirement that "the TCP data receiver SHOULD ignore the CE codepoint on out-of-window packets". Therefore in Section 3.2.7 the sender is allowed to set ECT on retransmitted packets, in order to reduce the chance of them being dropped. We also strengthen the receiver's requirement from "SHOULD ignore" to "MUST ignore". And we generalize the receiver's requirement to include failure of any validity check, not just out-of-window checks, in order to include the more stringent validity checks in RFC 5961 that have been developed since RFC 3168.

Finally, the third argument is about unnecessary retransmissions. For those retransmitted packets that arrive at the receiver after the original packet has been properly received (so-called spurious retransmissions), RFC 3168 raises the concern that any CE marking will be ignored, because any spurious retransmission is out of window and CE markings on out of window packets will be ignored (by the above rule). In mitigation against this argument, the fact that the original packet has been delivered implies that the sender's original congestion response (when it deemed the packet to be lost and retransmitted it) was unnecessary. However, omitting a congestion response to the CE one round trip later does not strictly compensate for the previous unnecessary response, because the response should be in the same round as the congestion occurs. Nonetheless, there is a stronger argument against the concern in RFC 3168: TCP does not detect the loss of a spurious retransmission, and therefore does not respond to this congestion loss. So not responding to CE on ECN-capable supurious retransmissions is no worse than TCP's existing lack of response to loss of spurious retransmissions.

Therefore, in all three cases, it is not incorrect to set ECT on retransmissions.

4.9. General Fall-back for any Control Packet

Extensive experiments have found no evidence of any traversal problems with ECT on any TCP control packet [Mandalari18]. Nonetheless, Sections 3.2.1.4 and 3.2.2.3 specify fall-back measures if ECT on the first packet of either half-connection (SYN or SYN-ACK) appears to be blocking progress. Here, the question of fall-back measures for ECT on other control packets is explored. It supports the advice given in Section 3.2.8, paraphrased here as, "Until there's evidence that something's broken, don't fix it."

If an implementation has had to disable ECT to ensure the first packet of a flow (SYN or SYN-ACK) gets through, the question arises whether it ought to disable ECT on all subsequent control packets within the same TCP connection. Without evidence of any such

problems, this seems unnecessarily cautious. Particularly given it would be hard to detect loss of most other types of TCP control packets that are not ACK'd. And particularly given that unnecessarily removing ECT from other control packets could lead to performance problems, e.g. by directing them into another queue [RFC9331] or over a different path, because some broken multipath equipment (erroneously) routes based on all 8 bits of the ex-Traffic Class octet (IPv6) or the ex-ToS octet (IPv4).

In the case where a connection starts without ECT on the SYN (perhaps because problems with previous connections had been cached), there will have been no test for ECT traversal in the client-server direction until the pure ACK that completes the handshake. It is possible that some middlebox might block ECT on this pure ACK or on later retransmissions of lost packets. Similarly, after a route change, the new path might include some middlebox that blocks ECT on some or all TCP control packets. However, without evidence of such problems, the complexity of a fix does not seem worthwhile.

> MORE MEASUREMENTS NEEDED (?): If further two-ended measurements do find evidence for these traversal problems, measurements would be needed to check for correlation of ECT traversal problems between different control packets. It might then be necessary to introduce a catch-all fall-back rule that disables ECT on certain subsequent TCP control packets based on some criteria developed from these measurements.

5. Interaction with popular variants or derivatives of TCP

The designs of the following TCP variants have been assessed and found not to interact adversely with ECT on TCP control packets: SYN cookies (see Appendix A of [RFC4987] and section 3.1 of [RFC5562]), Initial Window of ten (IW10 [RFC6928], TCP Fast Open (TFO [RFC7413]), DCTCP [RFC8257] and L4S [RFC9330].

The following subsections assess the interaction between setting ECT on all packets and each of these variants (except SYN cookies where no detail is necessary). A final subsection briefly notes the possibility that the principles applied here should translate to protocols derived from TCP.

This section is informative not normative, because no interactions have been identified that require any change to specifications. The subsection on IW10 discusses potential changes to specifications but recommends that no changes are needed.

5.1. IW10

IW10 is an experiment to determine whether it is safe for TCP to use an initial window of 10 SMSS [RFC6928].

This subsection does not recommend any additions to the present specification in order to interwork with IW10. The specifications as they stand are safe, and there is only a corner-case with ECT on the SYN where performance could be occasionally improved, as explained below.

As specified in Section 3.2.1.1, a TCP initiator will typically only set ECT on the SYN if it requests AccECN support. If, however, the SYN-ACK tells the initiator that the responder does not support AccECN, Section 3.2.1.1 advises the initiator to conservatively reduce its initial window, preferably to 1 SMSS because, if the SYN was CE-marked, the SYN-ACK has no way to feed that back.

If the initiator implements IW10, it seems rather over-conservative to reduce IW from 10 to 1 just in case a congestion marking was missed. Nonetheless, a reduction to 1 SMSS will rarely harm performance, because:

* as long as the initiator is caching failures to negotiate AccECN, subsequent attempts to access the same server will not use ECT on the SYN anyway, so there will no longer be any need to conservatively reduce IW;

* currently, at least for web sessions, it is extremely rare for a TCP initiator (client) to have more than one data segment to send at the start of a TCP connection (see Fig 3 in [Manzoor17]) - IW10 is primarily exploited by TCP servers.

If a responder receives feedback that the SYN-ACK was CE-marked, Section 3.2.2.2 recommends that it reduces its initial window, preferably to 1 SMSS. When the responder also implements IW10, it might again seem rather over-conservative to reduce IW from 10 to 1. But in this case the rationale is somewhat different:

* The IW10 spec [RFC6928] recommends not reducing cwnd to 1 segment on the grounds that it is uncertain whether absence of feedback implies loss. However, in contrast, explicit feedback that the SYN-ACK was CE-marked is a positive indication that the queue has been building.

* Given it is now likely that a queue already exists, the more data packets that the server sends in its IW, the more likely at least one will be CE marked, leading it to exit slow-start early.

Experimentation will be needed to determine the best strategy.

It should be noted that experience from recent congestion avoidance experiments where the window is reduced by less than half in response to ECN marking [RFC8511] is not necessarily applicable to a flow start scenario.

5.2. TFO

TCP Fast Open (TFO [RFC7413]) is an experiment to remove the round trip delay of TCP's 3-way hand-shake (3WHS). A TFO initiator caches a cookie from a previous connection with a TFO-enabled server. Then, for subsequent connections to the same server, any data included on the SYN can be passed directly to the server application, which can then return up to an initial window of response data on the SYN-ACK and on data segments straight after it, without waiting for the ACK that completes the 3WHS.

The TFO experiment and the present experiment to add ECN-support for TCP control packets can be combined without altering either specification, which is justified as follows:

* The handling of ECN marking on a SYN is no different whether or not it carries data.

* In response to any CE-marking on the SYN-ACK, the responder adopts the normal response to congestion, as discussed in Section 7.2 of [RFC7413].

5.3. L4S

A Low Latency, Low Loss, and Scalable throughput (L4S) variant of TCP such as TCP Prague [I-D.briscoe-iccrg-prague-congestion-control] requires AccECN feedback and uses ECN++. Also the spec of the L4S-ECN protocol [RFC9331] mentions ECN++ as a useful performance optimization.

Therefore, the L4S experiment and the present ECN++ experiment can be combined without altering any of the specifications. The only difference would be in the recommendation of the best SYN cache strategy.

The normative specification for ECT on a SYN in Section 3.2.1.2 recommends the "optimistic ECT and cache failures" strategy (S2B defined in Section 4.2.3) for the general Internet. However, if a user's Internet access bottleneck supported L4S ECN but not Classic ECN, the "optimistic ECT without a cache" strategy (S2A) would make most sense, because there would be little point trying to avoid the

'over-strict' test and negotiate Classic ECN, if L4S ECN but not Classic ECN was available on that user's access link (as is the case with Low Latency DOCSIS [DOCSIS3.1]).

Strategy (S2A) is the simplest, because it requires no cache. It would satisfy the goal of an implementer who is solely interested in ultra-low latency using AccECN and ECN++ (e.g. accessing L4S servers) and is not concerned about fall-back to Classic ECN (e.g. when accessing other servers).

5.4. Other transport protocols

Experience from experiments on adding ECN support to all TCP packets ought to be directly transferable between TCP and derivatives of TCP, like SCTP.

Stream Control Transmission Protocol (SCTP) [RFC9260] is a standards track transport protocol derived from TCP. SCTP currently does not include ECN support, but Appendix A of an obsoleted earlier version of the spec [RFC4960] broadly describes how it would be supported and a draft on the addition of ECN to SCTP has been produced [I-D.stewart-tsvwg-sctpecn]. The question of whether SCTP ought to set ECT on retransmissions and control packets is work in progress.

QUIC [RFC9000] is another standards track transport protocol offering similar services to TCP but intended to exploit some of the benefits of running over UDP. Building on the arguments in the current draft, a QUIC sender sets ECT on all packets unless it fails the test for path support.

6. Security Considerations

There are several security arguments presented in RFC 3168 for preventing the ECN marking of TCP control packets and retransmitted segments. We believe all of them have been properly countered in Section 4, particularly Section 4.2.4 and Section 4.8 on DoS attacks using spoofed ECT-marked SYNs and spoofed CE-marked retransmissions. In both cases, RFC 3168 attempted to legislate against the sender setting ECT, which degrades the performance of genuine senders, but will not be heeded by attackers. Instead, the approach adopted is to specify reinforced defences against such attacks that already reside in the network and at the receiver.

Section 3.2.6 on sending TCP RSTs considers the question of whether ECT on RSTs will allow RST attacks to be intensified. It also points out that implementers need to take care to ensure that the ECN field on a RST does not depend on TCP's state machine. Otherwise the internal information revealed could be of use to potential attackers.

This point actually applies more generally to all control packets (unless it has become necessary to disable ECN to evade path traversal problems).

If each TCP implementation chooses to make different control packets ECN-capable, it could contribute to easier fingerprinting of TCP stacks.

7. IANA Considerations

This section is to be removed before publishing as an RFC.

There are no IANA considerations in this memo.

8. References

8.1. Normative References

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, DOI 10.17487/RFC2018, October 1996, <https://www.rfc-editor.org/info/rfc2018>.

[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>.

[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>.

[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>.

[RFC5961] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", RFC 5961, DOI 10.17487/RFC5961, August 2010, <https://www.rfc-editor.org/info/rfc5961>.

[I-D.ietf-tcpm-accurate-ecn]
Briscoe, B., Kühlewind, M., and R. Scheffenegger, "More Accurate Explicit Congestion Notification (ECN) Feedback in TCP", Work in Progress, Internet-Draft, draft-ietf-tcpm-accurate-ecn-28, 17 November 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-accurate-ecn-28>.

[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>.

[RFC9293] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <https://www.rfc-editor.org/info/rfc9293>.

8.2. 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>.

[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>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/info/rfc4960>.

[RFC9260] Stewart, R., Tüxen, M., and K. Nielsen, "Stream Control Transmission Protocol", RFC 9260, DOI 10.17487/RFC9260, June 2022, <https://www.rfc-editor.org/info/rfc9260>.

[I-D.stewart-tsvwg-sctpecn]
Stewart, R. R. and M. Tüxen, "Explicit Congestion Notification for the Stream Control Transmission Protocol", Work in Progress, Internet-Draft, draft-stewart-tsvwg-sctpecn-06, 20 October 2023, <https://datatracker.ietf.org/doc/html/draft-stewart-tsvwg-sctpecn-06>.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <https://www.rfc-editor.org/info/rfc4987>.

[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>.

[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>.

[RFC5690] Floyd, S., Arcia, A., Ros, D., and J. Iyengar, "Adding Acknowledgement Congestion Control to TCP", RFC 5690, DOI 10.17487/RFC5690, February 2010, <https://www.rfc-editor.org/info/rfc5690>.

[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>.

[RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <https://www.rfc-editor.org/info/rfc6928>.

[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>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.

[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>.

[RFC7661] Fairhurst, G., Sathiaseelan, A., and R. Secchi, "Updating TCP to Support Rate-Limited Traffic", RFC 7661, DOI 10.17487/RFC7661, October 2015, <https://www.rfc-editor.org/info/rfc7661>.

[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>.

[RFC8511] Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst, "TCP Alternative Backoff with ECN (ABE)", RFC 8511, DOI 10.17487/RFC8511, December 2018, <https://www.rfc-editor.org/info/rfc8511>.

[RFC9040] Touch, J., Welzl, M., and S. Islam, "TCP Control Block Interdependence", RFC 9040, DOI 10.17487/RFC9040, July 2021, <https://www.rfc-editor.org/info/rfc9040>.

[RFC9331] De Schepper, K. and B. Briscoe, Ed., "The Explicit Congestion Notification (ECN) Protocol for Low Latency, Low Loss, and Scalable Throughput (L4S)", RFC 9331, DOI 10.17487/RFC9331, January 2023, <https://www.rfc-editor.org/info/rfc9331>.

[RFC9330] Briscoe, B., Ed., De Schepper, K., Bagnulo, M., and G. White, "Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture", RFC 9330, DOI 10.17487/RFC9330, January 2023, <https://www.rfc-editor.org/info/rfc9330>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

[RFC9438] Xu, L., Ha, S., Rhee, I., Goel, V., and L. Eggert, Ed., "CUBIC for Fast and Long-Distance Networks", RFC 9438, DOI 10.17487/RFC9438, August 2023, <https://www.rfc-editor.org/info/rfc9438>.

[I-D.briscoe-iccrg-prague-congestion-control] De Schepper, K., Tilmans, O., Briscoe, B., and V. Goel, "Prague Congestion Control", Work in Progress, Internet-Draft, draft-briscoe-iccrg-prague-congestion-control-03, 14 October 2023, <https://datatracker.ietf.org/doc/html/draft-briscoe-iccrg-prague-congestion-control-03>.

[judd-nsdi] Judd, G.J., "Attaining the promise and avoiding the pitfalls of TCP in the Datacenter", USENIX Symposium on Networked Systems Design and Implementation (NSDI'15) pp.145-157, May 2015, <https://www.usenix.org/node/188966>.

[ecn-pam] Trammell, B., Kühlewind, M., Boppart, D., Learmonth, I., Fairhurst, G., and R. Scheffenegger, "Enabling Internet-Wide Deployment of Explicit Congestion Notification",

Int'l Conf. on Passive and Active Network Measurement (PAM'15) pp193-205, 2015, <https://link.springer.com/chapter/10.1007/978-3-319-15509-8_15>.

[ECN-PLUS] Kuzmanovic, A., "The Power of Explicit Congestion Notification", ACM SIGCOMM 35(4):61--72, 2005, <https://dl.acm.org/citation.cfm?id=1080100>.

[Mandalari18] Mandalari, A., Lutu, A., Briscoe, B., Bagnulo, M., and Ö. Alay, "Measuring ECN++: Good News for ++, Bad News for ECN over Mobile", IEEE Communications Magazine , March 2018, <https://ieeexplore.ieee.org/document/8316790>.

[Manzoor17] Manzoor, J., Drago, I., and R. Sadre, "How HTTP/2 is changing Web traffic and how to detect it", In Proc: Network Traffic Measurement and Analysis Conference (TMA) 2017 pp.1-9, June 2017, <https://ieeexplore.ieee.org/document/8002899>.

[Kuehlewind18] Kühlewind, M., Walter, M., Learmonth, I., and B. Trammell, "Tracing Internet Path Transparency", In Proc: Network Traffic Measurement and Analysis Conference (TMA) 2018 , June 2018, <https://ieeexplore.ieee.org/document/8506532>.

[strict-ecn] Dumazet, E., "tcp: be more strict before accepting ECN negociation", Linux netdev patch list , 4 May 2012, <https://patchwork.ozlabs.org/patch/156953/>.

[relax-strict-ecn] Tilmans, O., "tcp: Accept ECT on SYN in the presence of RFC8311", Linux netdev patch list , 3 April 2019, <https://lore.kernel.org/patchwork/patch/1057812/>.

[ecn-overload] Steen, H., "Destruction Testing: Ultra-Low Delay using Dual Queue Coupled Active Queue Management", Masters Thesis, Uni Oslo , May 2017, <https://urn.nb.no/URN:NBN:no-60155>.

[DOCSIS3.1]
CableLabs, "MAC and Upper Layer Protocols Interface (MULPI) Specification, CM-SP-MULPIv3.1", Data-Over-Cable Service Interface Specifications DOCSIS® 3.1 Version i17 or later, 21 January 2019, <https://specification-search.cablelabs.com/CM-SP-MULPIv3.1>.

Acknowledgments

Thanks to Mirja Kühlewind, David Black, Padma Bhooma, Gorry Fairhurst, Michael Scharf, Yuchung Cheng and Christophe Paasch for their useful reviews. Richard Scheffenegger provided useful advice gained from implementing ECN++ for FreeBSD.

The work of Marcelo Bagnulo has been partially funded by EU under projects Stand-ICT CCI and H2020-ICT-2014-2 5G NORMA.

Bob Briscoe's contribution was partly funded by Apple Inc, partly by the Research Council of Norway through the TimeIn project, partly by CableLabs and partly by the Comcast Innovation Fund. The views expressed here are solely those of the authors.

Authors' Addresses

Marcelo Bagnulo
Universidad Carlos III de Madrid
Av. Universidad 30
28911 Leganes Madrid
Spain
Phone: 34 91 6249500
Email: marcelo@it.uc3m.es
URI: https://www.it.uc3m.es

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

Transport Services (tsv)
Internet-Draft
Intended status: Experimental
Expires: January 3, 2019

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

July 2, 2018

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

Abstract

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

Status of This Memo

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

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

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

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

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

Copyright Notice

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

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

Table of Contents

1. Introduction

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

An example of an active queue management (AQM) marking algorithm that enables the L4S service is the DualQ Coupled AQM defined in a complementary specification [I-D.ietf-tsvwg-aqm-dualq-coupled]. An example of a scalable transport that would enable the L4S service is Data Centre TCP (DCTCP), which until now has been applicable solely to controlled environments like data centres [RFC8257], because it is too aggressive to co-exist with existing TCP. However, AQMs like DualQ Coupled enable scalable transports like DCTCP to co-exist with

existing traffic, each getting roughly the same flow rate when they compete under similar conditions. Note that DCTCP will still not be safe to deploy on the Internet until it satisfies the requirements listed in Section 2.4.

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

1.1. Problem

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

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

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

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

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

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

It turns out that a TCP algorithm like DCTCP that solves TCP's scalability problem also solves the latency problem, because the finer sawteeth cause very little queuing delay. A supporting paper [DCttH15] gives the full explanation of why the design solves both the latency and the scaling problems, both in plain English and in more precise mathematical form. The explanation is summarised without the maths in the L4S architecture document [I-D.ietf-tsvwg-l4s-arch].

1.2. Terminology

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

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

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

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

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

1.3. Scope

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

The L4S identifier is an orthogonal packet classification to the Differentiated Services Code Point (DSCP [RFC2474]). Section 2.3 explains what this means in practice.

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

- updates the ECN proposed standard [RFC3168] (in certain specified cases including the present document) to relax the requirement that an ECN mark must be equivalent to a drop, both when applied by the network, and when responded to by the sender;
- changes the status of the experimental ECN nonce [RFC3540] to historic;
- makes consequent updates to the following proposed standard RFCs to reflect the above two bullets:
  - ECN for RTP [RFC6679];
  - the congestion control specifications of various DCCP congestion control identifier (CCID) profiles [RFC4341], [RFC4342], [RFC5622].

2. L4S Packet Identifier

2.1. Consensus Choice of L4S Packet Identifier: Requirements

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

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

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

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

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

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

2.2. L4S Packet Identification at Run-Time

The L4S treatment is an alternative packet marking treatment [RFC4774] to the classic ECN treatment [RFC3168]. Like classic ECN, it identifies both network and host behaviour: it identifies the marking treatment that network nodes are expected to apply to L4S packets, and it identifies packets that have been sent from hosts that are expected to comply with a broad type of behaviour.

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

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

An L4S AQM treatment follows similar codepoint transition rules to those in RFC 3168. Specifically, the ECT(1) codepoint MUST NOT be changed to any other codepoint than CE, and CE MUST NOT be changed to any other codepoint. An ECT(1) packet is classified as ECN-capable and, if congestion increases, an L4S AQM algorithm will mark the ECN field as CE for an increasing proportion of packets, otherwise forwarding packets unchanged as ECT(1). Necessary conditions for an L4S marking treatment are defined in Section 2.6. Under persistent overload an L4S marking treatment SHOULD turn off ECN marking, using drop as a congestion signal until the overload episode has subsided, as recommended for all AQMs [RFC7567] (Section 4.2.1), which follows similar advice in RFC 3168 (Section 7).

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

2.3. Interaction of the L4S Identifier with other Identifiers

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

- addresses of specific applications or hosts configured to be safe (but for example cannot set the ECN field for some temporary reason);

- certain protocols that are usually lightweight (e.g. ARP, DNS);

- specific Diffserv codepoints that indicate traffic with limited burstiness such as the EF (Expedited Forwarding) and Voice-Admit service classes or equivalent local-use DSCPs (see [I-D.briscoe-tsvwg-l4s-diffserv]).

For clarity, non-ECN identifiers, such as the examples itemized above, might be used by some network operators who believe they identify non-L4S traffic that would be safe to mix with L4S traffic. They are not alternative ways for a host to indicate that it is sending L4S packets. Only the ECT(1) and CE ECN codepoints indicate to a network element that a host is sending L4S packets - specifically that the host claims its behaviour satisfies the pre-requisite transport requirements in Section 2.4.

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

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

- A common case is where a network element only offers Default Forwarding (DF a.k.a. Best Efforts). In this case, if a packet carries ECT(1) or CE, a network element would classify it for the L4S treatment irrespective of its DSCP. And, if a packet matched any of the DSCPs in the bullet above, the network element could classify it for the L4S treatment irrespective of its ECN codepoint.

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

2.4. Pre-Requisite Transport Layer Behaviour

2.4.1. Pre-Requisite Congestion Response

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

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

Each sender in a session can use a scalable congestion control independently of the congestion control used by the receiver(s) when they send data. Therefore there might be ECT(1) packets in one direction and ECT(0) in the other.

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

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

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

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

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

- o A scalable congestion control MUST detect loss by counting in units of time, which is scalable, and MUST NOT count in units of packets (as in the 3 DupACK rule of traditional TCP), which is not scalable. Then link technologies that support L4S can remove the head-of-line blocking delay they have to introduce while trying to keep packets in tight order to avoid triggering loss detection based on counting packets (see Appendix A.1.7 for rationale).

2.4.2. Pre-Requisite Transport Feedback

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

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

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

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

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

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

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

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

2.6. The Meaning of L4S CE Relative to Drop

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

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

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

3. L4S Experiments

{ToDo: See https://tools.ietf.org/html/draft-ietf-tsvwg-ecn-experimentation-07#section-2.3 which refers to the checklist in: https://tools.ietf.org/html/rfc5706#appendix-A }

4. IANA Considerations

This specification contains no IANA considerations.

5. Security Considerations

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

6. Acknowledgements

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

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

7. References

7.1. Normative References

[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>.

7.2. Informative References

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

[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", 2015, <http://www.bobbriscoe.net/projects/latency/dctth_preprint.pdf>.

(Under submission)

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

[I-D.ietf-tcpm-accurate-ecn]
Briscoe, B., Kuehlewind, M., and R. Scheffenegger, "More Accurate ECN Feedback in TCP", draft-ietf-tcpm-accurate-ecn-06 (work in progress), March 2018.

[I-D.ietf-tcpm-generalized-ecn]
Bagnulo, M. and B. Briscoe, "ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control Packets", draft-ietf-tcpm-generalized-ecn-02 (work in progress), October 2017.

[I-D.ietf-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-03 (work in progress), March 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-04 (work in progress), March 2018.

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

[I-D.ietf-tsvwg-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.

[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>.

[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>.

[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>.

[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 2.4 that a sender has to comply with before it can set the L4S identifier in packets it sends into the Internet. As well as necessary safety improvements (requirements) this appendix also includes preferable performance improvements (optimizations).

These recommendations have become 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/Cubic congestion control on packet loss

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

Motivation: Part of the safety conditions for deploying a scalable congestion control on the public Internet is to make sure that it behaves properly when it builds a queue at a network bottleneck that has not been upgraded to support L4S. Packet loss can have many causes, but it usually has to be conservatively assumed that it is a sign of congestion. Therefore, on detecting packet loss, a scalable congestion control will need to fall back to classic congestion control behaviour. If it does not comply with this requirement it could starve classic traffic.

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

Packet loss might (rarely) occur in the case that the bottleneck is L4S capable. In this case, the sender may receive a high number of packets marked with the CE bit set and also experience a loss. Current DCTCP implementations react differently to this situation.

At least one implementation reacts only to the drop signal (e.g. by halving the CWND) and at least another DCTCP implementation reacts to both signals (e.g. by halving the CWND due to the drop and also further reducing the CWND based on the proportion of marked packet). We believe that further experimentation is needed to understand what is the best behaviour for the public Internet, which may or not be one of these existing approaches.

A.1.4. Fall back to Reno/Cubic congestion control on classic ECN bottlenecks

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

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

It would take time for endpoints to distinguish classic and L4S ECN marking. An increase in queuing delay or in delay variation would be a tell-tale sign, but it is not yet clear where a line would be drawn between the two behaviours. It might be possible to cache what was learned about the path to help subsequent attempts to detect the type of marking.

A.1.5. Reduce RTT dependence

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

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

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

A.1.6. Scaling down the congestion window

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

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

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

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

A.1.7. Measuring Reordering Tolerance in Time Units

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

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

time over which the network has to keep packets in order stays constant.

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

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

Classic TCP implementations are switching over to the time-based approach of RACK (Recent ACKnowledgements [I-D.ietf-tcpm-rack]). However, it will be many years (decades?) before networks no longer have to allow for the presence of traditional TCP senders still using the 3 DupACK rule. This specification (Section 2.4.1) says that senders are not entitled to identify packets as L4S in the IP/ECN field unless they use the time-based approach. Then networks that identify L4S traffic separately (e.g. using [I-D.ietf-tsvwg-aqm-dualq-coupled]) can know for certain that all L4S traffic is using the scalable time-based approach.

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

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

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

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

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

A.2. Scalable Transport Protocol Optimizations

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

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

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

A.2.2. Faster than Additive Increase

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

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

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

A.2.3. Faster Convergence at Flow Start

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

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

Appendix B. Alternative Identifiers

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

features of the leading alternatives. It is intended to supplement, not replace the detailed text.

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

B.1. ECT(1) and CE codepoints

Definition:

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

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

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

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

Cons:

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

ECN hard in some lower layers: It is not always possible to support ECN in an AQM acting in a buffer below the IP layer [I-D.ietf-tsvwg-ecn-encap-guidelines]. In such cases, the L4S

service would have to drop rather than mark frames even though they might contain an ECN-capable packet. However, such cases would be unusual.

Risk of reordering classic CE packets: Having to classify all CE packets as L4S risks some classic CE packets arriving early, which is a form of reordering. Reordering can cause the TCP sender to retransmit spuriously. However, one or two packets delivered early does not cause any spurious retransmissions because the subsequent packets continue to move the cumulative acknowledgement boundary forwards. Anyway, the risk of reordering would be low, because: i) it is quite unusual to experience more than one bottleneck queue on a path; ii) even then, reordering would only occur if there was simultaneous mixing of classic and L4S traffic, which would be 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;

o 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/