DNS Transport over TCP - Operational Requirements
draft-ietf-dnsop-dns-tcp-requirements-04

Abstract

This document encourages the practice of permitting DNS messages to be carried over TCP on the Internet. It also considers the consequences with this form of DNS communication and the potential operational issues that can arise when this best common practice is not upheld.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 26, 2019.

Copyright Notice

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

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

Table of Contents

1. Introduction .................................................. 3
   1.1. Requirements Language .................................. 4
2. Background .................................................... 4
   2.1. Uneven Transport Usage and Preference ................. 4
   2.2. Waiting for Large Messages and Reliability ........... 5
   2.3. EDNS0 .................................................. 5
   2.4. Fragmentation and Truncation ........................... 6
   2.5. "Only Zone Transfers Use TCP" .......................... 7
3. DNS over TCP Requirements ..................................... 7
4. Network and System Considerations ............................ 8
   4.1. Connection Admission .................................... 8
   4.2. Connection Management ................................... 9
   4.3. Connection Termination .................................. 10
5. DNS over TCP Filtering Risks .................................. 10
   5.1. DNS Wedgie .............................................. 11
   5.2. DNS Root Zone KSK Rollover ............................ 11
   5.3. DNS-over-TLS ........................................... 11
6. Logging and Monitoring ......................................... 12
7. Acknowledgments ............................................... 12
8. IANA Considerations .......................................... 13
9. Security Considerations ....................................... 13
10. Privacy Considerations ....................................... 13
11. Examples ...................................................... 13
12. References ................................................... 13
   12.1. Normative References ................................... 13
   12.2. Informative References ................................ 13
Appendix A. Standards Related to DNS Transport over TCP ....... 19
   A.1. IETF RFC 1035 - DOMAIN NAMES - IMPLEMENTATION AND
        SPECIFICATION ......................................... 19
   A.2. IETF RFC 1536 - Common DNS Implementation Errors and
        Suggested Fixes ......................................... 19
   A.3. IETF RFC 1995 - Incremental Zone Transfer in DNS ....... 19
   A.4. IETF RFC 1996 - A Mechanism for Prompt Notification of
        Zone Changes (DNS NOTIFY) ............................. 20
   A.5. IETF RFC 2181 - Clarifications to the DNS Specification .. 20
   A.6. IETF RFC 2694 - DNS extensions to Network Address
        Translators (DNS_ALG) .................................. 20
   A.7. IETF RFC 3225 - Indicating Resolver Support of DNSSEC .. 20
   A.8. IETF RFC 3326 - DNSSEC and IPv6 A6 aware server/resolver
        message size requirements ................................ 20
   A.9. IETF RFC 4472 - Operational Considerations and Issues
        with IPv6 DNS ........................................... 21
   A.10. IETF RFC 5452 - Measures for Making DNS More Resilient
1. Introduction

DNS messages may be delivered using UDP or TCP communications. While most DNS transactions are carried over UDP, some operators have been led to believe that any DNS over TCP traffic is unwanted or unnecessary for general DNS operation. When DNS over TCP has been restricted, a variety of communication failures and debugging challenges often arise. As DNS and new naming system features have evolved, TCP as a transport has become increasingly important for the correct and safe operation of an Internet DNS. Reflecting modern usage, the DNS standards were recently updated to declare support for TCP is now a required part of the DNS implementation.
This document is the formal requirements equivalent for the operational community, encouraging system administrators, network engineers, and security staff to ensure DNS over TCP communications support is on par with DNS over UDP communications.

1.1. Requirements Language

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

2. Background

The curious state of disagreement in operational best practices and guidance for DNS transport protocols derives from conflicting messages operators have gotten from other operators, implementors, and even the IETF. Sometimes these mixed signals have been explicit, on other occasions they have suspiciously implicit. This section presents an interpretation of the storied and conflicting history that led to this document.

2.1. Uneven Transport Usage and Preference

In the original suite of DNS specifications, [RFC1034] and [RFC1035] clearly specified that DNS messages could be carried in either UDP or TCP, but they also stated a preference for UDP as the best transport for queries in the general case. As stated in [RFC1035]:

"While virtual circuits can be used for any DNS activity, datagrams are preferred for queries due to their lower overhead and better performance."

Another early, important, and influential document, [RFC1123], marked the preference for a transport protocol more explicitly:

"DNS resolvers and recursive servers MUST support UDP, and SHOULD support TCP, for sending (non-zone-transfer) queries."

and further stipulated:

"A name server MAY limit the resources it devotes to TCP queries, but it SHOULD NOT refuse to service a TCP query just because it would have succeeded with UDP."

Culminating in [RFC1536], DNS over TCP came to be associated primarily with the zone transfer mechanism, while most DNS queries and responses were seen as the dominion of UDP.
2.2. Waiting for Large Messages and Reliability

In the original specifications, the maximum DNS over UDP message size was enshrined at 512 bytes. However, even while [RFC1123] preferred UDP for non-zone transfer queries, it foresaw DNS over TCP becoming more popular in the future to overcome this limitation:

"[...] it is also clear that some new DNS record types defined in the future will contain information exceeding the 512 byte limit that applies to UDP, and hence will require TCP.

At least two new, widely anticipated developments were set to elevate the need for DNS over TCP transactions. The first was dynamic updates defined in [RFC2136] and the second was the set of extensions collectively known as DNSSEC originally specified in [RFC2541]. The former suggested "requestors who require an accurate response code must use TCP", while the later warned "[...] larger keys increase the size of KEY and SIG RRs. This increases the chance of DNS UDP packet overflow and the possible necessity for using higher overhead TCP in responses."

Yet defying some expectations, DNS over TCP remained little used in real traffic across the Internet. Dynamic updates saw little deployment between autonomous networks. Around the time DNSSEC was first defined, another new feature helped solidify UDP transport dominance for message transactions.

2.3. EDNS0

In 1999 the IETF published the Extension Mechanisms for DNS (EDNS0) in [RFC2671] (superseded in 2013 by an update in [RFC6891]). This document standardized a way for communicating DNS nodes to perform rudimentary capabilities negotiation. One such capability written into the base specification and present in every EDNS0 compatible message is the value of the maximum UDP payload size the sender can support. This unsigned 16-bit field specifies in bytes the maximum (possibly fragmented) DNS message size a node is capable of receiving. In practice, typical values are a subset of the 512 to 4096 byte range. EDNS0 became widely deployed over the next several years and numerous surveys have shown many systems currently support larger UDP MTUs [CASTRO2010], [NETALYZR] with EDNS0.

The natural effect of EDNS0 deployment meant DNS messages larger than 512 bytes would be less reliant on TCP than they might otherwise have been. While a non-negligible population of DNS systems lacked EDNS0 or fall back to TCP when necessary, DNS over TCP transactions remained a very small fraction of overall DNS traffic [VERISIGN].
2.4. Fragmentation and Truncation

Although EDNS0 provides a way for endpoints to signal support for DNS messages exceeding 512 bytes, the realities of a diverse and inconsistently deployed Internet may result in some large messages being unable to reach their destination. Any IP datagram whose size exceeds the MTU of a link it transits will be fragmented and then reassembled by the receiving host. Unfortunately, it is not uncommon for middleboxes and firewalls to block IP fragments. If one or more fragments do not arrive, the application does not receive the message and the request times out.

For IPv4-connected hosts, the de-facto MTU is often the Ethernet payload size of 1500 bytes. This means that the largest unfragmented UDP DNS message that can be sent over IPv4 is likely 1472 bytes. For IPv6, the situation is a little more complicated. First, IPv6 headers are 40 bytes (versus 20 without options in IPv4). Second, it seems as though some people have mis-interpreted IPv6’s required minimum MTU of 1280 as a required maximum. Third, fragmentation in IPv6 can only be done by the host originating the datagram. The need to fragment is conveyed in an ICMPv6 "packet too big" message. The originating host indicates a fragmented datagram with IPv6 extension headers. Unfortunately, it is quite common for both ICMPv6 and IPv6 extension headers to be blocked by middleboxes. According to [HUSTON] some 35% of IPv6-capable recursive resolvers were unable to receive a fragmented IPv6 packet.

The practical consequence of all this is that DNS requestors must be prepared to retry queries with different EDNS0 maximum message size values. Administrators of BIND are likely to be familiar with seeing "success resolving ... after reducing the advertised EDNS0 UDP packet size to 512 octets" messages in their system logs.

Often, reducing the EDNS0 UDP packet size leads to a successful response. That is, the necessary data fits within the smaller message size. However, when the data does not fit, the server sets the truncated flag in its response, indicating the client should retry over TCP to receive the whole response. This is undesirable from the client’s point of view because it adds more latency, and potentially undesirable from the server’s point of view due to the increased resource requirements of TCP.

The issues around fragmentation, truncation, and TCP are driving certain implementation and policy decisions in the DNS. Notably, Cloudflare implemented what it calls "DNSSEC black lies" [CLOUDFLARE] and uses ECDSA algorithms, such that their signed responses fit easily in 512 bytes. The KSK Rollover design team [DESIGNE TEAM] spent a lot of time thinking and worrying about response sizes. There is
growing sentiment in the DNSSEC community that RSA key sizes beyond 2048-bits are impractical and that critical infrastructure zones should transition to elliptic curve algorithms to keep response sizes manageable.

2.5. "Only Zone Transfers Use TCP"

Today, the majority of the DNS community expects, or at least has a desire, to see DNS over TCP transactions occur without interference. However there has also been a long held belief by some operators, particularly for security-related reasons, that DNS over TCP services should be purposely limited or not provided at all [CHES94], [DJBDNS]. A popular meme has also held the imagination of some that DNS over TCP is only ever used for zone transfers and is generally unnecessary otherwise, with filtering all DNS over TCP traffic even described as a best practice.

The position on restricting DNS over TCP had some justification given that historic implementations of DNS nameservers provided very little in the way of TCP connection management (for example see Section 6.1.2 of [RFC7766] for more details). However modern standards and implementations are nearing parity with the more sophisticated TCP management techniques employed by, for example, HTTP(S) servers and load balancers.

3. DNS over TCP Requirements

An average increase in DNS message size, the continued development of new DNS features [Appendix A], and a denial of service mitigation technique [Section 9] have suggested that DNS over TCP transactions are as important to the correct and safe operation of the Internet DNS as ever, if not more so. Furthermore, there has been serious research that argues connection-oriented DNS transactions may provide security and privacy advantages over UDP transport. [TDNS] In fact [RFC7858], a Standards Track document, is just this sort of specification. Therefore, this document makes explicit that it is undesirable for network operators to artificially inhibit DNS over TCP transport.

Section 6.1.3.2 in [RFC1123] is updated: All DNS resolvers and servers MUST support and service both UDP and TCP queries.

- Authoritative servers MUST support and service all TCP queries so that they do not limit the size of responses to what fits in a single UDP packet.
Recursive servers (or forwarders) MUST support and service all TCP queries so that they do not prevent large responses from a TCP-capable server from reaching its TCP-capable clients.

Regarding the choice of limiting the resources a server devotes to queries, Section 6.1.3.2 in [RFC1123] also says:

"A name server MAY limit the resources it devotes to TCP queries, but it SHOULD NOT refuse to service a TCP query just because it would have succeeded with UDP."

This requirement is hereby updated: A name server MAY limit the resources it devotes to queries, but it MUST NOT refuse to service a query just because it would have succeeded with another transport protocol.

Filtering of DNS over TCP is considered harmful in the general case. DNS resolver and server operators MUST support and provide DNS service over both UDP and TCP transports. Likewise, network operators MUST allow DNS service over both UDP and TCP transports. It is acknowledged that DNS over TCP service can pose operational challenges that are not present when running DNS over UDP alone, and vice-versa. However, it is the aim of this document to argue that the potential damage incurred by prohibiting DNS over TCP service is more detrimental to the continued utility and success of the DNS than when its usage is allowed.

4. Network and System Considerations

This section describes measures that systems and applications can take to optimize performance over TCP and to protect themselves from TCP-based resource exhaustion and attacks.

4.1. Connection Admission

The SYN flooding attack is a denial-of-service method affecting hosts that run TCP server processes [RFC4987]. This attack can be very effective if not mitigated. One of the most effective mitigation techniques is SYN cookies, which allows the server to avoid allocating any state until the successful completion of the three-way handshake.

Services not intended for use by the public Internet, such as most recursive name servers, SHOULD be protected with access controls. Ideally these controls are placed in the network, well before before any unwanted TCP packets can reach the DNS server host or application. If this is not possible, the controls can be placed in the application itself. In some situations (e.g. attacks) it may be...
necessary to deploy access controls for DNS services that should otherwise be globally reachable.

The FreeBSD operating system has an "accept filter" feature that postpones delivery of TCP connections to applications until a complete, valid request has been received. The dns_accf(9) filter ensures that a valid DNS message is received. If not, the bogus connection never reaches the application. Applications must be coded and configured to make use of this filter.

Per [RFC7766], applications and administrators are advised to remember that TCP MAY be used before sending any UDP queries. Networks and applications MUST NOT be configured to refuse TCP queries that were not preceded by a UDP query.

TCP Fast Open [RFC7413] (TFO) allows TCP clients to shorten the handshake for subsequent connections to the same server. TFO saves one round-trip time in the connection setup. DNS servers SHOULD enable TFO when possible. Furthermore, DNS servers clustered behind a single service address (e.g., anycast or load-balancing), SHOULD use the same TFO server key on all instances.

DNS clients SHOULD also enable TFO when possible. Currently, on some operating systems it is not implemented or disabled by default. [WIKIPEDIA_TFO] describes applications and operating systems that support TFO.

### 4.2. Connection Management

Since host memory for TCP state is a finite resource, DNS servers MUST actively manage their connections. Applications that do not actively manage their connections can encounter resource exhaustion leading to denial of service. For DNS, as in other protocols, there is a tradeoff between keeping connections open for potential future use and the need to free up resources for new connections that will arrive.

DNS server software SHOULD provide a configurable limit on the total number of established TCP connections. If the limit is reached, the application is expected to either close existing (idle) connections or refuse new connections. Operators SHOULD ensure the limit is configured appropriately for their particular situation.

DNS server software MAY provide a configurable limit on the number of established connections per source IP address or subnet. This can be used to ensure that a single or small set of users can not consume all TCP resources and deny service to other users. Operators SHOULD
ensure this limit is configured appropriately, based on their number of diversity of users.

DNS server software SHOULD provide a configurable timeout for idle TCP connections. For very busy name servers this might be set to a low value, such as a few seconds. For less busy servers it might be set to a higher value, such as tens of seconds. DNS clients and servers SHOULD signal their timeout values using the edns-tcp-keepalive option.[RFC7828]

DNS server software MAY provide a configurable limit on the number of transactions per TCP connection. This document does not offer advice on particular values for such a limit.

Similarly, DNS server software MAY provide a configurable limit on the total duration of a TCP connection. This document does not offer advice on particular values for such a limit.

Since clients may not be aware of server-imposed limits, clients utilizing TCP for DNS need to always be prepared to re-establish connections or otherwise retry outstanding queries.

4.3. Connection Termination

In general, it is preferable for clients to initiate the close of a TCP connection. The TCP peer that initiates a connection close retains the socket in the TIME_WAIT state for some amount of time, possibly a few minutes. On a busy server, the accumulation of many sockets in TIME_WAIT can cause performance problems or even denial of service.

On systems where large numbers of sockets in TIME_WAIT are observed, it may be beneficial to tune the local TCP parameters. For example, the Linux kernel provides a number of "syscall" parameters related to TIME_WAIT, such as net.ipv4.tcp_fin_timeout, net.ipv4.tcp_tw_recycle, and net.ipv4.tcp_tw_reuse. In extreme cases, implementors and operators of very busy servers may find it necessary to utilize the SO_LINGER socket option ([Stevens] Section 7.5) with a value of zero so that the server doesn’t accumulate TIME_WAIT sockets.

5. DNS over TCP Filtering Risks

Networks that filter DNS over TCP risk losing access to significant or important pieces of the DNS name space. For a variety of reasons a DNS answer may require a DNS over TCP query. This may include large message sizes, lack of EDNS0 support, DDoS mitigation techniques, or perhaps some future capability that is as yet unforeseen will also demand TCP transport.
For example, [RFC7901] describes a latency-avoiding technique that sends extra data in DNS responses. This makes responses larger and potentially increases the risk of DDoS reflection attacks. The specification mandates the use of TCP or DNS Cookies. [RFC7873]

Even if any or all particular answers have consistently been returned successfully with UDP in the past, this continued behavior cannot be guaranteed when DNS messages are exchanged between autonomous systems. Therefore, filtering of DNS over TCP is considered harmful and contrary to the safe and successful operation of the Internet. This section enumerates some of the known risks known at the time of this writing when networks filter DNS over TCP.

5.1. DNS Wedgie

Networks that filter DNS over TCP may inadvertently cause problems for third party resolvers as experienced by [TOYAMA]. If for instance a resolver receives a truncated answer from a server, but when the resolver resends the query using TCP and the TCP response never arrives, not only will a complete answer be unavailable, but the resolver will incur the full extent of TCP retransmissions and time outs. This situation might place extreme strain on resolver resources. If the number and frequency of these truncated answers are sufficiently high, the steady-state of lost resources as a result is a "DNS" wedgie". A DNS wedgie is generally not easily or completely mitigated by the affected DNS resolver operator.

5.2. DNS Root Zone KSK Rollover

Recent plans for a new root zone DNSSEC KSK have highlighted a potential problem in retrieving the keys. [LEWIS] Some packets in the KSK rollover process will be larger than 1280 bytes, the IPv6 minimum MTU for links carrying IPv6 traffic. [RFC2460] While studies have shown that problems due to fragment filtering or an inability to generate and receive these larger messages are negligible, any DNS server that is unable to receive large DNS over UDP messages or perform DNS over TCP may experience severe disruption of DNS service if performing DNSSEC validation.

TODO: Is this "overcome by events" now? We’ve had 1414 byte DNSKEY responses at the three ZSK rollover periods since KSK-2017 became published in the root zone.

5.3. DNS-over-TLS

DNS messages may be sent over TLS to provide privacy between stubs and recursive resolvers. [RFC7858] is a standards track document describing how this works. Although it utilizes TCP port 853 instead
of port 53, this document applies equally well to DNS-over-TLS. Note, however, DNS-over-TLS is currently only defined between stubs and recursives.

The use of TLS places even strong operational burdens on DNS clients and servers. Cryptographic functions for authentication and encryption require additional processing. Unoptimized connection setup takes two additional round-trips compared to TCP, but can be reduced with Fast TLS connection resumption [RFC5077] and TLS False Start [RFC7918].

6. Logging and Monitoring

Developers of applications that log or monitor DNS are advised to not ignore TCP because it is rarely used or because it is hard to process. Operators are advised to ensure that their monitoring and logging applications properly capture DNS-over-TCP messages. Otherwise, attacks, exfiltration attempts, and normal traffic may go undetected.

DNS messages over TCP are in no way guaranteed to arrive in single segments. In fact, a clever attacker may attempt to hide certain messages by forcing them over very small TCP segments. Applications that capture network packets (e.g., with libpcap) should be prepared to implement and perform full TCP segment reassembly. dnscap [dnscap] is an open-source example of a DNS logging program that implements TCP reassembly.

Developers should also keep in mind connection reuse, pipelining, and out-of-order responses when building and testing DNS monitoring applications.

7. Acknowledgments

This document was initially motivated by feedback from students who pointed out that they were hearing contradictory information about filtering DNS over TCP messages. Thanks in particular to a teaching colleague, JPL, who perhaps unknowingly encouraged the initial research into the differences of what the community has historically said and did. Thanks to all the NANOG 63 attendees who provided feedback to an early talk on this subject.

The following individuals provided an array of feedback to help improve this document: Sara Dickinson, Bob Harold, Tatuya Jinmei, and Paul Hoffman. The authors are also indebted to the contributions stemming from discussion in the tcpm working group meeting at IETF 104. Any remaining errors or imperfections are the sole responsibility of the document authors.
8. IANA Considerations

This memo includes no request to IANA.

9. Security Considerations

Ironically, returning truncated DNS over UDP answers in order to induce a client query to switch to DNS over TCP has become a common response to source address spoofed, DNS denial-of-service attacks [RRL]. Historically, operators have been wary of TCP-based attacks, but in recent years, UDP-based flooding attacks have proven to be the most common protocol attack on the DNS. Nevertheless, a high rate of short-lived DNS transactions over TCP may pose challenges. While many operators have provided DNS over TCP service for many years without duress, past experience is no guarantee of future success.

DNS over TCP is not unlike many other Internet TCP services. TCP threats and many mitigation strategies have been well documented in a series of documents such as [RFC4953], [RFC4987], [RFC5927], and [RFC5961].

10. Privacy Considerations

TODO: Does this document warrant privacy considerations?

11. Examples

Suggestion from IETF104 to include example config snippets ala 7706.

12. References

12.1. Normative References


12.2. Informative References


[RRL] Vixie, P. and V. Schryver, "DNS Response Rate Limiting (DNS RRL)", ISC-TN 2012-1 Draft1, April 2012.
Appendix A. Standards Related to DNS Transport over TCP

This section enumerates all known IETF RFC documents that are currently of status standard, informational, best common practice or experimental and either implicitly or explicitly make assumptions or statements about the use of TCP as a transport for the DNS germane to this document.

A.1. IETF RFC 1035 - DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION

The internet standard [RFC1035] is the base DNS specification that explicitly defines support for DNS over TCP.

A.2. IETF RFC 1536 - Common DNS Implementation Errors and Suggested Fixes

The informational document [RFC1536] states UDP is the "chosen protocol for communication though TCP is used for zone transfers." That statement should now be considered in its historical context and is no longer a proper reflection of modern expectations.

A.3. IETF RFC 1995 - Incremental Zone Transfer in DNS

The [RFC1995] standards track document documents the use of TCP as the fallback transport when IXFR responses do not fit into a single UDP response. As with AXFR, IXFR messages are typically delivered over TCP by default in practice. XXX: is this an accurate statement?
A.4. IETF RFC 1996 - A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)

The [RFC1996] standards track document suggests a zone master may decide to issue NOTIFY messages over TCP. In practice NOTIFY messages are generally sent over UDP, but this specification leaves open the possibility that the choice of transport protocol is up to the master, and therefore a slave ought to be able to operate over both UDP and TCP.

A.5. IETF RFC 2181 - Clarifications to the DNS Specification

The [RFC2181] standards track document includes clarifying text on how a client should react to the TC flag set on responses. It is advised the the response should be discarded and the query resent using TCP.

A.6. IETF RFC 2694 - DNS extensions to Network Address Translators (DNS_ALG)

The informational document [RFC2694] enumerates considerations for network address translation (NAT) middle boxes to properly handle DNS traffic. This document is noteworthy in its suggestion that DNS over TCP is "[t]ypically" used for zone transfer requests, further evidence that helps explain why DNS over TCP may often have been treated very differently than DNS over UDP in operational networks.

A.7. IETF RFC 3225 - Indicating Resolver Support of DNSSEC

The [RFC3225] standards track document makes statements indicating DNS over TCP is "detrimental" as a result of increased traffic, latency, and server load. This document is a companion to the next document in the RFC series expressing the requirement for EDNS0 support for DNSSEC.

A.8. IETF RFC 3326 - DNSSEC and IPv6 A6 aware server/resolver message size requirements

The [RFC3326] standards track document, although updated by later DNSSEC strongly argued in favor of UDP messages over TCP largely for performance reasons. The document declares EDNS0 a requirement for DNSSEC servers and advocated packet fragmentation may be preferable to TCP in certain situations.
A.9. IETF RFC 4472 - Operational Considerations and Issues with IPv6 DNS

This informational document [RFC4472] notes that IPv6 data may increase DNS responses beyond what would fit in a UDP message. Particularly noteworthy, perhaps less common today then when this document was written, refers to implementations that truncate data without setting the TC bit to encourge the client to resend the query using TCP.

A.10. IETF RFC 5452 - Measures for Making DNS More Resilient against Forged Answers

This informational document [RFC5452] arose as public DNS systems began to experience widespread abuse from spoofed queries, resulting in amplification and reflection attacks against unwitting victims. One of the leading justifications for supporting DNS over TCP to thwart these attacks is briefly described in this document’s 9.3 Spoof Detection and Countermeasure section.

A.11. IETF RFC 5507 - Design Choices When Expanding the DNS

This informational document [RFC5507] was largely an attempt to dissuade new DNS data types from overloading the TXT resource record type. In so doing it summarizes the conventional wisdom of DNS design and implementation practices. The authors suggest TCP overhead and stateful properties pose challenges compared to UDP, and imply that UDP is generally preferred for performance and robustness.

A.12. IETF RFC 5625 - DNS Proxy Implementation Guidelines

This best current practice document [RFC5625] provides DNS proxy implementation guidance including the mandate that a proxy "MUST [...] be prepared to receive and forward queries over TCP" even though it suggests historically TCP transport has not been strictly mandatory in stub resolvers or recursive servers.

A.13. IETF RFC 5936 - DNS Zone Transfer Protocol (AXFR)

The [RFC5936] standards track document provides a detailed specification for the zone transfer protocol, as originally outlined in the early DNS standards. AXFR operation is limited to TCP and not specified for UDP. This document discusses TCP usage at length.
A.14. IETF RFC 5966 - DNS Transport over TCP - Implementation Requirements

This standards track document [RFC5966] instructs DNS implementers to provide support for carrying DNS over TCP messages in their software. The authors explicitly make no recommendations to operators, which we seek to address here.

A.15. IETF RFC 6304 - AS112 Nameserver Operations

[RFC6304] is an informational document enumerating the requirements for operation of AS112 project DNS servers. New AS112 nodes are tested for their ability to provide service on both UDP and TCP transports, with the implication that TCP service is an expected part of normal operations.

A.16. IETF RFC 6762 - Multicast DNS

In this standards track document [RFC6762] the TC bit is deemed to have essentially the same meaning as described in the original DNS specifications. That is, if a response with the TCP bit set is receiver "[...] the querier SHOULD reissue its query using TCP in order to receive the larger response."

A.17. IETF RFC 6891 - Extension Mechanisms for DNS (EDNS(0))

This standards track document [RFC6891] helped slow the use and need for DNS over TCP messages. This document highlights concerns over server load and scalability in widespread use of DNS over TCP.

A.18. IETF RFC 6950 - Architectural Considerations on Application Features in the DNS

An informational document [RFC6950] that draws attention to large data in the DNS. TCP is referenced in the context as a common fallback mechanism and counter to some spoofing attacks.

A.19. IETF RFC 7477 - Child-to-Parent Synchronization in DNS

This standards track document [RFC7477] specifies a RRType and protocol to signal and synchronize NS, A, and AAAA resource record changes from a child to parent zone. Since this protocol may require multiple requests and responses, it recommends utilizing DNS over TCP to ensure the conversation takes place between a consistent pair of end nodes.
A.20. IETF RFC 7720 - DNS Root Name Service Protocol and Deployment Requirements

This best current practice[RFC7720] declares root name service "MUST support UDP [RFC768] and TCP [RFC793] transport of DNS queries and responses."

A.21. IETF RFC 7766 - DNS Transport over TCP - Implementation Requirements

The standards track document [RFC7766] might be considered the direct ancestor of this operational requirements document. The implementation requirements document codifies mandatory support for DNS over TCP in compliant DNS software.

A.22. IETF RFC 7828 - The edns-tcp-keepalive EDNS0 Option

This standards track document [RFC7828] defines an EDNS0 option to negotiate an idle timeout value for long-lived DNS over TCP connections. Consequently, this document is only applicable and relevant to DNS over TCP sessions and between implementations that support this option.

A.23. IETF RFC 7858 - Specification for DNS over Transport Layer Security (TLS)

This standards track document [RFC7858] defines a method for putting DNS messages into a TCP-based encrypted channel using TLS. This specification is noteworthy for explicitly targeting the stub-to-recursive traffic, but does not preclude its application from recursive-to-authoritative traffic.

A.24. IETF RFC 7873 - Domain Name System (DNS) Cookies

This standards track document [RFC7873] describes an EDNS0 option to provide additional protection against query and answer forgery. This specification mentions DNS over TCP as a reasonable fallback mechanism when DNS Cookies are not available. The specification does make mention of DNS over TCP processing in two specific situations. In one, when a server receives only a client cookie in a request, the server should consider whether the request arrived over TCP and if so, it should consider accepting TCP as sufficient to authenticate the request and respond accordingly. In another, when a client receives a BADCOOKIE reply using a fresh server cookie, the client should retry using TCP as the transport.
A.25. IETF RFC 7901 - CHAIN Query Requests in DNS

This experimental specification [RFC7901] describes an EDNS0 option that can be used by a security-aware validating resolver to request and obtain a complete DNSSEC validation path for any single query. This document requires the use of DNS over TCP or a source IP address verified transport mechanism such as EDNS-COOKIE.[RFC7873]

A.26. IETF RFC 8027 - DNSSEC Roadblock Avoidance

This document [RFC8027] details observed problems with DNSSEC deployment and mitigation techniques. Network traffic blocking and restrictions, including DNS over TCP messages, are highlighted as one reason for DNSSEC deployment issues. While this document suggests these sorts of problems are due to "non-compliant infrastructure" and is of type BCP, the scope of the document is limited to detection and mitigation techniques to avoid so-called DNSSEC roadblocks.

A.27. IETF RFC 8094 - DNS over Datagram Transport Layer Security (DTLS)

This experimental specification [RFC8094] details a protocol that uses a datagram transport (UDP), but stipulates that "DNS clients and servers that implement DNS over DTLS MUST also implement DNS over TLS in order to provide privacy for clients that desire Strict Privacy [...]". This requirement implies DNS over TCP must be supported in case the message size is larger than the path MTU.

A.28. IETF RFC 8162 - Using Secure DNS to Associate Certificates with Domain Names for S/MIME

This experimental specification [RFC8162] describes a technique to authenticate user X.509 certificates in an S/MIME system via the DNS. The document points out that the new experimental resource record types are expected to carry large payloads, resulting in the suggestion that "applications SHOULD use TCP -- not UDP -- to perform queries for the SMIMEA resource record."

A.29. IETF RFC 8324 - DNS Privacy, Authorization, Special Uses, Encoding, Characters, Matching, and Root Structure: Time for Another Look?

An informational document [RFC8324] that briefly discusses the common role and challenges of DNS over TCP throughout the history of DNS.
A.30. IETF RFC 8467 - Padding Policies for Extension Mechanisms for DNS (EDNS(0))

An experimental document [RFC8467] reminds implementers to consider the underlying transport protocol (e.g. TCP) when calculating the padding length when artificially increasing the DNS message size with an EDNS(0) padding option.

A.31. IETF RFC 8483 - Yeti DNS Testbed

This informational document [RFC8483] describes a testbed environment that highlights some DNS over TCP behaviors, including issues involving packet fragmentation and operational requirements for TCP stream assembly in order to conduct DNS measurement and analysis.

A.32. IETF RFC 8484 - DNS Queries over HTTPS (DoH)

This standards track document [RFC8484] defines a protocol for sending DNS queries and responses over HTTPS. This specification assumes TLS and TCP for the underlying security and transport layers respectively. Self-described as a a technique that more closely resembles a tunneling mechanism, DoH nevertheless likely implies DNS over TCP in some sense if not directly.

A.33. IETF RFC 8490 - DNS Stateful Operations

This standards track document [RFC8490] updates the base protocol specification with a new OPCODE to help manage stateful operations in persistent sessions such as those that might be used by DNS over TCP.

Authors’ Addresses

John Kristoff
DePaul University
Chicago, IL  60604
US

Phone: +1 312 493 0305
Email: jtk@depaul.edu
URI:   https://aharp.iorc.depaul.edu

Kristoff & Wessels             Expires December 26, 2019
Abstract

This document describes loss detection and congestion control mechanisms for QUIC.

Note to Readers

Discussion of this draft takes place on the QUIC working group mailing list (quic@ietf.org), which is archived at https://mailarchive.ietf.org/arch/search/?email_list=quic [1].

Working Group information can be found at https://github.com/quicwg [2]; source code and issues list for this draft can be found at https://github.com/quicwg/base-drafts/labels/-recovery [3].

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 October 25, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.
This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction .................................................. 4
2. Conventions and Definitions ................................ 4
3. Design of the QUIC Transmission Machinery .................. 5
   3.1. Relevant Differences Between QUIC and TCP ............ 5
      3.1.1. Separate Packet Number Spaces .................. 6
      3.1.2. Monotonically Increasing Packet Numbers .......... 6
      3.1.3. No Reneging .................................... 6
      3.1.4. More ACK Ranges ................................ 7
      3.1.5. Explicit Correction For Delayed Acknowledgements 7
4. Generating Acknowledgements .................................. 7
   4.1. Crypto Handshake Data ................................... 7
   4.2. ACK Ranges .............................................. 8
   4.3. Receiver Tracking of ACK Frames ....................... 8
   4.4. Measuring and Reporting Host Delay .................... 8
5. Estimating the Round-Trip Time ............................... 9
   5.1. Generating RTT samples .................................. 9
   5.2. Estimating min_rtt ..................................... 10
   5.3. Estimating smoothed_rtt and rttvar ..................... 10
6. Loss Detection .................................................. 11
   6.1. Acknowledgement-based Detection ....................... 11
      6.1.1. Packet Threshold .................................. 12
      6.1.2. Time Threshold .................................. 12
   6.2. Crypto Retransmission Timeout ........................... 13
      6.2.1. Retry and Version Negotiation .................... 14
      6.2.2. Discarding Keys and Packet State ................ 14
   6.3. Probe Timeout ........................................... 15
      6.3.1. Computing PTO .................................... 15
      6.3.2. Sending Probe Packets ........................... 16
      6.3.3. Loss Detection .................................. 17
   6.4. Discussion .............................................. 17
7. Congestion Control ............................................. 17
   7.1. Explicit Congestion Notification ....................... 17
   7.2. Slow Start ............................................. 18
   7.3. Congestion Avoidance .................................. 18
   7.4. Recovery Period ....................................... 18
   7.5. Ignoring Loss of Undecryptable Packets ................ 18
### 7.6. Probe Timeout

8. Security Considerations

8.1. Congestion Signals

8.2. Traffic Analysis

8.3. Misreporting ECN Markings

9. IANA Considerations

10. References

10.1. Normative References

10.2. Informative References

10.3. URIs

Appendix A. Loss Recovery Pseudocode

A.1. Tracking Sent Packets

A.1.1. Sent Packet Fields

A.2. Constants of interest

A.3. Variables of interest

A.4. Initialization

A.5. On Sending a Packet

A.6. On Receiving an Acknowledgment

A.7. On Packet Acknowledgment

A.8. Setting the Loss Detection Timer

A.9. On Timeout

A.10. Detecting Lost Packets

Appendix B. Congestion Control Pseudocode

B.1. Constants of interest

B.2. Variables of interest

B.3. Initialization

B.4. On Packet Sent

B.5. On Packet Acknowledgement

B.6. On New Congestion Event

B.7. Process ECN Information

B.8. On Packets Lost

Appendix C. Change Log

C.1. Since draft-ietf-quic-recovery-19

C.2. Since draft-ietf-quic-recovery-18

C.3. Since draft-ietf-quic-recovery-17

C.4. Since draft-ietf-quic-recovery-16

C.5. Since draft-ietf-quic-recovery-14


C.7. Since draft-ietf-quic-recovery-12

C.8. Since draft-ietf-quic-recovery-11

C.9. Since draft-ietf-quic-recovery-10

C.10. Since draft-ietf-quic-recovery-09

C.11. Since draft-ietf-quic-recovery-08

C.12. Since draft-ietf-quic-recovery-07

C.13. Since draft-ietf-quic-recovery-06
1.  Introduction

QUIC is a new multiplexed and secure transport atop UDP. QUIC builds on decades of transport and security experience, and implements mechanisms that make it attractive as a modern general-purpose transport. The QUIC protocol is described in [QUIC-TRANSPORT].

QUIC implements the spirit of existing TCP loss recovery mechanisms, described in RFCs, various Internet-drafts, and also those prevalent in the Linux TCP implementation. This document describes QUIC congestion control and loss recovery, and where applicable, attributes the TCP equivalent in RFCs, Internet-drafts, academic papers, and/or TCP implementations.

2.  Conventions and Definitions

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.

Definitions of terms that are used in this document:

ACK-only: Any packet containing only one or more ACK frame(s).

In-flight: Packets are considered in-flight when they have been sent and neither acknowledged nor declared lost, and they are not ACK-only.

Ack-eliciting Frames: All frames besides ACK or PADDING are considered ack-eliciting.

Ack-eliciting Packets: Packets that contain ack-eliciting frames elicit an ACK from the receiver within the maximum ack delay and are called ack-eliciting packets.
Crypto Packets: Packets containing CRYPTO data sent in Initial or Handshake packets.

Out-of-order Packets: Packets that do not increase the largest received packet number for its packet number space by exactly one. Packets arrive out of order when earlier packets are lost or delayed.

3. Design of the QUIC Transmission Machinery

All transmissions in QUIC are sent with a packet-level header, which indicates the encryption level and includes a packet sequence number (referred to below as a packet number). The encryption level indicates the packet number space, as described in [QUIC-TRANSPORT]. Packet numbers never repeat within a packet number space for the lifetime of a connection. Packet numbers monotonically increase within a space, preventing ambiguity.

This design obviates the need for disambiguating between transmissions and retransmissions and eliminates significant complexity from QUIC’s interpretation of TCP loss detection mechanisms.

QUIC packets can contain multiple frames of different types. The recovery mechanisms ensure that data and frames that need reliable delivery are acknowledged or declared lost and sent in new packets as necessary. The types of frames contained in a packet affect recovery and congestion control logic:

- All packets are acknowledged, though packets that contain no ack-eliciting frames are only acknowledged along with ack-eliciting packets.
- Long header packets that contain CRYPTO frames are critical to the performance of the QUIC handshake and use shorter timers for acknowledgement and retransmission.
- Packets that contain only ACK frames do not count toward congestion control limits and are not considered in-flight.
- PADDING frames cause packets to contribute toward bytes in flight without directly causing an acknowledgment to be sent.

3.1. Relevant Differences Between QUIC and TCP

Readers familiar with TCP’s loss detection and congestion control will find algorithms here that parallel well-known TCP ones. Protocol differences between QUIC and TCP however contribute to
algorithmic differences. We briefly describe these protocol differences below.

3.1.1. Separate Packet Number Spaces

QUIC uses separate packet number spaces for each encryption level, except 0-RTT and all generations of 1-RTT keys use the same packet number space. Separate packet number spaces ensures acknowledgement of packets sent with one level of encryption will not cause spurious retransmission of packets sent with a different encryption level. Congestion control and round-trip time (RTT) measurement are unified across packet number spaces.

3.1.2. Monotonically Increasing Packet Numbers

TCP conflates transmission order at the sender with delivery order at the receiver, which results in retransmissions of the same data carrying the same sequence number, and consequently leads to "retransmission ambiguity". QUIC separates the two: QUIC uses a packet number to indicate transmission order, and any application data is sent in one or more streams, with delivery order determined by stream offsets encoded within STREAM frames.

QUIC’s packet number is strictly increasing within a packet number space, and directly encodes transmission order. A higher packet number signifies that the packet was sent later, and a lower packet number signifies that the packet was sent earlier. When a packet containing ack-eliciting frames is detected lost, QUIC rebundles necessary frames in a new packet with a new packet number, removing ambiguity about which packet is acknowledged when an ACK is received. Consequently, more accurate RTT measurements can be made, spurious retransmissions are trivially detected, and mechanisms such as Fast Retransmit can be applied universally, based only on packet number.

This design point significantly simplifies loss detection mechanisms for QUIC. Most TCP mechanisms implicitly attempt to infer transmission ordering based on TCP sequence numbers - a non-trivial task, especially when TCP timestamps are not available.

3.1.3. No Reneging

QUIC ACKs contain information that is similar to TCP SACK, but QUIC does not allow any acked packet to be reneged, greatly simplifying implementations on both sides and reducing memory pressure on the sender.
3.1.4. More ACK Ranges

QUIC supports many ACK ranges, opposed to TCP’s 3 SACK ranges. In high loss environments, this speeds recovery, reduces spurious retransmits, and ensures forward progress without relying on timeouts.

3.1.5. Explicit Correction For Delayed Acknowledgements

QUIC endpoints measure the delay incurred between when a packet is received and when the corresponding acknowledgment is sent, allowing a peer to maintain a more accurate round-trip time estimate (see Section 4.4).

4. Generating Acknowledgements

An acknowledgement SHOULD be sent immediately upon receipt of a second ack-eliciting packet. QUIC recovery algorithms do not assume the peer sends an ACK immediately when receiving a second ack-eliciting packet.

In order to accelerate loss recovery and reduce timeouts, the receiver SHOULD send an immediate ACK after it receives an out-of-order packet. It could send immediate ACKs for in-order packets for a period of time that SHOULD NOT exceed 1/8 RTT unless more out-of-order packets arrive. If every packet arrives out-of-order, then an immediate ACK SHOULD be sent for every received packet.

Similarly, packets marked with the ECN Congestion Experienced (CE) codepoint in the IP header SHOULD be acknowledged immediately, to reduce the peer’s response time to congestion events.

As an optimization, a receiver MAY process multiple packets before sending any ACK frames in response. In this case the receiver can determine whether an immediate or delayed acknowledgement should be generated after processing incoming packets.

4.1. Crypto Handshake Data

In order to quickly complete the handshake and avoid spurious retransmissions due to crypto retransmission timeouts, crypto packets SHOULD use a very short ack delay, such as the local timer granularity. ACK frames SHOULD be sent immediately when the crypto stack indicates all data for that packet number space has been received.
4.2. ACK Ranges

When an ACK frame is sent, one or more ranges of acknowledged packets are included. Including older packets reduces the chance of spurious retransmits caused by losing previously sent ACK frames, at the cost of larger ACK frames.

ACK frames SHOULD always acknowledge the most recently received packets, and the more out-of-order the packets are, the more important it is to send an updated ACK frame quickly, to prevent the peer from declaring a packet as lost and spuriously retransmitting the frames it contains.

Below is one recommended approach for determining what packets to include in an ACK frame.

4.3. Receiver Tracking of ACK Frames

When a packet containing an ACK frame is sent, the largest acknowledged in that frame may be saved. When a packet containing an ACK frame is acknowledged, the receiver can stop acknowledging packets less than or equal to the largest acknowledged in the sent ACK frame.

In cases without ACK frame loss, this algorithm allows for a minimum of 1 RTT of reordering. In cases with ACK frame loss and reordering, this approach does not guarantee that every acknowledgement is seen by the sender before it is no longer included in the ACK frame. Packets could be received out of order and all subsequent ACK frames containing them could be lost. In this case, the loss recovery algorithm may cause spurious retransmits, but the sender will continue making forward progress.

4.4. Measuring and Reporting Host Delay

An endpoint measures the delay incurred between when a packet is received and when the corresponding acknowledgment is sent. The endpoint encodes this host delay for the largest acknowledged packet in the Ack Delay field of an ACK frame (see Section 19.3 of [QUIC-TRANSPORT]). This allows the receiver of the ACK to adjust for any host delays, which is important for delayed acknowledgements, when estimating the path RTT. In certain deployments, a packet might be held in the OS kernel or elsewhere on the host before being processed by the QUIC stack. Where possible, an endpoint MAY include these delays when populating the Ack Delay field in an ACK frame.

An endpoint MUST NOT excessively delay acknowledgements of ack-eliciting packets. The maximum ack delay is communicated in the
max_ack_delay transport parameter, see Section 18.1 of [QUIC-TRANSPORT]. max_ack_delay implies an explicit contract: an endpoint promises to never delay acknowledgments of an ack-eliciting packet by more than the indicated value. If it does, any excess accrues to the RTT estimate and could result in spurious retransmissions from the peer.

5. Estimating the Round-Trip Time

At a high level, an endpoint measures the time from when a packet was sent to when it is acknowledged as a round-trip time (RTT) sample. The endpoint uses RTT samples and peer-reported host delays (Section 4.4) to generate a statistical description of the connection’s RTT. An endpoint computes the following three values: the minimum value observed over the lifetime of the connection (min_rtt), an exponentially-weighted moving average (smoothed_rtt), and the variance in the observed RTT samples (rttvar).

5.1. Generating RTT samples

An endpoint generates an RTT sample on receiving an ACK frame that meets the following two conditions:

- the largest acknowledged packet number is newly acknowledged, and
- at least one of the newly acknowledged packets was ack-eliciting.

The RTT sample, latest_rtt, is generated as the time elapsed since the largest acknowledged packet was sent:

\[ \text{latest_rtt} = \text{ack_time} - \text{send_time_of_largest_acked} \]

An RTT sample is generated using only the largest acknowledged packet in the received ACK frame. This is because a peer reports host delays for only the largest acknowledged packet in an ACK frame. While the reported host delay is not used by the RTT sample measurement, it is used to adjust the RTT sample in subsequent computations of smoothed_rtt and rttvar Section 5.3.

To avoid generating multiple RTT samples using the same packet, an ACK frame SHOULD NOT be used to update RTT estimates if it does not newly acknowledge the largest acknowledged packet.

An RTT sample MUST NOT be generated on receiving an ACK frame that does not newly acknowledge at least one ack-eliciting packet. A peer does not send an ACK frame on receiving only non-ack-eliciting packets, so an ACK frame that is subsequently sent can include an
arbitrarily large Ack Delay field. Ignoring such ACK frames avoids complications in subsequent smoothed_rtt and rttvar computations.

A sender might generate multiple RTT samples per RTT when multiple ACK frames are received within an RTT. As suggested in [RFC6298], doing so might result in inadequate history in smoothed_rtt and rttvar. Ensuring that RTT estimates retain sufficient history is an open research question.

5.2. Estimating min_rtt

min_rtt is the minimum RTT observed over the lifetime of the connection. min_rtt is set to the latest_rtt on the first sample in a connection, and to the lesser of min_rtt and latest_rtt on subsequent samples.

An endpoint uses only locally observed times in computing the min_rtt and does not adjust for host delays reported by the peer (Section 4.4). Doing so allows the endpoint to set a lower bound for the smoothed_rtt based entirely on what it observes (see Section 5.3), and limits potential underestimation due to erroneously-reported delays by the peer.

5.3. Estimating smoothed_rtt and rttvar

smoothed_rtt is an exponentially-weighted moving average of an endpoint’s RTT samples, and rttvar is the endpoint’s estimated variance in the RTT samples.

smoothed_rtt uses path latency after adjusting RTT samples for peer-reported host delays (Section 4.4). A peer limits any delay in sending an acknowledgement for an ack-eliciting packet to no greater than the advertised max_ack_delay transport parameter. Consequently, when a peer reports an Ack Delay that is greater than its max_ack_delay, the delay is attributed to reasons out of the peer’s control, such as scheduler latency at the peer or loss of previous ACK frames. Any delays beyond the peer’s max_ack_delay are therefore considered effectively part of path delay and incorporated into the smoothed_rtt estimate.

When adjusting an RTT sample using peer-reported acknowledgement delays, an endpoint:

- MUST use the lesser of the value reported in Ack Delay field of the ACK frame and the peer’s max_ack_delay transport parameter (Section 4.4).
o MUST NOT apply the adjustment if the resulting RTT sample is smaller than the min_rtt. This limits the underestimation that a misreporting peer can cause to the smoothed_rtt.

On the first RTT sample in a connection, the smoothed_rtt is set to the latest_rtt.

smoothed_rtt and rttvar are computed as follows, similar to [RFC6298]. On the first RTT sample in a connection:

\[
\text{smoothed}_\text{rtt} = \text{latest}_\text{rtt} \\
\text{rttvar} = \frac{\text{latest}_\text{rtt}}{2}
\]

On subsequent RTT samples, smoothed_rtt and rttvar evolve as follows:

\[
\begin{align*}
\text{ack}_\text{delay} &= \min(\text{Ack Delay in ACK Frame, max}_\text{ack}_\text{delay}) \\
\text{adjusted}_\text{rtt} &= \text{latest}_\text{rtt} \\
\text{if } (\text{min}_\text{rtt} + \text{ack}_\text{delay} < \text{latest}_\text{rtt}): \\
\text{smoothed}_\text{rtt} &= \frac{7}{8} \times \text{smoothed}_\text{rtt} + \frac{1}{8} \times \text{adjusted}_\text{rtt} \\
\text{rttvar}_\text{sample} &= \text{abs}(\text{smoothed}_\text{rtt} - \text{adjusted}_\text{rtt}) \\
\text{rttvar} &= \frac{3}{4} \times \text{rttvar} + \frac{1}{4} \times \text{rttvar}_\text{sample}
\end{align*}
\]

6. Loss Detection

QUIC senders use both ack information and timeouts to detect lost packets, and this section provides a description of these algorithms.

If a packet is lost, the QUIC transport needs to recover from that loss, such as by retransmitting the data, sending an updated frame, or abandoning the frame. For more information, see Section 13.2 of [QUIC-TRANSPORT].

6.1. Acknowledgement-based Detection

Acknowledgement-based loss detection implements the spirit of TCP’s Fast Retransmit [RFC5681], Early Retransmit [RFC5827], FACK [FACK], SACK loss recovery [RFC6675], and RACK [RACK]. This section provides an overview of how these algorithms are implemented in QUIC.

A packet is declared lost if it meets all the following conditions:

- The packet is unacknowledged, in-flight, and was sent prior to an acknowledged packet.

- Either its packet number is kPacketThreshold smaller than an acknowledged packet (Section 6.1.1), or it was sent long enough in the past (Section 6.1.2).
The acknowledgement indicates that a packet sent later was delivered, while the packet and time thresholds provide some tolerance for packet reordering.

Spuriously declaring packets as lost leads to unnecessary retransmissions and may result in degraded performance due to the actions of the congestion controller upon detecting loss. Implementations that detect spurious retransmissions and increase the reordering threshold in packets or time MAY choose to start with smaller initial reordering thresholds to minimize recovery latency.

6.1.1. Packet Threshold

The RECOMMENDED initial value for the packet reordering threshold (kPacketThreshold) is 3, based on best practices for TCP loss detection [RFC5681] [RFC6675].

Some networks may exhibit higher degrees of reordering, causing a sender to detect spurious losses. Implementers MAY use algorithms developed for TCP, such as TCP-NCR [RFC4653], to improve QUIC’s reordering resilience.

6.1.2. Time Threshold

Once a later packet has been acknowledged, an endpoint SHOULD declare an earlier packet lost if it was sent a threshold amount of time in the past. The time threshold is computed as kTimeThreshold * max(SRTT, latest_RTT). If packets sent prior to the largest acknowledged packet cannot yet be declared lost, then a timer SHOULD be set for the remaining time.

The RECOMMENDED time threshold (kTimeThreshold), expressed as a round-trip time multiplier, is 9/8.

Using max(SRTT, latest_RTT) protects from the two following cases:

- the latest RTT sample is lower than the SRTT, perhaps due to reordering where the acknowledgement encountered a shorter path;
- the latest RTT sample is higher than the SRTT, perhaps due to a sustained increase in the actual RTT, but the smoothed SRTT has not yet caught up.

An endpoint might consistently record RTT samples as 0 in extremely low latency networks, leading to a smoothed_rtt of 0. Consequently, the endpoint could declare all earlier packets as lost immediately upon receiving an acknowledgement for a later packet. That is, the endpoint would not provide any reordering tolerance. To avoid
declaring packets as lost too early, the time threshold MUST be set
to at least \texttt{kGranularity} (defined in Appendix A.2).

Implementations MAY experiment with absolute thresholds, thresholds
from previous connections, adaptive thresholds, or including RTT
variance. Smaller thresholds reduce reordering resilience and
increase spurious retransmissions, and larger thresholds increase
loss detection delay.

6.2. Crypto Retransmission Timeout

Data in CRYPTO frames is critical to QUIC transport and crypto
negotiation, so a more aggressive timeout is used to retransmit it.

The initial crypto retransmission timeout SHOULD be set to twice the
initial RTT.

At the beginning, there are no prior RTT samples within a connection.
Resumed connections over the same network SHOULD use the previous
connection's final smoothed RTT value as the resumed connection's
initial RTT. If no previous RTT is available, or if the network
changes, the initial RTT SHOULD be set to 500ms, resulting in a 1
second initial handshake timeout as recommended in [RFC6298].

When a crypto packet is sent, the sender MUST set a timer for twice
the smoothed RTT. This timer MUST be updated when a new crypto
packet is sent and when an acknowledgement is received which computes
a new RTT sample. Upon timeout, the sender MUST retransmit all
unacknowledged CRYPTO data if possible. The sender MUST NOT declare
in-flight crypto packets as lost when the crypto timer expires.

On each consecutive expiration of the crypto timer without receiving
an acknowledgement for a new packet, the sender MUST double the
crypto retransmission timeout and set a timer for this period.

Until the server has validated the client’s address on the path, the
amount of data it can send is limited, as specified in Section 8.1 of
[QUIC-TRANSPORT]. If not all unacknowledged CRYPTO data can be sent,
then all unacknowledged CRYPTO data sent in Initial packets should be
retransmitted. If no data can be sent, then no alarm should be armed
until data has been received from the client.

Because the server could be blocked until more packets are received,
the client MUST ensure that the crypto retransmission timer is set if
there is unacknowledged crypto data or if the client does not yet
have 1-RTT keys. If the crypto retransmission timer expires before
the client has 1-RTT keys, it is possible that the client may not
have any crypto data to retransmit. However, the client MUST send a
new packet, containing only PING or PADDING frames if necessary, to allow the server to continue sending data. If Handshake keys are available to the client, it MUST send a Handshake packet, and otherwise it MUST send an Initial packet in a UDP datagram of at least 1200 bytes.

The crypto retransmission timer is not set if the time threshold Section 6.1.2 loss detection timer is set. The time threshold loss detection timer is expected to both expire earlier than the crypto retransmission timeout and be less likely to spuriously retransmit data. The Initial and Handshake packet number spaces will typically contain a small number of packets, so losses are less likely to be detected using packet-threshold loss detection.

When the crypto retransmission timer is active, the probe timer (Section 6.3) is not active.

6.2.1. Retry and Version Negotiation

A Retry or Version Negotiation packet causes a client to send another Initial packet, effectively restarting the connection process and resetting congestion control and loss recovery state, including resetting any pending timers. Either packet indicates that the Initial was received but not processed. Neither packet can be treated as an acknowledgment for the Initial.

The client MAY however compute an RTT estimate to the server as the time period from when the first Initial was sent to when a Retry or a Version Negotiation packet is received. The client MAY use this value to seed the RTT estimator for a subsequent connection attempt to the server.

6.2.2. Discarding Keys and Packet State

When packet protection keys are discarded (see Section 4.9 of [QUIC-TLS]), all packets that were sent with those keys can no longer be acknowledged because their acknowledgements cannot be processed anymore. The sender MUST discard all recovery state associated with those packets and MUST remove them from the count of bytes in flight.

Endpoints stop sending and receiving Initial packets once they start exchanging Handshake packets (see Section 17.2.2.1 of [QUIC-TRANSPORT]). At this point, recovery state for all in-flight Initial packets is discarded.

When 0-RTT is rejected, recovery state for all in-flight 0-RTT packets is discarded.
If a server accepts 0-RTT, but does not buffer 0-RTT packets that arrive before Initial packets, early 0-RTT packets will be declared lost, but that is expected to be infrequent.

It is expected that keys are discarded after packets encrypted with them would be acknowledged or declared lost. Initial secrets however might be destroyed sooner, as soon as handshake keys are available (see Section 4.10 of [QUIC-TLS]).

6.3. Probe Timeout

A Probe Timeout (PTO) triggers a probe packet when ack-eliciting data is in flight but an acknowledgement is not received within the expected period of time. A PTO enables a connection to recover from loss of tail packets or acks. The PTO algorithm used in QUIC implements the reliability functions of Tail Loss Probe [TLP] [RACK], RTO [RFC5681] and F-RTO algorithms for TCP [RFC5682], and the timeout computation is based on TCP’s retransmission timeout period [RFC6298].

6.3.1. Computing PTO

When an ack-eliciting packet is transmitted, the sender schedules a timer for the PTO period as follows:

\[ \text{PTO} = \text{smoothed}\_\text{rtt} + \max(4*\text{rttvar}, \text{kGranularity}) + \max\_\text{ack}\_\text{delay} \]

kGranularity, smoothed_rtt, rttvar, and max_ack_delay are defined in Appendix A.2 and Appendix A.3.

The PTO period is the amount of time that a sender ought to wait for an acknowledgement of a sent packet. This time period includes the estimated network roundtrip-time (smoothed_rtt), the variance in the estimate (4*rttvar), and max_ack_delay, to account for the maximum time by which a receiver might delay sending an acknowledgement.

The PTO value MUST be set to at least kGranularity, to avoid the timer expiring immediately.

When a PTO timer expires, the sender probes the network as described in the next section. The PTO period MUST be set to twice its current value. This exponential reduction in the sender’s rate is important because the PTOs might be caused by loss of packets or acknowledgements due to severe congestion.

A sender computes its PTO timer every time an ack-eliciting packet is sent. A sender might choose to optimize this by setting the timer
fewer times if it knows that more ack-eliciting packets will be sent within a short period of time.

6.3.2. Sending Probe Packets

When a PTO timer expires, a sender MUST send at least one ack-eliciting packet as a probe, unless there is no data available to send. An endpoint MAY send up to two ack-eliciting packets, to avoid an expensive consecutive PTO expiration due to a single packet loss.

It is possible that the sender has no new or previously-sent data to send. As an example, consider the following sequence of events: new application data is sent in a STREAM frame, deemed lost, then retransmitted in a new packet, and then the original transmission is acknowledged. In the absence of any new application data, a PTO timer expiration now would find the sender with no new or previously-sent data to send.

When there is no data to send, the sender SHOULD send a PING or other ack-eliciting frame in a single packet, re-arming the PTO timer.

Alternatively, instead of sending an ack-eliciting packet, the sender MAY mark any packets still in flight as lost. Doing so avoids sending an additional packet, but increases the risk that loss is declared too aggressively, resulting in an unnecessary rate reduction by the congestion controller.

Consecutive PTO periods increase exponentially, and as a result, connection recovery latency increases exponentially as packets continue to be dropped in the network. Sending two packets on PTO expiration increases resilience to packet drops, thus reducing the probability of consecutive PTO events.

Probe packets sent on a PTO MUST be ack-eliciting. A probe packet SHOULD carry new data when possible. A probe packet MAY carry retransmitted unacknowledged data when new data is unavailable, when flow control does not permit new data to be sent, or to opportunistically reduce loss recovery delay. Implementations MAY use alternate strategies for determining the content of probe packets, including sending new or retransmitted data based on the application’s priorities.

When the PTO timer expires multiple times and new data cannot be sent, implementations must choose between sending the same payload every time or sending different payloads. Sending the same payload may be simpler and ensures the highest priority frames arrive first. Sending different payloads each time reduces the chances of spurious retransmission.
6.3.3. Loss Detection

Delivery or loss of packets in flight is established when an ACK frame is received that newly acknowledges one or more packets.

A PTO timer expiration event does not indicate packet loss and MUST NOT cause prior unacknowledged packets to be marked as lost. When an acknowledgement is received that newly acknowledges packets, loss detection proceeds as dictated by packet and time threshold mechanisms, see Section 6.1.

6.4. Discussion

The majority of constants were derived from best common practices among widely deployed TCP implementations on the internet. Exceptions follow.

A shorter delayed ack time of 25ms was chosen because longer delayed acks can delay loss recovery and for the small number of connections where less than packet per 25ms is delivered, acking every packet is beneficial to congestion control and loss recovery.

7. Congestion Control

QUIC’s congestion control is based on TCP NewReno [RFC6582]. NewReno is a congestion window based congestion control. QUIC specifies the congestion window in bytes rather than packets due to finer control and the ease of appropriate byte counting [RFC3465].

QUIC hosts MUST NOT send packets if they would increase bytes_in_flight (defined in Appendix B.2) beyond the available congestion window, unless the packet is a probe packet sent after a PTO timer expires, as described in Section 6.3.

Implementations MAY use other congestion control algorithms, such as Cubic [RFC8312], and endpoints MAY use different algorithms from one another. The signals QUIC provides for congestion control are generic and are designed to support different algorithms.

7.1. Explicit Congestion Notification

If a path has been verified to support ECN, QUIC treats a Congestion Experienced codepoint in the IP header as a signal of congestion. This document specifies an endpoint’s response when its peer receives packets with the Congestion Experienced codepoint. As discussed in [RFC8311], endpoints are permitted to experiment with other response functions.
7.2. Slow Start

QUIC begins every connection in slow start and exits slow start upon loss or upon increase in the ECN-CE counter. QUIC re-enters slow start anytime the congestion window is less than ssthresh, which typically only occurs after an PTO. While in slow start, QUIC increases the congestion window by the number of bytes acknowledged when each acknowledgment is processed.

7.3. Congestion Avoidance

Slow start exits to congestion avoidance. Congestion avoidance in NewReno uses an additive increase multiplicative decrease (AIMD) approach that increases the congestion window by one maximum packet size per congestion window acknowledged. When a loss is detected, NewReno halves the congestion window and sets the slow start threshold to the new congestion window.

7.4. Recovery Period

Recovery is a period of time beginning with detection of a lost packet or an increase in the ECN-CE counter. Because QUIC does not retransmit packets, it defines the end of recovery as a packet sent after the start of recovery being acknowledged. This is slightly different from TCP’s definition of recovery, which ends when the lost packet that started recovery is acknowledged.

The recovery period limits congestion window reduction to once per round trip. During recovery, the congestion window remains unchanged irrespective of new losses or increases in the ECN-CE counter.

7.5. Ignoring Loss of Undecryptable Packets

During the handshake, some packet protection keys might not be available when a packet arrives. In particular, Handshake and 0-RTT packets cannot be processed until the Initial packets arrive, and 1-RTT packets cannot be processed until the handshake completes. Endpoints MAY ignore the loss of Handshake, 0-RTT, and 1-RTT packets that might arrive before the peer has packet protection keys to process those packets.

7.6. Probe Timeout

Probe packets MUST NOT be blocked by the congestion controller. A sender MUST however count these packets as being additionally in flight, since these packets add network load without establishing packet loss. Note that sending probe packets might cause the sender’s bytes in flight to exceed the congestion window until an
acknowledgement is received that establishes loss or delivery of packets.

7.7. Persistent Congestion

When an ACK frame is received that establishes loss of all in-flight packets sent over a long enough period of time, the network is considered to be experiencing persistent congestion. Commonly, this can be established by consecutive PTOs, but since the PTO timer is reset when a new ack-eliciting packet is sent, an explicit duration must be used to account for those cases where PTOs do not occur or are substantially delayed. This duration is computed as follows:

\[(\text{smoothed}_\text{rtt} + 4 \times \text{rttvar} + \text{max}_\text{ack}_\text{delay}) \times k_{\text{PersistentCongestionThreshold}}\]

For example, assume:

smoothed\_rtt = 1 rttvar = 0 max\_ack\_delay = 0
kPersistentCongestionThreshold = 3

If an eck-eliciting packet is sent at time = 0, the following scenario would illustrate persistent congestion:

+------------------------+---------+------------------------+
| t=0                    | Send Pkt #1 (App Data) |
|                        |                     |
| t=1                    | Send Pkt #2 (PTO 1)  |
|                        |                     |
| t=3                    | Send Pkt #3 (PTO 2)  |
|                        |                     |
| t=7                    | Send Pkt #4 (PTO 3)  |
|                        |                     |
| t=8                    | Recv ACK of Pkt #4   |
+------------------------+---------+------------------------+

The first three packets are determined to be lost when the ACK of packet 4 is received at t=8. The congestion period is calculated as the time between the oldest and newest lost packets: (3 - 0) = 3. The duration for persistent congestion is equal to: (1 * kPersistentCongestionThreshold) = 3. Because the threshold was reached and because none of the packets between the oldest and the newest packets are acknowledged, the network is considered to have experienced persistent congestion.

When persistent congestion is established, the sender’s congestion window MUST be reduced to the minimum congestion window (kMinimumWindow). This response of collapsing the congestion window
on persistent congestion is functionally similar to a sender’s response on a Retransmission Timeout (RTO) in TCP [RFC5681] after Tail Loss Probes (TLP) [TLP].

7.8. Pacing

This document does not specify a pacer, but it is RECOMMENDED that a sender pace sending of all in-flight packets based on input from the congestion controller. For example, a pacer might distribute the congestion window over the SRTT when used with a window-based controller, and a pacer might use the rate estimate of a rate-based controller.

An implementation should take care to architect its congestion controller to work well with a pacer. For instance, a pacer might wrap the congestion controller and control the availability of the congestion window, or a pacer might pace out packets handed to it by the congestion controller. Timely delivery of ACK frames is important for efficient loss recovery. Packets containing only ACK frames should therefore not be paced, to avoid delaying their delivery to the peer.

As an example of a well-known and publicly available implementation of a flow pacer, implementers are referred to the Fair Queue packet scheduler (fq qdisc) in Linux (3.11 onwards).

7.9. Under-utilizing the Congestion Window

A congestion window that is under-utilized SHOULD NOT be increased in either slow start or congestion avoidance. This can happen due to insufficient application data or flow control credit.

A sender MAY use the pipeACK method described in section 4.3 of [RFC7661] to determine if the congestion window is sufficiently utilized.

A sender that paces packets (see Section 7.8) might delay sending packets and not fully utilize the congestion window due to this delay. A sender should not consider itself application limited if it would have fully utilized the congestion window without pacing delay.

Bursting more than an initial window’s worth of data into the network might cause short-term congestion and losses. Implementations SHOULD either use pacing or reduce their congestion window to limit such bursts.
A sender MAY implement alternate mechanisms to update its congestion window after periods of under-utilization, such as those proposed for TCP in [RFC7661].

8. Security Considerations

8.1. Congestion Signals

Congestion control fundamentally involves the consumption of signals – both loss and ECN codepoints – from unauthenticated entities. On-path attackers can spoof or alter these signals. An attacker can cause endpoints to reduce their sending rate by dropping packets, or alter send rate by changing ECN codepoints.

8.2. Traffic Analysis

Packets that carry only ACK frames can be heuristically identified by observing packet size. Acknowledgement patterns may expose information about link characteristics or application behavior. Endpoints can use PADDING frames or bundle acknowledgments with other frames to reduce leaked information.

8.3. Misreporting ECN Markings

A receiver can misreport ECN markings to alter the congestion response of a sender. Suppressing reports of ECN-CE markings could cause a sender to increase their send rate. This increase could result in congestion and loss.

A sender MAY attempt to detect suppression of reports by marking occasional packets that they send with ECN-CE. If a packet marked with ECN-CE is not reported as having been marked when the packet is acknowledged, the sender SHOULD then disable ECN for that path.

Reporting additional ECN-CE markings will cause a sender to reduce their sending rate, which is similar in effect to advertising reduced connection flow control limits and so no advantage is gained by doing so.

Endpoints choose the congestion controller that they use. Though congestion controllers generally treat reports of ECN-CE markings as equivalent to loss [RFC8311], the exact response for each controller could be different. Failure to correctly respond to information about ECN markings is therefore difficult to detect.
9. IANA Considerations

This document has no IANA actions. Yet.

10. References

10.1. Normative References

[QUIC-TLS]

[QUIC-TRANSPORT]


10.2. Informative References


Appendix A. Loss Recovery Pseudocode

We now describe an example implementation of the loss detection mechanisms described in Section 6.

A.1. Tracking Sent Packets

To correctly implement congestion control, a QUIC sender tracks every ack-eliciting packet until the packet is acknowledged or lost. It is expected that implementations will be able to access this information by packet number and crypto context and store the per-packet fields (Appendix A.1.1) for loss recovery and congestion control.

After a packet is declared lost, it SHOULD be tracked for an amount of time comparable to the maximum expected packet reordering, such as 1 RTT. This allows for detection of spurious retransmissions.

Sent packets are tracked for each packet number space, and ACK processing only applies to a single space.

A.1.1. Sent Packet Fields

- **packet_number**: The packet number of the sent packet.
- **ack_eliciting**: A boolean that indicates whether a packet is ack-eliciting. If true, it is expected that an acknowledgement will be received, though the peer could delay sending the ACK frame containing it by up to the MaxAckDelay.
in_flight: A boolean that indicates whether the packet counts
towards bytes in flight.

is_crypto_packet: A boolean that indicates whether the packet
contains cryptographic handshake messages critical to the
completion of the QUIC handshake. In this version of QUIC, this
includes any packet with the long header that includes a CRYPTO
frame.

sent_bytes: The number of bytes sent in the packet, not including
UDP or IP overhead, but including QUIC framing overhead.

time_sent: The time the packet was sent.

A.2. Constants of interest

Constants used in loss recovery are based on a combination of RFCs,
papers, and common practice. Some may need to be changed or
negotiated in order to better suit a variety of environments.

kPacketThreshold: Maximum reordering in packets before packet
threshold loss detection considers a packet lost. The RECOMMENDED
value is 3.

kTimeThreshold: Maximum reordering in time before time threshold
loss detection considers a packet lost. Specified as an RTT
multiplier. The RECOMMENDED value is 9/8.

kGranularity: Timer granularity. This is a system-dependent value.
However, implementations SHOULD use a value no smaller than 1ms.

kInitialRtt: The RTT used before an RTT sample is taken. The
RECOMMENDED value is 500ms.

kPacketNumberSpace: An enum to enumerate the three packet number
spaces.

enum kPacketNumberSpace {
    Initial,
    Handshake,
    ApplicationData,
}

A.3. Variables of interest

Variables required to implement the congestion control mechanisms are
described in this section.
A.4. Initialization

At the beginning of the connection, initialize the loss detection variables as follows:

- **loss_detection_timer**: Multi-modal timer used for loss detection.
- **crypto_count**: The number of times all unacknowledged CRYPTO data has been retransmitted without receiving an ack.
- **pto_count**: The number of times a PTO has been sent without receiving an ack.
- **time_of_last_sent_ack_eliciting_packet**: The time the most recent ack-eliciting packet was sent.
- **time_of_last_sent_crypto_packet**: The time the most recent crypto packet was sent.
- **largest_acked_packet[kPacketNumberSpace]**: The largest packet number acknowledged in the packet number space so far.
- **latest_rtt**: The most recent RTT measurement made when receiving an ack for a previously unacked packet.
- **smoothed_rtt**: The smoothed RTT of the connection, computed as described in [RFC6298]
- **rttvar**: The RTT variance, computed as described in [RFC6298]
- **min_rtt**: The minimum RTT seen in the connection, ignoring ack delay.
- **max_ack_delay**: The maximum amount of time by which the receiver intends to delay acknowledgments, in milliseconds. The actual ack_delay in a received ACK frame may be larger due to late timers, reordering, or lost ACKs.
- **loss_time[kPacketNumberSpace]**: The time at which the next packet in that packet number space will be considered lost based on exceeding the reordering window in time.
- **sent_packets[kPacketNumberSpace]**: An association of packet numbers in a packet number space to information about them. Described in detail above in Appendix A.1.
loss_detection_timer.reset()
crypto_count = 0
pto_count = 0
latest_rtt = 0
smoothed_rtt = 0
rttvar = 0
min_rtt = 0
time_of_last_sent_ack_eliciting_packet = 0
time_of_last_sent_crypto_packet = 0
for pn_space in [Initial, Handshake, ApplicationData]:
    largest_acked_packet[pn_space] = 0
    loss_time[pn_space] = 0

A.5. On Sending a Packet

After a packet is sent, information about the packet is stored. The
parameters to OnPacketSent are described in detail above in
Appendix A.1.1.

Pseudocode for OnPacketSent follows:

OnPacketSent(packet_number, pn_space, ack_eliciting,
              in_flight, is_crypto_packet, sent_bytes):
    sent_packets[pn_space][packet_number].packet_number =
        packet_number
    sent_packets[pn_space][packet_number].time_sent = now
    sent_packets[pn_space][packet_number].ack_eliciting =
        ack_eliciting
    sent_packets[pn_space][packet_number].in_flight = in_flight
    if (in_flight):
        if (is_crypto_packet):
            time_of_last_sent_crypto_packet = now
        if (ack_eliciting):
            time_of_last_sent_ack_eliciting_packet = now
    OnPacketSentCC(sent_bytes)
    sent_packets[pn_space][packet_number].size = sent_bytes
    SetLossDetectionTimer()

A.6. On Receiving an Acknowledgment

When an ACK frame is received, it may newly acknowledge any number of
packets.

Pseudocode for OnAckReceived and UpdateRtt follow:

OnAckReceived(ack, pn_space):
    largest_acked_packet[pn_space] =
        max(largest_acked_packet[pn_space], ack.largest_acked)
// Nothing to do if there are no newly acked packets.
newly_acked_packets = DetermineNewlyAckedPackets(ack, pn_space)
if (newly_acked_packets.empty()):
    return

// If the largest acknowledged is newly acked and
// at least one ack-eliciting was newly acked, update the RTT.
if (sent_packets[pn_space][ack.largest_acked] &&
    IncludesAckEliciting(newly_acked_packets))
    latest_rtt =
        now - sent_packets[pn_space][ack.largest_acked].time_sent
    UpdateRtt(ack.ack_delay)

// Process ECN information if present.
if (ACK frame contains ECN information):
    ProcessECN(ack)

for acked_packet in newly_acked_packets:
    OnPacketAcked(acked_packet.packet_number, pn_space)

DetectLostPackets(pn_space)

crypto_count = 0
pto_count = 0

SetLossDetectionTimer()

UpdateRtt(ack_delay):
    // First RTT sample.
    if (smoothed_rtt == 0):
        min_rtt = latest_rtt
        smoothed_rtt = latest_rtt
        rttvar = latest_rtt / 2
        return

    // min_rtt ignores ack delay.
    min_rtt = min(min_rtt, latest_rtt)
    // Limit ack_delay by max_ack_delay
    ack_delay = min(ack_delay, max_ack_delay)
    // Adjust for ack delay if plausible.
    adjusted_rtt = latest_rtt
    if (latest_rtt > min_rtt + ack_delay):
        adjusted_rtt = latest_rtt - ack_delay

    rttvar = 3/4 * rttvar + 1/4 * abs(smoothed_rtt - adjusted_rtt)
    smoothed_rtt = 7/8 * smoothed_rtt + 1/8 * adjusted_rtt
A.7. On Packet Acknowledgment

When a packet is acknowledged for the first time, the following
OnPacketAcked function is called. Note that a single ACK frame may
newly acknowledge several packets. OnPacketAcked must be called once
for each of these newly acknowledged packets.

OnPacketAcked takes two parameters: acked_packet, which is the struct
detailed in Appendix A.1.1, and the packet number space that this ACK
frame was sent for.

Pseudocode for OnPacketAcked follows:

```python
OnPacketAcked(acked_packet, pn_space):
    if (acked_packet.in_flight):
        OnPacketAckedCC(acked_packet)
        sent_packets[pn_space].remove(acked_packet.packet_number)
```

A.8. Setting the Loss Detection Timer

QUIC loss detection uses a single timer for all timeout loss
detection. The duration of the timer is based on the timer’s mode,
which is set in the packet and timer events further below. The
function SetLossDetectionTimer defined below shows how the single
timer is set.

This algorithm may result in the timer being set in the past,
particularly if timers wake up late. Timers set in the past SHOULD
fire immediately.

Pseudocode for SetLossDetectionTimer follows:
GetEarliestLossTime():
    time = loss_time[Initial]
    space = Initial
    for pn_space in [Handshake, ApplicationData]:
        if loss_time[pn_space] != 0 &&
           (time == 0 || loss_time[pn_space] < time):
            time = loss_time[pn_space];
            space = pn_space
    return time, space

SetLossDetectionTimer():
    loss_time, _ = GetEarliestLossTime()
    if (loss_time != 0):
        // Time threshold loss detection.
        loss_detection_timer.update(loss_time)
        return

    if (has unacknowledged crypto data
        || endpoint is client without 1-RTT keys):
        // Crypto retransmission timer.
        if (smoothed_rtt == 0):
            timeout = 2 * kInitialRtt
        else:
            timeout = 2 * smoothed_rtt
            timeout = max(timeout, kGranularity)
            timeout = timeout * (2 ^ crypto_count)
        loss_detection_timer.update(  
            time_of_last_sent_crypto_packet + timeout)
        return

    // Don’t arm timer if there are no ack-eliciting packets
    // in flight.
    if (no ack-eliciting packets in flight):
        loss_detection_timer.cancel()
        return

    // Calculate PTO duration
    timeout =  
        smoothed_rtt + max(4 * rttvar, kGranularity) + max_ack_delay
    timeout = timeout * (2 ^ pto_count)
    loss_detection_timer.update(  
        time_of_last_sent_ack_eliciting_packet + timeout)
A.9. On Timeout

When the loss detection timer expires, the timer’s mode determines the action to be performed.

Pseudocode for OnLossDetectionTimeout follows:

```python
OnLossDetectionTimeout():
    loss_time, pn_space = GetEarliestLossTime()
    if (loss_time != 0):
        // Time threshold loss Detection
        DetectLostPackets(pn_space)
        // Retransmit crypto data if no packets were lost
        // and there is crypto data to retransmit.
        else if (has unacknowledged crypto data):
            // Crypto retransmission timeout.
            RetransmitUnackedCryptoData()
            crypto_count++
        else if (endpoint is client without 1-RTT keys):
            // Client sends an anti-deadlock packet: Initial is padded
            // to earn more anti-amplification credit,
            // a Handshake packet proves address ownership.
            if (has Handshake keys):
                SendOneHandshakePacket()
            else:
                SendOnePaddedInitialPacket()
                crypto_count++
        else:
            // PTO. Send new data if available, else retransmit old data.
            // If neither is available, send a single PING frame.
            SendOneOrTwoPackets()
            pto_count++
    SetLossDetectionTimer()
```

A.10. Detecting Lost Packets

DetectLostPackets is called every time an ACK is received and operates on the sent_packets for that packet number space.

Pseudocode for DetectLostPackets follows:
DetectLostPackets(pn_space):
  loss_time[pn_space] = 0
  lost_packets = {}
  loss_delay = kTimeThreshold * max(latest_rtt, smoothed_rtt)

  // Minimum time of kGranularity before packets are deemed lost.
  loss_delay = max(loss_delay, kGranularity)

  // Packets sent before this time are deemed lost.
  lost_send_time = now() - loss_delay

  // Packets with packet numbers before this are deemed lost.
  lost_pn = largest_acked_packet[pn_space] - kPacketThreshold

  foreach unacked in sent_packets[pn_space]:
    if (unacked.packet_number > largest_acked_packet[pn_space]):
      continue

    // Mark packet as lost, or set time when it should be marked.
    if (unacked.time_sent <= lost_send_time ||
        unacked.packet_number <= lost_pn):
      sent_packets[pn_space].remove(unacked.packet_number)
      if (unacked.in_flight):
        lost_packets.insert(unacked)
    else:
      if (loss_time[pn_space] == 0):
        loss_time[pn_space] = unacked.time_sent + loss_delay
      else:
        loss_time[pn_space] = min(loss_time[pn_space],
                                   unacked.time_sent + loss_delay)

  // Inform the congestion controller of lost packets and
  // let it decide whether to retransmit immediately.
  if (!lost_packets.empty()):
    OnPacketsLost(lost_packets)

Appendix B. Congestion Control Pseudocode

We now describe an example implementation of the congestion controller described in Section 7.

B.1. Constants of interest

Constants used in congestion control are based on a combination of RFCs, papers, and common practice. Some may need to be changed or negotiated in order to better suit a variety of environments.
kMaxDatagramSize: The sender's maximum payload size. Does not include UDP or IP overhead. The max packet size is used for calculating initial and minimum congestion windows. The RECOMMENDED value is 1200 bytes.

kInitialWindow: Default limit on the initial amount of data in flight, in bytes. Taken from [RFC6928], but increased slightly to account for the smaller 8 byte overhead of UDP vs 20 bytes for TCP. The RECOMMENDED value is the minimum of 10 * kMaxDatagramSize and max(2 * kMaxDatagramSize, 14720)).

kMinimumWindow: Minimum congestion window in bytes. The RECOMMENDED value is 2 * kMaxDatagramSize.

kLossReductionFactor: Reduction in congestion window when a new loss event is detected. The RECOMMENDED value is 0.5.

kPersistentCongestionThreshold: Period of time for persistent congestion to be established, specified as a PTO multiplier. The rationale for this threshold is to enable a sender to use initial PTOs for aggressive probing, as TCP does with Tail Loss Probe (TLP) [TLP] [RACK], before establishing persistent congestion, as TCP does with a Retransmission Timeout (RTO) [RFC5681]. The recommended value for kPersistentCongestionThreshold is 3, which is approximately equivalent to having two TLPs before an RTO in TCP.

B.2. Variables of interest

Variables required to implement the congestion control mechanisms are described in this section.

ecn_ce_counter: The highest value reported for the ECN-CE counter by the peer in an ACK frame. This variable is used to detect increases in the reported ECN-CE counter.

bytes_in_flight: The sum of the size in bytes of all sent packets that contain at least one ack-eliciting or PADDING frame, and have not been acked or declared lost. The size does not include IP or UDP overhead, but does include the QUIC header and AEAD overhead. Packets only containing ACK frames do not count towards bytes_in_flight to ensure congestion control does not impede congestion feedback.

congestion_window: Maximum number of bytes-in-flight that may be sent.
congestion_recovery_start_time: The time when QUIC first detects congestion due to loss or ECN, causing it to enter congestion recovery. When a packet sent after this time is acknowledged, QUIC exits congestion recovery.

ssthresh: Slow start threshold in bytes. When the congestion window is below ssthresh, the mode is slow start and the window grows by the number of bytes acknowledged.

B.3. Initialization

At the beginning of the connection, initialize the congestion control variables as follows:

congestion_window = kInitialWindow
bytes_in_flight = 0
congestion_recovery_start_time = 0
ssthresh = infinite
ecn_ce_counter = 0

B.4. On Packet Sent

Whenever a packet is sent, and it contains non-ACK frames, the packet increases bytes_in_flight.

OnPacketSentCC(bytes_sent):
    bytes_in_flight += bytes_sent

B.5. On Packet Acknowledgement

Invoked from loss detection's OnPacketAcked and is supplied with the acked_packet from sent_packets.
InCongestionRecovery(sent_time):
    return sent_time <= congestion_recovery_start_time

OnPacketAckedCC(acked_packet):
    // Remove from bytes_in_flight.
    bytes_in_flight -= acked_packet.size
    if (InCongestionRecovery(acked_packet.time_sent)):
        // Do not increase congestion window in recovery period.
        return
    if (IsAppLimited()):
        // Do not increase congestion_window if application
        // limited.
        return
    if (congestion_window < ssthresh):
        // Slow start.
        congestion_window += acked_packet.size
    else:
        // Congestion avoidance.
        congestion_window += kMaxDatagramSize * acked_packet.size
        / congestion_window

B.6. On New Congestion Event

Invoked from ProcessECN and OnPacketsLost when a new congestion event
is detected. May start a new recovery period and reduces the
congestion window.

CongestionEvent(sent_time):
    // Start a new congestion event if packet was sent after the
    // start of the previous congestion recovery period.
    if (!InCongestionRecovery(sent_time)):
        congestion_recovery_start_time = Now()
        congestion_window *= kLossReductionFactor
        congestion_window = max(congestion_window, kMinimumWindow)
        ssthresh = congestion_window

B.7. Process ECN Information

Invoked when an ACK frame with an ECN section is received from the
peer.

ProcessECN(ack):
    // If the ECN-CE counter reported by the peer has increased,
    // this could be a new congestion event.
    if (ack.ce_counter > ecn_ce_counter):
        ecn_ce_counter = ack.ce_counter
        CongestionEvent(sent_packets[ack.largest_acked].time_sent)
B.8. On Packets Lost

Invoked from DetectLostPackets when packets are deemed lost.

```c
InPersistentCongestion(largest_lost_packet):
    pto = smoothed_rtt + max(4 * rttvar, kGranularity) +
        max_ack_delay
    congestion_period = pto * kPersistentCongestionThreshold
// Determine if all packets in the window before the
// newest lost packet, including the edges, are marked
// lost
    return IsWindowLost(largest_lost_packet, congestion_period)
```

```c
OnPacketsLost(lost_packets):
    // Remove lost packets from bytes_in_flight.
    for (lost_packet : lost_packets):
        bytes_in_flight -= lost_packet.size
    largest_lost_packet = lost_packets.last()
    CongestionEvent(largest_lost_packet.time_sent)

    // Collapse congestion window if persistent congestion
    if (InPersistentCongestion(largest_lost_packet)):
        congestion_window = kMinimumWindow
```

Appendix C. Change Log

*RFC Editor’s Note:* Please remove this section prior to publication of a final version of this document.

Issue and pull request numbers are listed with a leading octothorp.

C.1. Since draft-ietf-quic-recovery-19

- Send a PING if the PTO timer fires and there’s nothing to send (#2624)
- Set loss delay to at least kGranularity (#2617)
- Merge application limited and sending after idle sections. Always limit burst size instead of requiring resetting CWND to initial CWND after idle (#2605)
- Rewrite RTT estimation, allow RTT samples where a newly acked packet is ack-eliciting but the largest_acked is not (#2592)
- Don’t arm the handshake timer if there is no handshake data (#2590)
C.2. Since draft-ietf-quic-recovery-18

- Change IW byte limit to 14720 from 14600 (#2494)
- Update PTO calculation to match RFC6298 (#2480, #2489, #2490)
- Improve loss detection’s description of multiple packet number spaces and pseudocode (#2485, #2451, #2417)
- Declare persistent congestion even if non-probe packets are sent and don’t make persistent congestion more aggressive than RTO verified was (#2365, #2244)
- Move pseudocode to the appendices (#2408)
- What to send on multiple PTOs (#2380)

C.3. Since draft-ietf-quic-recovery-17

- After Probe Timeout discard in-flight packets or send another (#2212, #1965)
- Endpoints discard initial keys as soon as handshake keys are available (#1951, #2045)
- 0-RTT state is discarded when 0-RTT is rejected (#2300)
- Loss detection timer is cancelled when ack-eliciting frames are in flight (#2117, #2093)
- Packets are declared lost if they are in flight (#2104)
- After becoming idle, either pace packets or reset the congestion controller (#2138, 2187)
- Process ECN counts before marking packets lost (#2142)
- Mark packets lost before resetting crypto_count and pto_count (#2208, #2209)
- Congestion and loss recovery state are discarded when keys are discarded (#2327)
C.4. Since draft-ietf-quic-recovery-16

- Unify TLP and RTO into a single PTO; eliminate min RTO, min TLP and min crypto timeouts; eliminate timeout validation (#2114, #2166, #2168, #1017)
- Redefine how congestion avoidance in terms of when the period starts (#1928, #1930)
- Document what needs to be tracked for packets that are in flight (#765, #1724, #1939)
- Integrate both time and packet thresholds into loss detection (#1969, #1212, #934, #1974)
- Reduce congestion window after idle, unless pacing is used (#2007, #2023)
- Disable RTT calculation for packets that don’t elicit acknowledgment (#2060, #2078)
- Limit ack_delay by max_ack_delay (#2060, #2099)
- Initial keys are discarded once Handshake are available (#1951, #2045)
- Reorder ECN and loss detection in pseudocode (#2142)
- Only cancel loss detection timer if ack-eliciting packets are in flight (#2093, #2117)

C.5. Since draft-ietf-quic-recovery-14

- Used max_ack_delay from transport params (#1796, #1782)
- Merge ACK and ACK_ECN (#1783)


- Corrected the lack of ssthresh reduction in CongestionEvent pseudocode (#1598)
- Considerations for ECN spoofing (#1426, #1626)
- Clarifications for PADDING and congestion control (#837, #838, #1517, #1531, #1540)
- Reduce early retransmission timer to RTT/8 (#945, #1581)
Packets are declared lost after an RTO is verified (#935, #1582)

C.7. Since draft-ietf-quic-recovery-12

- Changes to manage separate packet number spaces and encryption levels (#1190, #1242, #1413, #1450)
- Added ECN feedback mechanisms and handling; new ACK_ECN frame (#804, #805, #1372)

C.8. Since draft-ietf-quic-recovery-11

No significant changes.

C.9. Since draft-ietf-quic-recovery-10

- Improved text on ack generation (#1139, #1159)
- Make references to TCP recovery mechanisms informational (#1195)
- Define time_of_last_sent_handshake_packet (#1171)
- Added signal from TLS the data it includes needs to be sent in a Retry packet (#1061, #1199)
- Minimum RTT (min_rtt) is initialized with an infinite value (#1169)

C.10. Since draft-ietf-quic-recovery-09

No significant changes.

C.11. Since draft-ietf-quic-recovery-08

- Clarified pacing and RTO (#967, #977)

C.12. Since draft-ietf-quic-recovery-07

- Include Ack Delay in RTO(and TLP) computations (#981)
- Ack Delay in SRTT computation (#961)
- Default RTT and Slow Start (#590)
- Many editorial fixes.
C.13. Since draft-ietf-quic-recovery-06
    No significant changes.
C.14. Since draft-ietf-quic-recovery-05
    o Add more congestion control text (#776)
C.15. Since draft-ietf-quic-recovery-04
    No significant changes.
C.16. Since draft-ietf-quic-recovery-03
    No significant changes.
C.17. Since draft-ietf-quic-recovery-02
    o Integrate F-RTO (#544, #409)
    o Add congestion control (#545, #395)
    o Require connection abort if a skipped packet was acknowledged (#415)
    o Simplify RTO calculations (#142, #417)
C.18. Since draft-ietf-quic-recovery-01
    o Overview added to loss detection
    o Changes initial default RTT to 100ms
    o Added time-based loss detection and fixes early retransmit
    o Clarified loss recovery for handshake packets
    o Fixed references and made TCP references informative
C.19. Since draft-ietf-quic-recovery-00
    o Improved description of constants and ACK behavior
C.20. Since draft-iyengar-quic-loss-recovery-01
    o Adopted as base for draft-ietf-quic-recovery
    o Updated authors/editors list

Iyengar & Swett Expires October 25, 2019 [Page 40]
Acknowledgments

Authors’ Addresses

Jana Iyengar (editor)
Fastly

Email: jri.ietf@gmail.com

Ian Swett (editor)
Google

Email: ianswett@google.com
Abstract

Explicit Congestion Notification (ECN) is a mechanism where network nodes can mark IP packets instead of dropping them to indicate incipient congestion to the end-points. Receivers with an ECN-capable transport protocol feed back this information to the sender. ECN is specified for TCP in such a way that only one feedback signal can be transmitted per Round-Trip Time (RTT). Recent new TCP mechanisms like Congestion Exposure (ConEx), Data Center TCP (DCTCP) or Low Latency Low Loss Scalable Throughput (L4S) need more accurate ECN feedback information whenever more than one marking is received in one RTT. This document specifies an experimental scheme to provide more than one feedback signal per RTT in the TCP header. Given TCP header space is scarce, it allocates a reserved header bit, that was previously used for the ECN-Nonce which has now been declared historic. It also overloads the two existing ECN flags in the TCP header. Supplementary feedback information can optionally be provided in a new TCP option, which is never used on the TCP SYN.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 12, 2019.

Briscoe, et al. Expires September 12, 2019
Table of Contents

1.  Introduction .................................................. 3
   1.1.  Document Roadmap ........................................ 4
   1.2.  Goals ..................................................... 5
   1.3.  Experiment Goals ......................................... 5
   1.4.  Terminology ............................................... 6
   1.5.  Recap of Existing ECN feedback in IP/TCP ................. 7
2.  AccECN Protocol Overview and Rationale ............................ 8
   2.1.  Capability Negotiation ................................... 9
   2.2.  Feedback Mechanism ....................................... 9
   2.3.  Delayed ACKs and Resilience Against ACK Loss .............. 10
   2.4.  Feedback Metrics ......................................... 11
   2.5.  Generic (Dumb) Reflector ................................ 11
3.  AccECN Protocol Specification ...................................... 12
   3.1.  Negotiating to use AccECN ................................. 12
         3.1.1.  Negotiation during the TCP handshake ............... 12
         3.1.2.  Forward Compatibility ............................... 14
         3.1.3.  Retransmission of the SYN .......................... 15
   3.2.  AccECN Feedback ........................................... 15
         3.2.1.  Initialization of Feedback Counters at the Data
                 Sender .............................................. 16
         3.2.2.  The ACE Field ....................................... 16
         3.2.3.  Testing for Zeroing of the ACE Field ............... 18
         3.2.4.  Testing for Mangling of the IP/ECN Field .......... 19
         3.2.5.  Safety against Ambiguity of the ACE Field .......... 20
         3.2.6.  The AccECN Option .................................. 20
         3.2.7.  Path Traversal of the AccECN Option ............... 22
         3.2.8.  Usage of the AccECN TCP Option .................... 25
   3.3.  Requirements for TCP Proxies, Offload Engines and other
         Middleboxes on AccECN Compliance .......................... 27
4.  Interaction with Other TCP Variants .................................. 28
   4.1.  Compatibility with SYN Cookies ............................ 28
1. Introduction

Explicit Congestion Notification (ECN) [RFC3168] is a mechanism where network nodes can mark IP packets instead of dropping them to indicate incipient congestion to the end-points. Receivers with an ECN-capable transport protocol feedback this information to the sender. ECN is specified for TCP in such a way that only one feedback signal can be transmitted per Round-Trip Time (RTT). Recently, proposed mechanisms like Congestion Exposure (ConEx [RFC7713]), DCTCP [RFC8257] or L4S [I-D.ietf-tsvwg-l4s-arch] need to know when more than one marking is received in one RTT which is information that cannot be provided by the feedback scheme as specified in [RFC3168]. This document specifies an alternative feedback scheme that provides more accurate information and could be used by these new TCP extensions. A fuller treatment of the motivation for this specification is given in the associated requirements document [RFC7560].

This document specifies an experimental scheme for ECN feedback in the TCP header to provide more than one feedback signal per RTT. It will be called the more accurate ECN feedback scheme, or AccECN for short. If AccECN progresses from experimental to the standards...
track, it is intended to be a complete replacement for classic TCP/ECN feedback, not a fork in the design of TCP. AccECN feedback complements TCP’s loss feedback and it supplements classic TCP/ECN feedback, so its applicability is intended to include all public and private IP networks (and even any non-IP networks over which TCP is used today), whether or not any nodes on the path support ECN of whatever flavour.

Until the AccECN experiment succeeds, [RFC3168] will remain as the only standards track specification for adding ECN to TCP. To avoid confusion, in this document we use the term ‘classic ECN’ for the pre-existing ECN specification [RFC3168].

AccECN feedback overloads the two existing ECN flags and allocates the currently reserved flag (previously called NS) in the TCP header, to be used as one field indicating the number of congestion experienced marked packets. Given the new definitions of these three bits, both ends have to support the new wire protocol before it can be used. Therefore during the TCP handshake the two ends use these three bits in the TCP header to negotiate the most advanced feedback protocol that they can both support, in a way that is backward compatible with [RFC3168].

AccECN is solely an (experimental) change to the TCP wire protocol; it only specifies the negotiation and signaling of more accurate ECN feedback from a TCP Data Receiver to a Data Sender. It is completely independent of how TCP might respond to congestion feedback, which is out of scope. For that we refer to [RFC3168] or any RFC that specifies a different response to TCP ECN feedback, for example: [RFC8257]; or the ECN experiments referred to in [RFC8311], namely: a TCP-based Low Latency Low Loss Scalable (L4S) congestion control [I-D.ietf-tsvwg-l4s-arch]; ECN-capable TCP control packets [I-D.ietf-tcpm-generalized-ecn], or Alternative Backoff with ECN (ABE) [RFC8511].

It is recommended that the AccECN protocol is implemented alongside the experimental ECN++ protocol [I-D.ietf-tcpm-generalized-ecn]. Therefore, this specification does not discuss implementing AccECN alongside [RFC5562], which was an earlier experimental protocol with narrower scope than ECN++.

1.1. Document Roadmap

The following introductory sections outline the goals of AccECN (Section 1.2) and the goal of experiments with ECN (Section 1.3) so that it is clear what success would look like. Then terminology is defined (Section 1.4) and a recap of existing prerequisite technology is given (Section 1.5).
Section 2 gives an informative overview of the AccECN protocol. Then Section 3 gives the normative protocol specification. Section 4 assesses the interaction of AccECN with commonly used variants of TCP, whether standardised or not. Section 5 summarises the features and properties of AccECN.

Section 6 summarises the protocol fields and numbers that IANA will need to assign and Section 7 points to the aspects of the protocol that will be of interest to the security community.

Appendix A gives pseudocode examples for the various algorithms that AccECN uses.

1.2. Goals

[RFC7560] enumerates requirements that a candidate feedback scheme will need to satisfy, under the headings: resilience, timeliness, integrity, accuracy (including ordering and lack of bias), complexity, overhead and compatibility (both backward and forward). It recognises that a perfect scheme that fully satisfies all the requirements is unlikely and trade-offs between requirements are likely. Section 5 presents the properties of AccECN against these requirements and discusses the trade-offs made.

The requirements document recognises that a protocol as ubiquitous as TCP needs to be able to serve as-yet-unspecified requirements. Therefore an AccECN receiver aims to act as a generic (dumb) reflector of congestion information so that in future new sender behaviours can be deployed unilaterally.

1.3. Experiment Goals

TCP is critical to the robust functioning of the Internet, therefore any proposed modifications to TCP need to be thoroughly tested. The present specification describes an experimental protocol that adds more accurate ECN feedback to the TCP protocol. The intention is to specify the protocol sufficiently so that more than one implementation can be built in order to test its function, robustness and interoperability (with itself and with previous version of ECN and TCP).

The experimental protocol will be considered successful if testing confirms that the proposed mechanism can be deployed at large scale. Testing will mostly focus on fall-back strategies in case of middlebox interference. Current recommended strategies are specified in Sections 3.1.3, 3.2.3, 3.2.4 and 3.2.7. The effectiveness of these strategies depends on the actual deployment situation of middleboxes. Therefore experimental verification to confirm large-
scale path traversal in the Internet is needed before finalizing this specification on the Standards Track.

Another experimentation focus is the implementation feasibility of change-triggered ACKs as described in section 3.2.8. While on average this should not lead to a higher ACK rate, it changes the ACK pattern which can particularly have an impact on hardware offload. It is currently specified as a hard requirement, because the sender can exploit the predictability of the receiver’s behaviour. However, further experimentation is needed to advise if will have to become just preferred behavior.

1.4. Terminology

AccECN: The more accurate ECN feedback scheme will be called AccECN for short.

Classic ECN: the ECN protocol specified in [RFC3168].

Classic ECN feedback: the feedback aspect of the ECN protocol specified in [RFC3168], including generation, encoding, transmission and decoding of feedback, but not the Data Sender’s subsequent response to that feedback.

ACK: A TCP acknowledgement, with or without a data payload.

Pure ACK: A TCP acknowledgement without a data payload.

TCP client: The TCP stack that originates a connection.

TCP server: The TCP stack that responds to a connection request.

Data Receiver: The endpoint of a TCP half-connection that receives data and sends AccECN feedback.

Data Sender: The endpoint of a TCP half-connection that sends data and receives AccECN feedback.

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.
1.5. Recap of Existing ECN feedback in IP/TCP

ECN [RFC3168] uses two bits in the IP header. Once ECN has been negotiated with the receiver at the transport layer, an ECN sender can set two possible codepoints (ECT(0) or ECT(1)) in the IP header to indicate an ECN-capable transport (ECT). If both ECN bits are zero, the packet is considered to have been sent by a Not-ECN-capable Transport (Not-ECT). When a network node experiences congestion, it will occasionally either drop or mark a packet, with the choice depending on the packet’s ECN codepoint. If the codepoint is Not-ECT, only drop is appropriate. If the codepoint is ECT(0) or ECT(1), the node can mark the packet by setting both ECN bits, which is termed ‘Congestion Experienced’ (CE), or loosely a ‘congestion mark’. Table 1 summarises these codepoints.

<table>
<thead>
<tr>
<th>IP-ECN codepoint (binary)</th>
<th>Codepoint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Not-ECT</td>
<td>Not ECN-Capable Transport</td>
</tr>
<tr>
<td>01</td>
<td>ECT(1)</td>
<td>ECN-Capable Transport (1)</td>
</tr>
<tr>
<td>10</td>
<td>ECT(0)</td>
<td>ECN-Capable Transport (0)</td>
</tr>
<tr>
<td>11</td>
<td>CE</td>
<td>Congestion Experienced</td>
</tr>
</tbody>
</table>

Table 1: The ECN Field in the IP Header

In the TCP header the first two bits in byte 14 are defined as flags for the use of ECN (CWR and ECE in Figure 1 [RFC3168]). A TCP client indicates it supports ECN by setting ECE=CWR=1 in the SYN, and an ECN-enabled server confirms ECN support by setting ECE=1 and CWR=0 in the SYN/ACK. On reception of a CE-marked packet at the IP layer, the Data Receiver starts to set the Echo Congestion Experienced (ECE) flag continuously in the TCP header of ACKs, which ensures the signal is received reliably even if ACKs are lost. The TCP sender confirms that it has received at least one ECE signal by responding with the congestion window reduced (CWR) flag, which allows the TCP receiver to stop repeating the ECN-Echo flag. This always leads to a full RTT of ACKs with ECE set. Thus any additional CE markings arriving within this RTT cannot be fed back.

The last bit in byte 13 of the TCP header was defined as the Nonce Sum (NS) for the ECN Nonce [RFC3540]. In the absence of widespread deployment RFC 3540 has been reclassified as historic [RFC8311] and
the respective flag has been marked as "reserved", making this TCP flag available for use by the AccECN experiment instead.

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               |           | N | C | E | U | A | P | R | S | F |
| Header Length | Reserved  | S | W | C | R | C | S | S | Y | I |
|               |           | R | E | G | K | H | T | N | N |
```

Figure 1: The (post-ECN Nonce) definition of the TCP header flags

2. AccECN Protocol Overview and Rationale

This section provides an informative overview of the AccECN protocol that will be normatively specified in Section 3.

Like the original TCP approach, the Data Receiver of each TCP half-connection sends AccECN feedback to the Data Sender on TCP acknowledgements, reusing data packets of the other half-connection whenever possible.

The AccECN protocol has had to be designed in two parts:

- an essential part that re-uses ECN TCP header bits to feed back the number of arriving CE marked packets. This provides more accuracy than classic ECN feedback, but limited resilience against ACK loss;

- a supplementary part using a new AccECN TCP Option that provides additional feedback on the number of bytes that arrive marked with each of the three ECN codepoints (not just CE marks). This provides greater resilience against ACK loss than the essential feedback, but it is more likely to suffer from middlebox interference.

The two part design was necessary, given limitations on the space available for TCP options and given the possibility that certain incorrectly designed middleboxes prevent TCP using any new options.

The essential part overloads the previous definition of the three flags in the TCP header that had been assigned for use by ECN. This design choice deliberately replaces the classic ECN feedback protocol, rather than leaving classic ECN feedback intact and adding more accurate feedback separately because:

- this efficiently reuses scarce TCP header space, given TCP option space is approaching saturation;
o a single upgrade path for the TCP protocol is preferable to a fork in the design;

o otherwise classic and accurate ECN feedback could give conflicting feedback on the same segment, which could open up new security concerns and make implementations unnecessarily complex;

o middleboxes are more likely to faithfully forward the TCP ECN flags than newly defined areas of the TCP header.

AccECN is designed to work even if the supplementary part is removed or zeroed out, as long as the essential part gets through.

2.1. Capability Negotiation

AccECN is a change to the wire protocol of the main TCP header, therefore it can only be used if both endpoints have been upgraded to understand it. The TCP client signals support for AccECN on the initial SYN of a connection and the TCP server signals whether it supports AccECN on the SYN/ACK. The TCP flags on the SYN that the client uses to signal AccECN support have been carefully chosen so that a TCP server will interpret them as a request to support the most recent variant of ECN feedback that it supports. Then the client falls back to the same variant of ECN feedback.

An AccECN TCP client does not send the new AccECN Option on the SYN as SYN option space is limited and successful negotiation using the flags in the main header is taken as sufficient evidence that both ends also support the AccECN Option. The TCP server sends the AccECN Option on the SYN/ACK and the client sends it on the first ACK to test whether the network path forwards the option correctly.

2.2. Feedback Mechanism

A Data Receiver maintains four counters initialised at the start of the half-connection. Three count the number of arriving payload bytes marked CE, ECT(1) and ECT(0) respectively. The fourth counts the number of packets arriving marked with a CE codepoint (including control packets without payload if they are CE-marked).

The Data Sender maintains four equivalent counters for the half connection, and the AccECN protocol is designed to ensure they will match the values in the Data Receiver’s counters, albeit after a little delay.

Each ACK carries the three least significant bits (LSBs) of the packet-based CE counter using the ECN bits in the TCP header, now
renamed the Accurate ECN (ACE) field (see Figure 2 later). The LSBs of each of the three byte counters are carried in the AccECN Option.

2.3. Delayed ACKs and Resilience Against ACK Loss

With both the ACE and the AccECN Option mechanisms, the Data Receiver continually repeats the current LSBs of each of its respective counters. There is no need to acknowledge these continually repeated counters, so the congestion window reduced (CWR) mechanism is no longer used. Even if some ACKs are lost, the Data Sender should be able to infer how much to increment its own counters, even if the protocol field has wrapped.

The 3-bit ACE field can wrap fairly frequently. Therefore, even if it appears to have incremented by one (say), the field might have actually cycled completely then incremented by one. The Data Receiver is required not to delay sending an ACK to such an extent that the ACE field would cycle. However, cycling is still a possibility at the Data Sender because a whole sequence of ACKs carrying intervening values of the field might all be lost or delayed in transit.

The fields in the AccECN Option are larger, but they will increment in larger steps because they count bytes not packets. Nonetheless, their size has been chosen such that a whole cycle of the field would never occur between ACKs unless there had been an infeasibly long sequence of ACK losses. Therefore, as long as the AccECN Option is available, it can be treated as a dependable feedback channel.

If the AccECN Option is not available, e.g. it is being stripped by a middlebox, the AccECN protocol will only feed back information on CE markings (using the ACE field). Although not ideal, this will be sufficient, because it is envisaged that neither ECT(0) nor ECT(1) will ever indicate more severe congestion than CE, even though future uses for ECT(0) or ECT(1) are still unclear [RFC8311]. Because the 3-bit ACE field is so small, when it is the only field available the Data Sender has to interpret it conservatively assuming the worst possible wrap.

Certain specified events trigger the Data Receiver to include an AccECN Option on an ACK. The rules are designed to ensure that the order in which different markings arrive at the receiver is communicated to the sender (as long as there is no ACK loss). Implementations are encouraged to send an AccECN Option more frequently, but this is left up to the implementer.
2.4. Feedback Metrics

The CE packet counter in the ACE field and the CE byte counter in the AccECN Option both provide feedback on received CE-marks. The CE packet counter includes control packets that do not have payload data, while the CE byte counter solely includes marked payload bytes. If both are present, the byte counter in the option will provide the more accurate information needed for modern congestion control and policing schemes, such as DCTCP or ConEx. If the option is stripped, a simple algorithm to estimate the number of marked bytes from the ACE field is given in Appendix A.3.

Feedback in bytes is recommended in order to protect against the receiver using attacks similar to ‘ACK-Division’ to artificially inflate the congestion window, which is why [RFC5681] now recommends that TCP counts acknowledged bytes not packets.

2.5. Generic (Dumb) Reflector

The ACE field provides information about CE markings on both data and control packets. According to [RFC3168] the Data Sender is meant to set control packets to Not-ECT. However, mechanisms in certain private networks (e.g. data centres) set control packets to be ECN capable because they are precisely the packets that performance depends on most.

For this reason, AccECN is designed to be a generic reflector of whatever ECN markings it sees, whether or not they are compliant with a current standard. Then as standards evolve, Data Senders can upgrade unilaterally without any need for receivers to upgrade too. It is also useful to be able to rely on generic reflection behaviour when senders need to test for unexpected interference with markings (for instance [I-D.kuehlewind-tcpm-ecn-fallback] and para 2 of Section 20.2 of [RFC3168]).

The initial SYN is the most critical control packet, so AccECN provides feedback on its ECN marking. Although RFC 3168 prohibits an ECN-capable SYN, providing feedback on the state of the ECN field when it arrives at the receiver could still be useful, because middleboxes have been known to overwrite the ECN IP field as if it is still part of the old Type of Service (ToS) field [Mandalari18]. If
a TCP client has set the SYN to Not-ECT, but receives feedback that the ECN field on the SYN arrived with a different codepoint, it can detect such middlebox interference and send Not-ECT for the rest of the connection (see [I-D.kuehlewind-tcpm-ecn-fallback]). Today, if a TCP server receives ECT or CE on a SYN, it cannot know whether it is invalid (or valid) because only the TCP client knows whether it originally marked the SYN as Not-ECT (or ECT). Therefore, prior to AccECN, the server’s only safe course of action was to disable ECN for the connection. Instead, the AccECN protocol allows the server to feed back the received ECN field to the client, which then has all the information to decide whether the connection has to fall-back from supporting ECN (or not).

3. AccECN Protocol Specification

3.1. Negotiating to use AccECN

3.1.1. Negotiation during the TCP handshake

Given the ECN Nonce [RFC3540] has been reclassified as historic [RFC8311], the present specification re-allocates the TCP flag at bit 7 of the TCP header, which was previously called NS (Nonce Sum), as the AE (Accurate ECN) flag (see IANA Considerations in Section 6).

During the TCP handshake at the start of a connection, to request more accurate ECN feedback the TCP client (host A) MUST set the TCP flags AE=1, CWR=1 and ECE=1 in the initial SYN segment.

If a TCP server (B) that is AccECN-enabled receives a SYN with the above three flags set, it MUST set both its half connections into AccECN mode. Then it MUST set the TCP flags on the SYN/ACK to one of the 4 values shown in the top block of Table 2 to confirm that it supports AccECN. The TCP server MUST NOT set one of these 4 combination of flags on the SYN/ACK unless the preceding SYN requested support for AccECN as above.

A TCP server in AccECN mode MUST set the AE, CWR and ECE TCP flags on the SYN/ACK to the value in Table 2 that feeds back the IP-ECN field that arrived on the SYN. This applies whether or not the server itself supports setting the IP-ECN field on a SYN or SYN/ACK (see Section 2.5 for rationale).

Once a TCP client (A) has sent the above SYN to declare that it supports AccECN, and once it has received the above SYN/ACK segment that confirms that the TCP server supports AccECN, the TCP client MUST set both its half connections into AccECN mode.
The procedure for the client to follow if a SYN/ACK does not arrive before its retransmission timer expires is given in Section 3.1.3.

The three flags set to 1 to indicate AccECN support on the SYN have been carefully chosen to enable natural fall-back to prior stages in the evolution of ECN. Table 2 tabulates all the negotiation possibilities for ECN-related capabilities that involve at least one AccECN-capable host. The entries in the first two columns have been abbreviated, as follows:

AccECN:  More Accurate ECN Feedback (the present specification)
Nonce:  ECN Nonce feedback [RFC3540]
ECN:  ‘Classic’ ECN feedback [RFC3168]
No ECN:  Not-ECN-capable. Implicit congestion notification using packet drop.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>SYN A-&gt;B</th>
<th>SYN/ACK B-&gt;A</th>
<th>Feedback Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccECN</td>
<td>AccECN</td>
<td>1 1 1</td>
<td>0 1 0</td>
<td>AccECN (Not-ECT on SYN)</td>
</tr>
<tr>
<td>AccECN</td>
<td>AccECN</td>
<td>1 1 1</td>
<td>0 1 1</td>
<td>AccECN (ECT1 on SYN)</td>
</tr>
<tr>
<td>AccECN</td>
<td>AccECN</td>
<td>1 1 1</td>
<td>1 0 0</td>
<td>AccECN (ECT0 on SYN)</td>
</tr>
<tr>
<td>AccECN</td>
<td>Nonce</td>
<td>1 1 1</td>
<td>1 1 1</td>
<td>classic ECN</td>
</tr>
<tr>
<td>AccECN</td>
<td>ECN</td>
<td>1 1 1</td>
<td>0 0 1</td>
<td>classic ECN</td>
</tr>
<tr>
<td>AccECN</td>
<td>No ECN</td>
<td>1 1 1</td>
<td>0 0 0</td>
<td>Not ECN</td>
</tr>
<tr>
<td>AccECN</td>
<td>Broken</td>
<td>1 1 1</td>
<td>1 1 1</td>
<td>Not ECN</td>
</tr>
</tbody>
</table>

Table 2: ECN capability negotiation between Client (A) and Server (B)

Table 2 is divided into blocks each separated by an empty row.

1. The top block shows the case already described where both endpoints support AccECN and how the TCP server (B) indicates congestion feedback.
2. The second block shows the cases where the TCP client (A) supports AccECN but the TCP server (B) supports some earlier variant of TCP feedback, indicated in its SYN/ACK. Therefore, as soon as an AccECN-capable TCP client (A) receives the SYN/ACK shown it MUST set both its half connections into the feedback mode shown in the rightmost column.

3. The third block shows the cases where the TCP server (B) supports AccECN but the TCP client (A) supports some earlier variant of TCP feedback, indicated in its SYN. Therefore, as soon as an AccECN-enabled TCP server (B) receives the SYN shown, it MUST set both its half connections into the feedback mode shown in the rightmost column.

4. The fourth block displays a combination labelled ‘Broken’. Some older TCP server implementations incorrectly set the reserved flags in the SYN/ACK by reflecting those in the SYN. Such broken TCP servers (B) cannot support ECN, so as soon as an AccECN-capable TCP client (A) receives such a broken SYN/ACK it MUST fall-back to Not ECN mode for both its half connections.

The following exceptional cases need some explanation:

ECN Nonce: With AccECN implementation, there is no need for the ECN Nonce feedback mode [RFC3540], which has been reclassified as historic [RFC8311], as AccECN is compatible with an alternative ECN feedback integrity approach that does not use up the ECT(1) codepoint and can be implemented solely at the sender (see Section 4.3).

Simultaneous Open: An originating AccECN Host (A), having sent a SYN with AE=1, CWR=1 and ECE=1, might receive another SYN from host B. Host A MUST then enter the same feedback mode as it would have entered had it been a responding host and received the same SYN. Then host A MUST send the same SYN/ACK as it would have sent had it been a responding host.

3.1.2. Forward Compatibility

If a TCP server that implements AccECN receives a SYN with the three TCP header flags (AE, CWR and ECE) set to any combination other than 000, 011 or 111, it MUST negotiate the use of AccECN as if they had been set to 111. This ensures that future uses of the other combinations on a SYN can rely on consistent behaviour from the installed base of AccECN servers.

For the avoidance of doubt, the negotiation tabulated in Table 2 solely concerns the three TCP header flags shown (AE, CWR and ECE).
An AccECN host (client or server) MUST ignore the three remaining reserved TCP header flags on all packets.

3.1.3. Retransmission of the SYN

If the sender of an AccECN SYN times out before receiving the SYN/ACK, the sender SHOULD attempt to negotiate the use of AccECN at least one more time by continuing to set all three TCP ECN flags on the first retransmitted SYN (using the usual retransmission time-outs). If this first retransmission also fails to be acknowledged, the sender SHOULD send subsequent retransmissions of the SYN without any TCP-ECN flags set. This adds delay, in the case where a middlebox drops an AccECN (or ECN) SYN deliberately. However, current measurements imply that a drop is less likely to be due to middlebox interference than other intermittent causes of loss, e.g. congestion, wireless interference, etc.

Implementers MAY use other fall-back strategies if they are found to be more effective (e.g. attempting to negotiate AccECN on the SYN only once or more than twice (most appropriate during high levels of congestion); or falling back to classic ECN feedback rather than non-ECN). Further it may make sense to also remove any other experimental fields or options on the SYN in case a middlebox might be blocking them, although the required behaviour will depend on the specification of the other option(s) and any attempt to co-ordinate fall-back between different modules of the stack. In any case, the TCP initiator SHOULD cache failed connection attempts. If it does, it SHOULD NOT give up attempting to negotiate AccECN on the SYN of subsequent connection attempts until it is clear that the blockage is persistently and specifically due to AccECN. The cache should be arranged to expire so that the initiator will infrequently attempt to check whether the problem has been resolved.

The fall-back procedure if the TCP server receives no ACK to acknowledge a SYN/ACK that tried to negotiate AccECN is specified in Section 3.2.7.

3.2. AccECN Feedback

Each Data Receiver of each half connection maintains four counters, r.cep, r.ceb, r.e0b and r.elb. The CE packet counter (r.cep), counts the number of packets the host receives with the CE code point in the IP ECN field, including CE marks on control packets without data. r.ceb, r.e0b and r.elb count the number of TCP payload bytes in packets marked respectively with the CE, ECT(0) and ECT(1) codepoint in their IP-ECN field. When a host first enters AccECN mode, it initializes its counters to r.cep = 5, r.e0b = 1 and r.ceb = r.elb = 0 (see Appendix A.5). Non-zero initial values are used to support a
stateless handshake (see Section 4.1) and to be distinct from cases
where the fields are incorrectly zeroed (e.g. by middleboxes – see
Section 3.2.7.4).

A host feeds back the CE packet counter using the Accurate ECN (ACE)
field, as explained in the next section. And it feeds back all the
byte counters using the AccECN TCP Option, as specified in
Section 3.2.6. Whenever a host feeds back the value of any counter,
it MUST report the most recent value, no matter whether it is in a
pure ACK, an ACK with new payload data or a retransmission.
Therefore the feedback carried on a retransmitted packet is unlikely
to be the same as the feedback on the original packet.

3.2.1. Initialization of Feedback Counters at the Data Sender

Each Data Sender of each half connection maintains four counters,
s.cep, s.ceb, s.e0b and s.e1b intended to track the equivalent
counters at the Data Receiver. When a host enters AccECN mode, it
initializes them to s.cep = 5, s.e0b = 1 and s.ceb = s.e1b.= 0.

If a TCP client (A) in AccECN mode receives a SYN/ACK with CE
feedback, i.e. AE=1, CWR=1, ECE=0, it increments s.cep to 6.
Otherwise, for any of the 3 other combinations of the 3 ECN TCP flags
(the top 3 rows in Table 2), s.cep remains initialized to 5.

3.2.2. The ACE Field

After AccECN has been negotiated on the SYN and SYN/ACK, both hosts
overload the three TCP flags (AE, CWR and ECE) in the main TCP header
as one 3-bit field. Then the field is given a new name, ACE, as
shown in Figure 2.

```
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
+-----------------------------------------------+
<table>
<thead>
<tr>
<th>Header Length</th>
<th>Reserved</th>
<th>ACE</th>
<th>U</th>
<th>A</th>
<th>P</th>
<th>R</th>
<th>S</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>G</td>
<td>K</td>
<td>H</td>
<td>T</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
+-----------------------------------------------+
```

Figure 2: Definition of the ACE field within bytes 13 and 14 of the
TCP Header (when AccECN has been negotiated and SYN=0).

The original definition of these three flags in the TCP header,
including the addition of support for the ECN Nonce, is shown for
comparison in Figure 1. This specification does not rename these
three TCP flags to ACE unconditionally; it merely overloads them with
another name and definition once an AccECN connection has been
established.
A host MUST interpret the AE, CWR and ECE flags as the 3-bit ACE counter on a segment with the SYN flag cleared (SYN=0) that it sends or receives if both of its half-connections are set into AccECN mode having successfully negotiated AccECN (see Section 3.1). A host MUST NOT interpret the 3 flags as a 3-bit ACE field on any segment with SYN=1 (whether ACK is 0 or 1), or if AccECN negotiation is incomplete or has not succeeded.

Both parts of each of these conditions are equally important. For instance, even if AccECN negotiation has been successful, the ACE field is not defined on any segments with SYN=1 (e.g. a retransmission of an unacknowledged SYN/ACK, or when both ends send SYN/ACKs after AccECN support has been successfully negotiated during a simultaneous open).

With only one exception, on any packet with the SYN flag cleared (SYN=0), the Data Receiver MUST encode the three least significant bits of its r.cep counter into the ACE field it feeds back to the Data Sender.

There is only one exception to this rule: On the final ACK of the 3-way handshake (3WHS), a TCP client (A) in AccECN mode MUST use the ACE field to feed back which of the 4 possible values of the IP-ECN field were on the SYN/ACK (the binary encoding is the same as that used on the SYN/ACK). Table 3 shows the meaning of each possible value of the ACE field on the ACK of the SYN/ACK and the value that an AccECN server MUST set s.cep to as a result. The encoding in Table 3 is solely applicable on a packet in the client-server direction with an acknowledgement number 1 greater than the Initial Sequence Number (ISN) that was used by the server.

<table>
<thead>
<tr>
<th>ACE on ACK of SYN/ACK</th>
<th>IP-ECN codepoint on SYN/ACK inferred by server</th>
<th>Initial s.cep of server in AccECN mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>(Notes 1, 2)</td>
<td>Disable ECN</td>
</tr>
<tr>
<td>0b001</td>
<td>(Notes 2, 3)</td>
<td>5</td>
</tr>
<tr>
<td>0b010</td>
<td>Not-ECT</td>
<td>5</td>
</tr>
<tr>
<td>0b011</td>
<td>ECT(1)</td>
<td>5</td>
</tr>
<tr>
<td>0b100</td>
<td>ECT(0)</td>
<td>5</td>
</tr>
<tr>
<td>0b101</td>
<td>Currently Unused (Note 3)</td>
<td>5</td>
</tr>
<tr>
<td>0b110</td>
<td>CE</td>
<td>6</td>
</tr>
<tr>
<td>0b111</td>
<td>Currently Unused (Note 3)</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 3: Meaning of the ACE field on the ACK of the SYN/ACK
Internet-Draft          Accurate TCP-ECN Feedback             March 2019

(Note 1): If the server is in AccECN mode, the value of zero raises suspicion of zeroing of the ACE field on the path (see Section 3.2.3).

(Note 2): If a server is in AccECN mode, there ought to be no valid case where the ACE field on the last ACK of the 3WHS has a value of 0b000 or 0b001.

However, in the case where a server that implements AccECN is also using a stateless handshake (termed a SYN cookie) it will not remember whether it entered AccECN mode. Then these two values remind it that it did not enter AccECN mode (see Section 4.1 for details).

(Note 3): If the server is in AccECN mode, these values are Currently Unused but the AccECN server’s behaviour is still defined for forward compatibility.

3.2.3. Testing for Zeroing of the ACE Field

Section 3.2.2 required the Data Receiver to initialize the r.cep counter to a non-zero value. Therefore, in either direction the initial value of the ACE field ought to be non-zero.

If AccECN has been successfully negotiated, the Data Sender SHOULD check the initial value of the ACE field in the first arriving segment with SYN=0. If the initial value of the ACE field is zero (0b000), the Data Sender MUST disable sending ECN-capable packets for the remainder of the half-connection by setting the IP/ECN field in all subsequent packets to Not-ECT.

For example, the server checks the ACK of the SYN/ACK or the first data segment from the client, while the client checks the first data segment from the server. More precisely, the "first segment with SYN=0" is defined as: the segment with SYN=0 that i) acknowledges sequence space at least covering the initial sequence number (ISN) plus 1; and ii) arrives before any other segments with SYN=0 so it is unlikely to be a retransmission. If no such segment arrives (e.g. because it is lost and the ISN is first acknowledged by a subsequent segment), no test for invalid initialization can be conducted, and the half-connection will continue in AccECN mode.

Note that the Data Sender MUST NOT test whether the arriving counter in the initial ACE field has been initialized to a specific valid value - the above check solely tests whether the ACE fields have been incorrectly zeroed. This allows hosts to use different initial values as an additional signalling channel in future.
3.2.4. Testing for Mangling of the IP/ECN Field

The value of the ACE field on the SYN/ACK indicates the value of the IP/ECN field when the SYN arrived at the server. The client can compare this with how it originally set the IP/ECN field on the SYN. If this comparison implies an unsafe transition (see below) of the IP/ECN field, for the remainder of the connection the client MUST NOT send ECN-capable packets, but it MUST continue to feedback any ECN markings on arriving packets.

The value of the ACE field on the last ACK of the 3WHs indicates the value of the IP/ECN field when the SYN/ACK arrived at the client. The server can compare this with how it originally set the IP/ECN field on the SYN/ACK. If this comparison implies an unsafe transition of the IP/ECN field, for the remainder of the connection the server MUST NOT send ECN-capable packets, but it MUST continue to feedback any ECN markings on arriving packets.

The ACK of the SYN/ACK is not reliably delivered (nonetheless, the count of CE marks is still eventually delivered reliably). If this ACK does not arrive, the server can continue to send ECN-capable packets without having tested for mangling of the IP/ECN field on the SYN/ACK. Experiments with AccECN deployment will assess whether this limitation has any effect in practice.

Invalid transitions of the IP/ECN field are defined in [RFC3168] and repeated here for convenience:

- the not-ECT codepoint changes;
- either ECT codepoint transitions to not-ECT;
- the CE codepoint changes.

RFC 3168 says that a router that changes ECT to not-ECT is invalid but safe. However, from a host’s viewpoint, this transition is unsafe because it could be the result of two transitions at different routers on the path: ECT to CE (safe) then CE to not-ECT (unsafe). This scenario could well happen where an ECN-enabled home router congests its upstream mobile broadband bottleneck link, then the ingress to the mobile network clears the ECN field [Mandalari18].

The above fall-back behaviours are necessary in case mangling of the IP/ECN field is asymmetric, which is currently common over some mobile networks [Mandalari18]. Then one end might see no unsafe transition and continue sending ECN-capable packets, while the other end sees an unsafe transition and stops sending ECN-capable packets.
3.2.5. Safety against Ambiguity of the ACE Field

If too many CE-marked segments are acknowledged at once, or if a long run of ACKs is lost, the 3-bit counter in the ACE field might have cycled between two ACKs arriving at the Data Sender.

Therefore an AccECN Data Receiver SHOULD immediately send an ACK once ‘n’ CE marks have arrived since the previous ACK, where ‘n’ SHOULD be 2 and MUST be no greater than 6.

If the Data Sender has not received AccECN TCP Options to give it more dependable information, and it detects that the ACE field could have cycled under the prevailing conditions, it SHOULD conservatively assume that the counter did cycle. It can detect if the counter could have cycled by using the jump in the acknowledgement number since the last ACK to calculate or estimate how many segments could have been acknowledged. An example algorithm to implement this policy is given in Appendix A.2. An implementer MAY develop an alternative algorithm as long as it satisfies these requirements.

If missing acknowledgement numbers arrive later (reordering) and prove that the counter did not cycle, the Data Sender MAY attempt to neutralise the effect of any action it took based on a conservative assumption that it later found to be incorrect.

3.2.6. The AccECN Option

The AccECN Option is defined as shown below in Figure 3. It consists of three 24-bit fields that provide the 24 least significant bits of the r.e0b, r.ceb and r.e1b counters, respectively. The initial ‘E’ of each field name stands for ‘Echo’.

```
| Kind = TBD1 | Length = 11 | EE0B field |
| EEOB field |
| EEB (cont’d) | ECEB field |
| EE1B field |
```

Figure 3: The AccECN Option

When a Data Receiver sends an AccECN Option, it MUST set the Kind field to TBD1, which is registered in Section 6 as a new TCP option Kind called AccECN. An experimental TCP option with Kind=254 MAY be used for initial experiments, with magic number 0xACCE.
Appendix A.1 gives an example algorithm for the Data Receiver to encode its byte counters into the AccECN Option, and for the Data Sender to decode the AccECN Option fields into its byte counters.

Note that there is no field to feed back Not-ECT bytes. Nonetheless an algorithm for the Data Sender to calculate the number of payload bytes received as Not-ECT is given in Appendix A.5.

Whenever a Data Receiver sends an AccECN Option, the rules in Section 3.2.8 expect it to always send a full-length option. To cope with option space limitations, it can omit unchanged fields from the tail of the option, as long as it preserves the order of the remaining fields and includes any field that has changed. The length field MUST indicate which fields are present as follows:

Length=11: EE0B, ECEB, EE1B
Length=8: EE0B, ECEB
Length=5: EE0B
Length=2: (empty)

The empty option of Length=2 is provided to allow for a case where an AccECN Option has to be sent (e.g. on the SYN/ACK to test the path), but there is very limited space for the option. For initial experiments, the Length field MUST be 2 greater to accommodate the 16-bit magic number.

All implementations of a Data Sender that read any AccECN Option MUST be able to read in AccECN Options of any of the above lengths. If the AccECN Option is of any other length, implementations MUST use those whole 3 octet fields that fit within the length and ignore the remainder of the option.

The AccECN Option has to be optional to implement, because both sender and receiver have to be able to cope without the option anyway - in cases where it does not traverse a network path. It is RECOMMENDED to implement both sending and receiving of the AccECN Option. If sending of the AccECN Option is implemented, the fall-backs described in this document will need to be implemented as well (unless solely for a controlled environment where path traversal is not considered a problem). Even if a developer does not implement sending of the AccECN Option, it is RECOMMENDED that they still implement logic to receive and understand any AccECN Options sent by remote peers.
If a Data Receiver intends to send the AccECN Option at any time during the rest of the connection it is strongly recommended to also test path traversal of the AccECN Option as specified in the next section.

3.2.7. Path Traversal of the AccECN Option

3.2.7.1. Testing the AccECN Option during the Handshake

The TCP client MUST NOT include the AccECN TCP Option on the SYN. A fall-back strategy for the loss of the SYN (possibly due to middlebox interference) is specified in Section 3.1.3.

A TCP server that confirms its support for AccECN (in response to an AccECN SYN from the client as described in Section 3.1) SHOULD include an AccECN TCP Option in the SYN/ACK.

A TCP client that has successfully negotiated AccECN SHOULD include an AccECN Option in the first ACK at the end of the 3WHS. However, this first ACK is not delivered reliably, so the TCP client SHOULD also include an AccECN Option on the first data segment it sends (if it ever sends one).

A host MAY NOT include an AccECN Option in any of these three cases if it has cached knowledge that the packet would be likely to be blocked on the path to the other host if it included an AccECN Option.

3.2.7.2. Testing for Loss of Packets Carrying the AccECN Option

If after the normal TCP timeout the TCP server has not received an ACK to acknowledge its SYN/ACK, the SYN/ACK might just have been lost, e.g. due to congestion, or a middlebox might be blocking the AccECN Option. To expedite connection setup, the TCP server SHOULD retransmit the SYN/ACK repeating the AE, CWR and ECE TCP flags on the original SYN/ACK but with no AccECN Option. If this retransmission times out, to expedite connection setup, the TCP server SHOULD disable AccECN and ECN for this connection by retransmitting the SYN/ACK with AE=CWR=ECE=0 and no AccECN Option. Implementers MAY use other fall-back strategies if they are found to be more effective (e.g. falling back to classic ECN feedback on the first retransmission; retrying the AccECN Option for a second time before fall-back (most appropriate during high levels of congestion); or falling back to classic ECN feedback rather than non-ECN on the third retransmission).

If the TCP client detects that the first data segment it sent with the AccECN Option was lost, it SHOULD fall back to no AccECN Option.
on the retransmission. Again, implementers MAY use other fall-back strategies such as attempting to retransmit a second segment with the AccECN Option before fall-back, and/or caching whether the AccECN Option is blocked for subsequent connections.

Either host MAY include the AccECN Option in a subsequent segment to retest whether the AccECN Option can traverse the path.

If the TCP server receives a second SYN with a request for AccECN support, it should resend the SYN/ACK, again confirming its support for AccECN, but this time without the AccECN Option. This approach rules out any interference by middleboxes that may drop packets with unknown options, even though it is more likely that the SYN/ACK would have been lost due to congestion. The TCP server MAY try to send another packet with the AccECN Option at a later point during the connection but should monitor if that packet got lost as well, in which case it SHOULD disable the sending of the AccECN Option for this half-connection.

Similarly, an AccECN end-point MAY separately memorize which data packets carried an AccECN Option and disable the sending of AccECN Options if the loss probability of those packets is significantly higher than that of all other data packets in the same connection.

3.2.7.3. Testing for Stripping of the AccECN Option

If the TCP client has successfully negotiated AccECN but does not receive an AccECN Option on the SYN/ACK, it switches into a mode that assumes that the AccECN Option is not available for this half connection.

Similarly, if the TCP server has successfully negotiated AccECN but does not receive an AccECN Option on the first segment that acknowledges sequence space at least covering the ISN, it switches into a mode that assumes that the AccECN Option is not available for this half connection.

While a host is in this mode that assumes incoming AccECN Options are not available, it MUST adopt the conservative interpretation of the ACE field discussed in Section 3.2.5. However, it cannot make any assumption about support of outgoing AccECN Options on the other half connection, so it SHOULD continue to send the AccECN Option itself (unless it has established that sending the AccECN Option is causing packets to be blocked as in Section 3.2.7.2).

If a host is in the mode that assumes incoming AccECN Options are not available, but it receives an AccECN Option at any later point during the connection, this clearly indicates that the AccECN Option is not
blocked on the respective path, and the AccECN endpoint MAY switch out of the mode that assumes the AccECN Option is not available for this half connection.

3.2.7.4. Test for Zeroing of the AccECN Option

For a related test for invalid initialization of the ACE field, see Section 3.2.3

Section 3.2 required the Data Receiver to initialize the r.e0b counter to a non-zero value. Therefore, in either direction the initial value of the EE0B field in the AccECN Option (if one exists) ought to be non-zero. If AccECN has been negotiated:

- the TCP server MAY check the initial value of the EE0B field in the first segment that acknowledges sequence space that at least covers the ISN plus 1. If the initial value of the EE0B field is zero, the server will switch into a mode that ignores the AccECN Option for this half connection.

- the TCP client MAY check the initial value of the EE0B field on the SYN/ACK. If the initial value of the EE0B field is zero, the client will switch into a mode that ignores the AccECN Option for this half connection.

While a host is in the mode that ignores the AccECN Option it MUST adopt the conservative interpretation of the ACE field discussed in Section 3.2.5.

Note that the Data Sender MUST NOT test whether the arriving byte counters in the initial AccECN Option have been initialized to specific valid values - the above checks solely test whether these fields have been incorrectly zeroed. This allows hosts to use different initial values as an additional signalling channel in future. Also note that the initial value of either field might be greater than its expected initial value, because the counters might already have been incremented. Nonetheless, the initial values of the counters have been chosen so that they cannot wrap to zero on these initial segments.

3.2.7.5. Consistency between AccECN Feedback Fields

When the AccECN Option is available it supplements but does not replace the ACE field. An endpoint using AccECN feedback MUST always consider the information provided in the ACE field whether or not the AccECN Option is also available.
If the AccECN option is present, the s.cep counter might increase while the s.ceb counter does not (e.g. due to a CE-marked control packet). The sender’s response to such a situation is out of scope, and needs to be dealt with in a specification that uses ECN-capable control packets. Theoretically, this situation could also occur if a middlebox mangled the AccECN Option but not the ACE field. However, the Data Sender has to assume that the integrity of the AccECN Option is sound, based on the above test of the well-known initial values and optionally other integrity tests (Section 4.3).

If either end-point detects that the s.ceb counter has increased but the s.cep has not (and by testing ACK coverage it is certain how much the ACE field has wrapped), this invalid protocol transition has to be due to some form of feedback mangling. So, the Data Sender MUST disable sending ECN-capable packets for the remainder of the half-connection by setting the IP/ECN field in all subsequent packets to Not-ECT.

3.2.8. Usage of the AccECN TCP Option

The following rules determine when a Data Receiver in AccECN mode sends the AccECN TCP Option, and which fields to include:

Change-Triggered ACKs: If an arriving packet increments a different byte counter to that incremented by the previous packet, the Data Receiver MUST immediately send an ACK with an AccECN Option, without waiting for the next delayed ACK (this is in addition to the safety recommendation in Section 3.2.5 against ambiguity of the ACE field).

This is stated as a "MUST" so that the data sender can rely on change-triggered ACKs to detect transitions right from the very start of a flow, without first having to detect whether the receiver complies. A concern has been raised that certain offload hardware needed for high performance might not be able to support change-triggered ACKs, although high performance protocols such as DCTCP successfully use change-triggered ACKs. One possible experimental compromise would be for the receiver to heuristically detect whether the sender is in slow-start, then to implement change-triggered ACKs in software while the sender is in slow-start, and offload to hardware otherwise. If the operator disables change-triggered ACKs, whether partially like this or otherwise, the operator will also be responsible for ensuring a co-ordinated sender algorithm is deployed;

Continual Repetition: Otherwise, if arriving packets continue to increment the same byte counter, the Data Receiver can include an AccECN Option on most or all (delayed) ACKs, but it does not have
to. If option space is limited on a particular ACK, the Data Receiver MUST give precedence to SACK information about loss. It SHOULD include an AccECN Option if the r.ceb counter has incremented and it MAY include an AccECN Option if r.ec0b or r.ec1b has incremented;

Full-Length Options Preferred: It SHOULD always use full-length AccECN Options. It MAY use shorter AccECN Options if space is limited, but it MUST include the counter(s) that have incremented since the previous AccECN Option and it MUST only truncate fields from the right-hand tail of the option to preserve the order of the remaining fields (see Section 3.2.6);

Beaconing Full-Length Options: Nonetheless, it MUST include a full-length AccECN TCP Option on at least three ACKs per RTT, or on all ACKs if there are less than three per RTT (see Appendix A.4 for an example algorithm that satisfies this requirement).

The following example series of arriving IP/ECN fields illustrates when a Data Receiver will emit an ACK if it is using a delayed ACK factor of 2 segments and change-triggered ACKs: 01 -> ACK, 01, 01 -> ACK, 10 -> ACK, 10, 01 -> ACK, 01, 11 -> ACK, 01 -> ACK.

For the avoidance of doubt, the change-triggered ACK mechanism is deliberately worded to ignore the arrival of a control packet with no payload, which therefore does not alter any byte counters, because it is important that TCP does not acknowledge pure ACKs. The change-triggered ACK approach can lead to some additional ACKs but it feeds back the timing and the order in which ECN marks are received with minimal additional complexity. If only CE marks are infrequent, or there are multiple marks in a row, the additional load will be low. Other marking patterns could increase the load significantly, Investigating the additional load is a goal of the proposed experiment.

Implementation note: sending an AccECN Option each time a different counter changes and including a full-length AccECN Option on every delayed ACK will satisfy the requirements described above and might be the easiest implementation, as long as sufficient space is available in each ACK (in total and in the option space).

Appendix A.3 gives an example algorithm to estimate the number of marked bytes from the ACE field alone, if the AccECN Option is not available.

If a host has determined that segments with the AccECN Option always seem to be discarded somewhere along the path, it is no longer obliged to follow the above rules.
3.3. Requirements for TCP Proxies, Offload Engines and other Middleboxes on AccECN Compliance

A large class of middleboxes split TCP connections. Such a middlebox would be compliant with the AccECN protocol if the TCP implementation on each side complied with the present AccECN specification and each side negotiated AccECN independently of the other side.

Another large class of middleboxes intervenes to some degree at the transport layer, but attempts to be transparent (invisible) to the end-to-end connection. A subset of this class of middleboxes attempts to ‘normalise’ the TCP wire protocol by checking that all values in header fields comply with a rather narrow interpretation of the TCP specifications. To comply with the present AccECN specification, such a middlebox MUST NOT change the ACE field or the AccECN Option and it SHOULD preserve the timing of each ACK (for example, if it coalesced ACKs it would not be AccECN-compliant) as these can be used by the Data Sender to infer further information about the path congestion level. A middlebox claiming to be transparent at the transport layer MUST forward the AccECN TCP Option unaltered, whether or not the length value matches one of those specified in Section 3.2.6, and whether or not the initial values of the byte-counter fields are correct. This is because blocking apparently invalid values does not improve security (because AccECN hosts are required to ignore invalid values anyway), while it prevents the standardised set of values being extended in future (because outdated normalisers would block updated hosts from using the extended AccECN standard).

Hardware to offload certain TCP processing represents another large class of middleboxes, even though it is often a function of a host’s network interface and rarely in its own ‘box’. Leeway has been allowed in the present AccECN specification in the expectation that offload hardware could comply and still serve its function. Nonetheless, such hardware SHOULD also preserve the timing of each ACK (for example, if it coalesced ACKs it would not be AccECN-compliant).

The ACE field changes with every received CE marking, so today’s receive offloading could lead to many interrupts in high congestion situations. Although that would be useful (because congestion information is received sooner), it could also significantly increase processor load, particularly in scenarios such as DCTCP or L4S where the marking rate is generally higher.

In data centres it has been fortunate for offload hardware that DCTCP-style feedback changes less often when there are long sequences of CE marks, which is more common with a step marking threshold. In
order to enable DCTCP to improve its responsiveness, DCs will need to move beyond step marking. Before this can happen, offload hardware will have to explicitly address the variability of ECN feedback.

ECN encodes a varying signal in the ACK stream, so it is inevitable that offload hardware will ultimately need to handle any form of ECN feedback exceptionally. The purpose of working towards standardized TCP ECN feedback is to reduce the risk for hardware developers, who will have to choose which scheme is likely to become dominant.

4. Interaction with Other TCP Variants

This section is informative, not normative.

4.1. Compatibility with SYN Cookies

A TCP server can use SYN Cookies (see Appendix A of [RFC4987]) to protect itself from SYN flooding attacks. It places minimal commonly used connection state in the SYN/ACK, and deliberately does not hold any state while waiting for the subsequent ACK (e.g. it closes the thread). Therefore it cannot record the fact that it entered AccECN mode for both half-connections. Indeed, it cannot even remember whether it negotiated the use of classic ECN [RFC3168].

Nonetheless, such a server can determine that it negotiated AccECN as follows. If a TCP server using SYN Cookies supports AccECN and if it receives a pure ACK that acknowledges an ISN that is a valid SYN cookie, and if the ACK contains an ACE field with the value 0b010 to 0b111 (decimal 2 to 7), it can assume that:

- the TCP client must have requested AccECN support on the SYN
- it (the server) must have confirmed that it supported AccECN

Therefore the server can switch itself into AccECN mode, and continue as if it had never forgotten that it switched itself into AccECN mode earlier.

If the pure ACK that acknowledges a SYN cookie contains an ACE field with the value 0b000 or 0b001, these values indicate that the client did not request support for AccECN and therefore the server does not enter AccECN mode for this connection. Further, 0b001 on the ACK implies that the server sent an ECN-capable SYN/ACK, which was marked CE in the network, and the non-AccECN client fed this back by setting ECE on the ACK of the SYN/ACK.
4.2. Compatibility with Other TCP Options and Experiments

AccECN is compatible (at least on paper) with the most commonly used TCP options: MSS, time-stamp, window scaling, SACK and TCP-AO. It is also compatible with the recent promising experimental TCP options TCP Fast Open (TFO [RFC7413]) and Multipath TCP (MPTCP [RFC6824]). AccECN is friendly to all these protocols, because space for TCP options is particularly scarce on the SYN, where AccECN consumes zero additional header space.

When option space is under pressure from other options, Section 3.2.8 provides guidance on how important it is to send an AccECN Option and whether it needs to be a full-length option.

4.3. Compatibility with Feedback Integrity Mechanisms

Three alternative mechanisms are available to assure the integrity of ECN and/or loss signals. AccECN is compatible with any of these approaches:

- The Data Sender can test the integrity of the receiver’s ECN (or loss) feedback by occasionally setting the IP-ECN field to a value normally only set by the network (and/or deliberately leaving a sequence number gap). Then it can test whether the Data Receiver’s feedback faithfully reports what it expects (similar to para 2 of Section 20.2 of [RFC3168]). Unlike the ECN Nonce [RFC3540], this approach does not waste the ECT(1) codepoint in the IP header, it does not require standardisation and it does not rely on misbehaving receivers volunteering to reveal feedback information that allows them to be detected. However, setting the CE mark by the sender might conceal actual congestion feedback from the network and should therefore only be done sparsely.

- Networks generate congestion signals when they are becoming congested, so networks are more likely than Data Senders to be concerned about the integrity of the receiver’s feedback of these signals. A network can enforce a congestion response to its ECN markings (or packet losses) using congestion exposure (ConEx) audit [RFC7713]. Whether the receiver or a downstream network is suppressing congestion feedback or the sender is unresponsive to the feedback, or both, ConEx audit can neutralise any advantage that any of these three parties would otherwise gain.

ConEx is a change to the Data Sender that is most useful when combined with AccECN. Without AccECN, the ConEx behaviour of a Data Sender would have to be more conservative than would be necessary if it had the accurate feedback of AccECN.
The TCP authentication option (TCP-AO [RFC5925]) can be used to detect any tampering with AccECN feedback between the Data Receiver and the Data Sender (whether malicious or accidental). The AccECN fields are immutable end-to-end, so they are amenable to TCP-AO protection, which covers TCP options by default. However, TCP-AO is often too brittle to use on many end-to-end paths, where middleboxes can make verification fail in their attempts to improve performance or security, e.g. by resegmentation or shifting the sequence space.

Originally the ECNNonce [RFC3540] was proposed to ensure integrity of congestion feedback. With minor changes AccECN could be optimised for the possibility that the ECT(1) codepoint might be used as an ECN Nonce. However, given RFC 3540 has been reclassified as historic, the AccECN design has been generalised so that it ought to be able to support other possible uses of the ECT(1) codepoint, such as a lower severity or a more instant congestion signal than CE.

5. Protocol Properties

This section is informative not normative. It describes how well the protocol satisfies the agreed requirements for a more accurate ECN feedback protocol [RFC7560].

Accuracy: From each ACK, the Data Sender can infer the number of new CE marked segments since the previous ACK. This provides better accuracy on CE feedback than classic ECN. In addition if the AccECN Option is present (not blocked by the network path) the number of bytes marked with CE, ECT(1) and ECT(0) are provided.

Overhead: The AccECN scheme is divided into two parts. The essential part reuses the 3 flags already assigned to ECN in the IP header. The supplementary part adds an additional TCP option consuming up to 11 bytes. However, no TCP option is consumed in the SYN.

Ordering: The order in which marks arrive at the Data Receiver is preserved in AccECN feedback, because the Data Receiver is expected to send an ACK immediately whenever a different mark arrives.

Timeliness: While the same ECN markings are arriving continually at the Data Receiver, it can defer ACKs as TCP does normally, but it will immediately send an ACK as soon as a different ECN marking arrives.

Timeliness vs Overhead: Change-Triggered ACKs are intended to enable latency-sensitive uses of ECN feedback by capturing the timing of
transitions but not wasting resources while the state of the signalling system is stable. The receiver can control how frequently it sends the AccECN TCP Option and therefore it can control the overhead induced by AccECN.

Resilience: All information is provided based on counters. Therefore if ACKs are lost, the counters on the first ACK following the losses allows the Data Sender to immediately recover the number of the ECN markings that it missed.

Resilience against Bias: Because feedback is based on repetition of counters, random losses do not remove any information, they only delay it. Therefore, even though some ACKs are change-triggered, random losses will not alter the proportions of the different ECN markings in the feedback.

Resilience vs Overhead: If space is limited in some segments (e.g. because more option are need on some segments, such as the SACK option after loss), the Data Receiver can send AccECN Options less frequently or truncate fields that have not changed, usually down to as little as 5 bytes. However, it has to send a full-sized AccECN Option at least three times per RTT, which the Data Sender can rely on as a regular beacon or checkpoint.

Resilience vs Timeliness and Ordering: Ordering information and the timing of transitions cannot be communicated in three cases: i) during ACK loss; ii) if something on the path strips the AccECN Option; or iii) if the Data Receiver is unable to support Change-Triggered ACKs.

Complexity: An AccECN implementation solely involves simple counter increments, some modulo arithmetic to communicate the least significant bits and allow for wrap, and some heuristics for safety against fields cycling due to prolonged periods of ACK loss. Each host needs to maintain eight additional counters. The hosts have to apply some additional tests to detect tampering by middleboxes, but in general the protocol is simple to understand, simple to implement and requires few cycles per packet to execute.

Integrity: AccECN is compatible with at least three approaches that can assure the integrity of ECN feedback. If the AccECN Option is stripped the resolution of the feedback is degraded, but the integrity of this degraded feedback can still be assured.

Backward Compatibility: If only one endpoint supports the AccECN scheme, it will fall-back to the most advanced ECN feedback scheme supported by the other end.
Backward Compatibility: If the AccECN Option is stripped by a middlebox, AccECN still provides basic congestion feedback in the ACE field. Further, AccECN can be used to detect mangling of the IP ECN field; mangling of the TCP ECN flags; blocking of ECT-marked segments; and blocking of segments carrying the AccECN Option. It can detect these conditions during TCP’s 3WHS so that it can fall back to operation without ECN and/or operation without the AccECN Option.

Forward Compatibility: The behaviour of endpoints and middleboxes is carefully defined for all reserved or currently unused codepoints in the scheme, to ensure that any blocking of anomalous values is always at least under reversible policy control.

6. IANA Considerations

This document reassigns bit 7 of the TCP header flags to the AccECN experiment. This bit was previously called the Nonce Sum (NS) flag [RFC3540], but RFC 3540 has been reclassified as historic [RFC8311]. The flag will now be defined as:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>AE (Accurate ECN)</td>
<td>RFC XXXX</td>
</tr>
</tbody>
</table>

[TO BE REMOVED: IANA is requested to update the existing entry in the Transmission Control Protocol (TCP) Header Flags registration (https://www.iana.org/assignments/tcp-header-flags/tcp-header-flags.xhtml#tcp-header-flags-1) for Bit 7 to "AE (Accurate ECN), previously used as NS (Nonce Sum) by [RFC3540], which is now Historic [RFC8311]" and change the reference to this RFC-to-be instead of RFC8311.]

This document also defines a new TCP option for AccECN, assigned a value of TBD1 (decimal) from the TCP option space. This value is defined as:

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Meaning</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD1</td>
<td>N</td>
<td>Accurate ECN (AccECN)</td>
<td>RFC XXXX</td>
</tr>
</tbody>
</table>

[TO BE REMOVED: This registration should take place at the following location: http://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml#tcp-parameters-1]
Early implementation before the IANA allocation MUST follow [RFC6994] and use experimental option 254 and magic number 0xACCE (16 bits), then migrate to the new option after the allocation.

7. Security Considerations

If ever the supplementary part of AccECN based on the new AccECN TCP Option is unusable (due for example to middlebox interference) the essential part of AccECN’s congestion feedback offers only limited resilience to long runs of ACK loss (see Section 3.2.5). These problems are unlikely to be due to malicious intervention (because if an attacker could strip a TCP option or discard a long run of ACKs it could wreak other arbitrary havoc). However, it would be of concern if AccECN’s resilience could be indirectly compromised during a flooding attack. AccECN is still considered safe though, because if the option is not presented, the AccECN Data Sender is then required to switch to more conservative assumptions about wrap of congestion indication counters (see Section 3.2.5 and Appendix A.2).

Section 4.1 describes how a TCP server can negotiate AccECN and use the SYN cookie method for mitigating SYN flooding attacks.

There is concern that ECN markings could be altered or suppressed, particularly because a misbehaving Data Receiver could increase its own throughput at the expense of others. AccECN is compatible with the three schemes known to assure the integrity of ECN feedback (see Section 4.3 for details). If the AccECN Option is stripped by an incorrectly implemented middlebox, the resolution of the feedback will be degraded, but the integrity of this degraded information can still be assured.

There is a potential concern that a receiver could deliberately omit the AccECN Option pretending that it had been stripped by a middlebox. No known way can yet be contrived to take advantage of this downgrade attack, but it is mentioned here in case someone else can contrive one.

The AccECN protocol is not believed to introduce any new privacy concerns, because it merely counts and feeds back signals at the transport layer that had already been visible at the IP layer.

8. Acknowledgements

We want to thank Koen De Schepper, Praveen Balasubramanian, Michael Welzl, Gorry Fairhurst, David Black, Spencer Dawkins, Michael Scharf and Michael Tuexen for their input and discussion. The idea of using the three ECN-related TCP flags as one field for more accurate TCP-
ECN feedback was first introduced in the re-ECN protocol that was the ancestor of ConEx.

Bob Briscoe was part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700) and through the Trilogy 2 project (ICT-317756). He was also part-funded by the Research Council of Norway through the TimelIn project. The views expressed here are solely those of the authors.

Mirja Kuehlewind was partly supported by the European Commission under Horizon 2020 grant agreement no. 688421 Measurement and Architecture for a Middleboxed Internet (MAMI), and by the Swiss State Secretariat for Education, Research, and Innovation under contract no. 15.0268. This support does not imply endorsement.

9. Comments Solicited

Comments and questions are encouraged and very welcome. They can be addressed to the IETF TCP maintenance and minor modifications working group mailing list <tcpm@ietf.org>, and/or to the authors.

10. References

10.1. Normative References


Briscoe, et al. Expires September 12, 2019 [Page 34]
10.2. Informative References

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

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

[I-D.kuehlewind-tcpm-ecn-fallback]

[Mandalari18]
(to appear)


Appendix A. Example Algorithms

This appendix is informative, not normative. It gives example algorithms that would satisfy the normative requirements of the AccECN protocol. However, implementers are free to choose other ways to implement the requirements.

A.1. Example Algorithm to Encode/Decode the AccECN Option

The example algorithms below show how a Data Receiver in AccECN mode could encode its CE byte counter r.ceb into the ECEB field within the AccECN TCP Option, and how a Data Sender in AccECN mode could decode the ECEB field into its byte counter s.ceb. The other counters for bytes marked ECT(0) and ECT(1) in the AccECN Option would be similarly encoded and decoded.

It is assumed that each local byte counter is an unsigned integer greater than 24b (probably 32b), and that the following constant has been assigned:

\[ \text{DIVOPT} = 2^{24} \]

Every time a CE marked data segment arrives, the Data Receiver increments its local value of r.ceb by the size of the TCP Data. Whenever it sends an ACK with the AccECN Option, the value it writes into the ECEB field is

\[ \text{ECEB} = r.ceb \mod \text{DIVOPT} \]

where ‘\mod’ is the modulo operator.

On the arrival of an AccECN Option, the Data Sender uses the TCP acknowledgement number and any SACK options to calculate newlyAckedB, the amount of new data that the ACK acknowledges in bytes. If newlyAckedB is negative it means that a more up to date ACK has already been processed, so this ACK has been superseded and the Data Sender has to ignore the AccECN Option. Then the Data Sender calculates the minimum difference d.ceb between the ECEB field and its local s.ceb counter, using modulo arithmetic as follows:

```
if (newlyAckedB >= 0) {
  d.ceb = (ECEB + DIVOPT - (s.ceb % DIVOPT)) % DIVOPT
  s.ceb += d.ceb
}
```

For example, if s.ceb is 33,554,433 and ECEB is 1461 (both decimal), then
\[
s.ceb \% \text{DIVOPT} = 1 \\
d.ceb = (1461 + 2^{24} - 1) \% 2^{24} \\
= 1460 \\
s.ceb = 33,554,433 + 1460 \\
= 33,555,893
\]

A.2. Example Algorithm for Safety Against Long Sequences of ACK Loss

The example algorithms below show how a Data Receiver in AccECN mode could encode its CE packet counter \( r.\text{cep} \) into the ACE field, and how the Data Sender in AccECN mode could decode the ACE field into its \( s.\text{cep} \) counter. The Data Sender’s algorithm includes code to heuristically detect a long enough unbroken string of ACK losses that could have concealed a cycle of the congestion counter in the ACE field of the next ACK to arrive.

Two variants of the algorithm are given: i) a more conservative variant for a Data Sender to use if it detects that the AccECN Option is not available (see Section 3.2.5 and Section 3.2.7); and ii) a less conservative variant that is feasible when complementary information is available from the AccECN Option.

A.2.1. Safety Algorithm without the AccECN Option

It is assumed that each local packet counter is a sufficiently sized unsigned integer (probably 32b) and that the following constant has been assigned:

\[
\text{DIVACE} = 2^3
\]

Every time a CE marked packet arrives, the Data Receiver increments its local value of \( r.\text{cep} \) by 1. It repeats the same value of ACE in every subsequent ACK until the next CE marking arrives, where

\[
\text{ACE} = r.\text{cep} \% \text{DIVACE}.
\]

If the Data Sender received an earlier value of the counter that had been delayed due to ACK reordering, it might incorrectly calculate that the ACE field had wrapped. Therefore, on the arrival of every ACK, the Data Sender uses the TCP acknowledgement number and any SACK options to calculate \( \text{newlyAckedB} \), the amount of new data that the ACK acknowledges. If \( \text{newlyAckedB} \) is negative it means that a more up to date ACK has already been processed, so this ACK has been superseded and the Data Sender has to ignore the AccECN Option. If \( \text{newlyAckedB} \) is zero, to break the tie the Data Sender could use timestamps (if present) to work out \( \text{newlyAckedT} \), the amount of new time that the ACK acknowledges. Then the Data Sender calculates the minimum difference...
d.cep between the ACE field and its local s.cep counter, using modulo arithmetic as follows:

\[
\text{if } ((\text{newlyAckedB} > 0) \text{ or } (\text{newlyAckedB} == 0 \&\& \text{newlyAckedT} > 0)) \\
\text{d.cep} = (\text{ACE} + \text{DIVACE} - (\text{s.cep} \mod \text{DIVACE})) \mod \text{DIVACE}
\]

Section 3.2.5 requires the Data Sender to assume that the ACE field did cycle if it could have cycled under prevailing conditions. The 3-bit ACE field in an arriving ACK could have cycled and become ambiguous to the Data Sender if a row of ACKs goes missing that covers a stream of data long enough to contain 8 or more CE marks. We use the word 'missing' rather than 'lost', because some or all the missing ACKs might arrive eventually, but out of order. Even if some of the lost ACKs are piggy-backed on data (i.e. not pure ACKs) retransmissions will not repair the lost AccECN information, because AccECN requires retransmissions to carry the latest AccECN counters, not the original ones.

The phrase 'under prevailing conditions' allows the Data Sender to take account of the prevailing size of data segments and the prevailing CE marking rate just before the sequence of ACK losses. However, we shall start with the simplest algorithm, which assumes segments are all full-sized and ultra-conservatively it assumes that ECN marking was 100% on the forward path when ACKs on the reverse path started to all be dropped. Specifically, if newlyAckedB is the amount of data that an ACK acknowledges since the previous ACK, then the Data Sender could assume that this acknowledges newlyAckedPkt full-sized segments, where newlyAckedPkt = newlyAckedB/MSS. Then it could assume that the ACE field incremented by

\[
\text{dSafer.cep} = \text{newlyAckedPkt} - (((\text{newlyAckedPkt} - \text{d.cep}) \mod \text{DIVACE}),
\]

For example, imagine an ACK acknowledges newlyAckedPkt=9 more full-size segments than any previous ACK, and that ACE increments by a minimum of 2 CE marks (d.cep=2). The above formula works out that it would still be safe to assume 2 CE marks (because 9 - ((9-2) % 8) = 2). However, if ACE increases by a minimum of 2 but acknowledges 10 full-sized segments, then it would be necessary to assume that there could have been 10 CE marks (because 10 - ((10-2) % 8) = 10).

Implementers could build in more heuristics to estimate prevailing average segment size and prevailing ECN marking. For instance, newlyAckedPkt in the above formula could be replaced with newlyAckedPktHeur = newlyAckedPkt*p*MSS/s, where s is the prevailing segment size and p is the prevailing ECN marking probability. However, ultimately, if TCP's ECN feedback becomes inaccurate it still has loss detection to fall back on. Therefore, it would seem safe to implement a simple algorithm, rather than a perfect one.
The simple algorithm for \texttt{dSafer.cep} above requires no monitoring of prevailing conditions and it would still be safe if, for example, segments were on average at least 5\% of full-sized as long as ECN marking was 5\% or less. Assuming it was used, the Data Sender would increment its packet counter as follows:

\[
s.\texttt{cep} += \texttt{dSafer.cep}
\]

If missing acknowledgement numbers arrive later (due to reordering), Section 3.2.5 says "the Data Sender \texttt{MAY} attempt to neutralise the effect of any action it took based on a conservative assumption that it later found to be incorrect". To do this, the Data Sender would have to store the values of all the relevant variables whenever it made assumptions, so that it could re-evaluate them later. Given this could become complex and it is not required, we do not attempt to provide an example of how to do this.

### A.2.2. Safety Algorithm with the AccECN Option

When the AccECN Option is available on the ACKs before and after the possible sequence of ACK losses, if the Data Sender only needs CE-marked bytes, it will have sufficient information in the AccECN Option without needing to process the ACE field. However, if for some reason it needs CE-marked packets, if \texttt{dSafer.cep} is different from \texttt{d.cep}, it can calculate the average marked segment size that each implies to determine whether \texttt{d.cep} is likely to be a safe enough estimate. Specifically, it could use the following algorithm, where \texttt{d.ceb} is the amount of newly CE-marked bytes (see Appendix A.1):

\[
\texttt{SAFETY\_FACTOR} = 2 \\
\text{if} (\texttt{dSafer.cep} > \texttt{d.cep}) \\
\quad \texttt{s} = \texttt{d.ceb}/\texttt{d.cep} \\
\quad \text{if} (\texttt{s} \leq \texttt{MSS}) \\
\quad \quad \texttt{sSafer} = \texttt{d.ceb}/\texttt{dSafer.cep} \\
\quad \quad \text{if} (\texttt{sSafer} < \texttt{MSS}/\texttt{SAFETY\_FACTOR}) \\
\quad \quad \quad \texttt{dSafer.cep} = \texttt{d.cep} % \texttt{d.cep} is a safe enough estimate \\
\quad \quad \% \texttt{else} \\
\quad \quad \% \texttt{No need for else; \texttt{dSafer.cep} is already correct,} \\
\quad \quad \% \texttt{because \texttt{d.cep} must have been too small}
\]

The chart below shows when the above algorithm will consider \texttt{d.cep} can replace \texttt{dSafer.cep} as a safe enough estimate of the number of CE-marked packets:
The following examples give the reasoning behind the algorithm, assuming MSS=1,460 [B]:

- if d.cep=0, dSafer.cep=8 and d.ceb=1,460, then s=infinity and sSafer=182.5. Therefore even though the average size of 8 data segments is unlikely to have been as small as MSS/8, d.cep cannot have been correct, because it would imply an average segment size greater than the MSS.

- if d.cep=2, dSafer.cep=10 and d.ceb=1,460, then s=730 and sSafer=146. Therefore d.cep is safe enough, because the average size of 10 data segments is unlikely to have been as small as MSS/10.

- if d.cep=7, dSafer.cep=15 and d.ceb=10,200, then s=1,457 and sSafer=680. Therefore d.cep is safe enough, because the average data segment size is more likely to have been just less than one MSS, rather than below MSS/2.

If pure ACKs were allowed to be ECN-capable, missing ACKs would be far less likely. However, because [RFC3168] currently precludes this, the above algorithm assumes that pure ACKs are not ECN-capable.

### A.3. Example Algorithm to Estimate Marked Bytes from Marked Packets

If the AccECN Option is not available, the Data Sender can only decode CE-marking from the ACE field in packets. Every time an ACK arrives, to convert this into an estimate of CE-marked bytes, it needs an average of the segment size, s_ave. Then it can add or subtract s_ave from the value of d.ceb as the value of d.cep increments or decrements.
To calculate $s_{ave}$, it could keep a record of the byte numbers of all the boundaries between packets in flight (including control packets), and recalculate $s_{ave}$ on every ACK. However it would be simpler to merely maintain a counter packets\_in\_flight for the number of packets in flight (including control packets), which it could update once per RTT. Either way, it would estimate $s_{ave}$ as:

$$s_{ave} \approx \text{flightsize} / \text{packets\_in\_flight},$$

where flightsize is the variable that TCP already maintains for the number of bytes in flight. To avoid floating point arithmetic, it could right-bit-shift by $\lg(\text{packets\_in\_flight})$, where $\lg()$ means log base 2.

An alternative would be to maintain an exponentially weighted moving average (EWMA) of the segment size:

$$s_{ave} = a \times s + (1-a) \times s_{ave},$$

where $a$ is the decay constant for the EWMA. However, then it is necessary to choose a good value for this constant, which ought to depend on the number of packets in flight. Also the decay constant needs to be power of two to avoid floating point arithmetic.

A.4.  Example Algorithm to Beacon AccECN Options

Section 3.2.8 requires a Data Receiver to beacon a full-length AccECN Option at least 3 times per RTT. This could be implemented by maintaining a variable to store the number of ACKs (pure and data ACKs) since a full AccECN Option was last sent and another for the approximate number of ACKs sent in the last round trip time:

```java
if (acks_since_full_last_sent > acks_in_round / BEACON_FREQ)
send_full_AccECN_Option()
```

For optimised integer arithmetic, $\text{BEACON\_FREQ} = 4$ could be used, rather than 3, so that the division could be implemented as an integer right bit-shift by $\lg(\text{BEACON\_FREQ})$.

In certain operating systems, it might be too complex to maintain $\text{acks\_in\_round}$. In others it might be possible by tagging each data segment in the retransmit buffer with the number of ACKs sent at the point that segment was sent. This would not work well if the Data Receiver was not sending data itself, in which case it might be necessary to beacon based on time instead, as follows:

```java
if ( time_now > time_last_option_sent + (RTT / BEACON_FREQ) )
send_full_AccECN_Option()
```
This time-based approach does not work well when all the ACKs are sent early in each round trip, as is the case during slow-start. In this case few options will be sent (evtl. even less than 3 per RTT). However, when continuously sending data, data packets as well as ACKs will spread out equally over the RTT and sufficient ACKs with the AccECN option will be sent.

A.5. Example Algorithm to Count Not-ECT Bytes

A Data Sender in AccECN mode can infer the amount of TCP payload data arriving at the receiver marked Not-ECT from the difference between the amount of newly ACKed data and the sum of the bytes with the other three markings, d.ceb, d.e0b and d.e1b. Note that, because r.e0b is initialized to 1 and the other two counters are initialized to 0, the initial sum will be 1, which matches the initial offset of the TCP sequence number on completion of the 3WHS.

For this approach to be precise, it has to be assumed that spurious (unnecessary) retransmissions do not lead to double counting. This assumption is currently correct, given that RFC 3168 requires that the Data Sender marks retransmitted segments as Not-ECT. However, the converse is not true; necessary transmissions will result in under-counting.

However, such precision is unlikely to be necessary. The only known use of a count of Not-ECT marked bytes is to test whether equipment on the path is clearing the ECN field (perhaps due to an out-dated attempt to clear, or bleach, what used to be the ToS field). To detect bleaching it will be sufficient to detect whether nearly all bytes arrive marked as Not-ECT. Therefore there should be no need to keep track of the details of retransmissions.

Appendix B. Rationale for Usage of TCP Header Flags

B.1. Three TCP Header Flags in the SYN-SYN/ACK Handshake

AccECN uses a rather unorthodox but justified approach to negotiate the highest version TCP ECN feedback scheme that both ends support. It follows from the original TCP ECN capability negotiation [RFC3168], in which the client set the 2 least significant reserved flags in the TCP header, and fell back to no ECN support if the server responded with the 2 flags cleared, which had previously been the default. It is not recorded why ECN originally used this approach instead of the more orthodox use of a TCP option.

In order to be backward compatible with RFC 3168, AccECN continues this approach, using the 3rd least significant TCP header flag that had previously been allocated for the ECN nonce (now historic).
Then, whatever form of server an AccECN client encounters, the connection can fall back to the highest version of feedback protocol that both ends support, as explained in Section 3.1.

If AccECN had used the more orthodox approach of a TCP option, it would still have to set the two ECN flags in the main TCP header, in order to be able to fall back to Classic RFC 3168 ECN, or to disable ECN support, without another round of negotiation. Then AccECN would also have had to handle all the different ways that servers currently respond to settings of the ECN flags in the main TCP header, including all the conflicting cases where a server might have said it supported one approach in the flags and another approach in the new TCP option. And AccECN would have had to deal with all the additional possibilities where a middlebox might have mangled the ECN flags, or removed the TCP option. Thus, usage of the 3rd reserved TCP header flag simplified the protocol.

The third flag was used in a way that could be distinguished from the ECN nonce, in case any nonce deployment was encountered. Previous usage of this flag for the ECN nonce was integrated into the original ECN negotiation. This further justified the 3rd flag’s use for AccECN, because a non-ECN usage of this flag would have had to use it as a separate single bit, rather than in combination with the other 2 ECN flags.

Indeed, having overloaded the original uses of these three flags for its handshake, AccECN overloads all three bits again as a 3-bit counter.

B.2. Four Codepoints in the SYN/ACK

Of the 8 possible codepoints that the 3 TCP header flags can indicate on the SYN/ACK, 4 already indicated earlier (or broken) versions of ECN support. In the early design of AccECN, an AccECN server could use only 2 of the 4 remaining codepoints. They both indicated AccECN support, but one fed back that the SYN had arrived marked as CE. Even though ECN support on a SYN is not yet on the standards track, the idea is for either end to act as a dumb reflector, so that future capabilities can be unilaterally deployed without requiring 2-ended deployment (justified in Section 2.5).

During traversal testing it was discovered that the ECN field in the SYN was mangled on a non-negligible proportion of paths. Therefore it was necessary to allow the SYN/ACK to feed all four IP/ECN codepoints that the SYN could arrive with back to the client. Without this, the client could not know whether to disable ECN for the connection due to mangling of the IP/ECN field (also explained in Section 2.5). This development consumed the remaining 2 codepoints.
on the SYN/ACK that had been reserved for future use by AccECN in earlier versions.

B.3. Space for Future Evolution

Despite availability of usable TCP header space being extremely scarce, the AccECN protocol has taken all possible steps to ensure that there is space to negotiate possible future variants of the protocol, either if the experiment proves that a variant of AccECN is required, or if a completely different ECN feedback approach is needed:

Future AccECN variants: When the AccECN capability is negotiated during TCP’s 3WHS, the rows in Table 2 tagged as ‘Nonce’ and ‘Broken’ in the column for the capability of node B are unused by any current protocol in the RFC series. These could be used by TCP servers in future to indicate a variant of the AccECN protocol. In recent measurement studies in which the response of large numbers of servers to an AccECN SYN has been tested, e.g. [Mandalari18], a very small number of SYN/ACKs arrive with the pattern tagged as ‘Nonce’, and a small but more significant number arrive with the pattern tagged as ‘Broken’. The ‘Nonce’ pattern could be a sign that a few servers have implemented the ECN Nonce [RFC3540], which has now been reclassified as historic [RFC8311], or it could be the random result of some unknown middlebox behaviour. The greater prevalence of the ‘Broken’ pattern suggests that some instances still exist of the broken code that reflects the reserved flags on the SYN.

The requirement not to reject unexpected initial values of the ACE counter (in the main TCP header) in the last para of Section 3.2.3 ensures that 5 unused codepoints on the final ACK of the 3WHS and 7 unused values on the first data packet from the server could be used to declare future variants of the AccECN protocol. The word ‘declare’ is used rather than ‘negotiate’ because, at this late stage in the 3WHS, it would be too late for a negotiation between the endpoints to be completed. A similar requirement not to reject unexpected initial values in the TCP option (Section 3.2.7.4) is for the same purpose. If traversal of the TCP option were reliable, this would have enabled a far wider range of future variation of the whole AccECN protocol. Nonetheless, it could be used to reliably negotiate a wide range of variation in the semantics of the AccECN Option.

Future non-AccECN variants: Five codepoints out of the 8 possible in the 3 TCP header flags used by AccECN are unused on the initial SYN (in the order AE,CWR,ECE): 001, 010, 100, 101, 110. Section 3.1.2 ensures that the installed base of AccECN servers
will all assume these are equivalent to AccECN negotiation with 111 on the SYN. These codepoints would not allow fall-back to Classic ECN support for a server that did not understand them, but this approach ensures they are available in future, perhaps for uses other than ECN alongside the AccECN scheme. All possible combinations of SYN/ACK could be used in response except either 000 or reflection of the same values sent on the SYN.

Of course, other ways could be resorted to in order to extend AccECN or ECN in future, although their traversal properties are likely to be inferior. They include a new TCP option; using the remaining reserved flags in the main TCP header (preferably extending the 3-bit combinations used by AccECN to 4-bit combinations, rather than burning one bit for just one state); a non-zero urgent pointer in combination with the URG flag cleared; or some other unexpected combination of fields yet to be invented.

Authors’ Addresses

Bob Briscoe
CableLabs
UK

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

Mirja Kuehlewind
ETH Zurich
Zurich
Switzerland

EMail: mirja.kuehlewind@tik.ee.ethz.ch

Richard Scheffenegger
Vienna
Austria

EMail: rscheff@gmx.at
Abstract

This document specifies an application proxy, called Transport Converter, to assist the deployment of TCP extensions such as Multipath TCP. This proxy is designed to avoid inducing extra delay when involved in a network-assisted connection (that is, 0-RTT).

This specification assumes an explicit model, where the proxy is explicitly configured on hosts.

-- Editorial Note (To be removed by RFC Editor)

Please update these statements with the RFC number to be assigned to this document: [This-RFC]

Please update TBA statements with the port number to be assigned to the 0-RTT TCP Convert Protocol.

Status of This Memo

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

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."
This Internet-Draft will expire on December 20, 2019.

Copyright Notice

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

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

Table of Contents

1. Introduction .................................................. 3
2. Requirements .................................................. 5
3. Architecture ................................................... 6
   3.1. Functional Elements ........................................ 6
   3.2. Theory of Operation ......................................... 8
   3.3. Sample Examples of Outgoing Converter-Assisted Multipath
        TCP Connections .......................................... 11
   3.4. Sample Example of Incoming Converter-Assisted Multipath
        TCP Connection ............................................ 13
4. The Convert Protocol (Convert) ................................ 14
   4.1. The Convert Fixed Header .................................. 14
   4.2. Convert TLVs .............................................. 15
       4.2.1. Generic Convert TLV Format ............................ 15
       4.2.2. Summary of Supported Convert TLVs .................... 16
       4.2.3. The Info TLV .......................................... 17
       4.2.4. Supported TCP Extensions TLV .......................... 17
       4.2.5. Connect TLV ........................................... 18
       4.2.6. Extended TCP Header TLV ............................... 20
       4.2.7. The Cookie TLV ....................................... 20
       4.2.8. Error TLV ............................................. 21
5. Compatibility of Specific TCP Options with the Conversion
   Service ......................................................... 24
   5.1. Base TCP Options .......................................... 24
   5.2. Window Scale (WS) ......................................... 25
   5.3. Selective Acknowledgements ................................ 25
   5.4. Timestamp ................................................ 26
   5.5. Multipath TCP ............................................ 26
   5.6. TCP Fast Open ............................................ 26
   5.7. TCP User Timeout .......................................... 27
1. Introduction

Transport protocols like TCP evolve regularly [RFC7414]. TCP has been improved in different ways. Some improvements such as changing the initial window size [RFC6928] or modifying the congestion control scheme can be applied independently on clients and servers. Other improvements such as Selective Acknowledgements [RFC2018] or large windows [RFC7323] require a new TCP option or to change the semantics of some fields in the TCP header. These modifications must be deployed on both clients and servers to be actually used on the Internet. Experience with the latter TCP extensions reveals that their deployment can require many years. Fukuda reports in [Fukuda2011] results of a decade of measurements showing the deployment of Selective Acknowledgements, Window Scale and TCP Timestamps. [ANRW17] describes measurements showing that TCP Fast Open (TFO) [RFC7413] is still not widely deployed.

There are some situations where the transport stack used on clients (or servers) can be upgraded at a faster pace than the transport stack running on servers (or clients). In those situations, clients
would typically want to benefit from the features of an improved transport protocol even if the servers have not yet been upgraded and conversely. Performance Enhancing Proxies [RFC3135], and other service functions have been deployed as solutions to improve TCP performance over links with specific characteristics.

Recent examples of TCP extensions include Multipath TCP [RFC6824] or TCPINC [RFC8548]. Those extensions provide features that are interesting for clients such as wireless devices. With Multipath TCP, those devices could seamlessly use WLAN (Wireless Local Area Network) and cellular networks, for bonding purposes, faster handovers, or better resiliency. Unfortunately, deploying those extensions on both a wide range of clients and servers remains difficult.

More recently, experimentation of 5G bonding, which has very scarce coverage, has been conducted into global range of the incumbent 4G (LTE) connectivity in newly devised clients using Multipath TCP proxy. Even if the 5G and the 4G bonding by using Multipath TCP increases the bandwidth, it is as well crucial to minimize latency for all the way between endhosts regardless of whether intermediate nodes are inside or outside of the mobile core. In order to handle URLLC (Ultra Reliable Low Latency Communication) for the next generation mobile network, Multipath TCP and its proxy mechanism such as the one used to provide Access Traffic Steering, Switching, and Splitting (ATSSS) must be optimised to reduce latency [TS23501].

This document specifies an application proxy, called Transport Converter. A Transport Converter is a function that is installed by a network operator to aid the deployment of TCP extensions and to provide the benefits of such extensions to clients. A Transport Converter may provide conversion service for one or more TCP extensions. Which TCP extensions are eligible to the conversion service is deployment-specific. The conversion service is provided by means of the 0-RTT TCP Convert Protocol (Convert), that is an application-layer protocol which uses TCP port number TBA (Section 8).

The Transport Converter adheres to the main principles drawn in [RFC1919]. In particular, a Transport Converter achieves the following:

- Listen for client sessions;
- Receive from a client the address of the final target server;
- Setup a session to the final server;
The main advantage of network-assisted conversion services is that they enable new TCP extensions to be used on a subset of the path between endpoints, which encourages the deployment of these extensions. Furthermore, the Transport Converter allows the client and the server to directly negotiate TCP options for the sake of native support along the full path.

The Convert Protocol is a generic mechanism to provide 0-RTT conversion service. As a sample applicability use case, this document specifies how the Convert Protocol applies for Multipath TCP. It is out of scope of this document to provide a comprehensive list of all potential conversion services. Applicability documents may be defined in the future.

This document does not assume that all the traffic is eligible to the network-assisted conversion service. Only a subset of the traffic will be forwarded to a Transport Converter according to a set of policies. These policies, and how they are communicated to endpoints, are out of scope. Furthermore, it is possible to bypass the Transport Converter to connect directly to the servers that already support the required TCP extension(s).

This document assumes an explicit model in which a client is configured with one or a list of Transport Converters (statically or through protocols such as [I-D.boucadair-tcpm-dhc-converter]). Configuration means are outside the scope of this document.

This document is organized as follows. We first provide a brief explanation of the operation of Transport Converters in Section 3. We describe the Convert Protocol in Section 4. We discuss in Section 5 how Transport Converters can be used to support different TCP extensions. We then discuss the interactions with middleboxes (Section 6) and the security considerations (Section 7).

Appendix A discusses how a TCP stack would need to support the protocol described in this document. Appendix B provides a comparison with SOCKS proxies that are already used to deploy Multipath TCP in some cellular networks (Section 2.2 of [RFC8041]).

2. Requirements

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
3. Architecture

3.1. Functional Elements

The Convert Protocol considers three functional elements:

- Clients;
- Transport Converters;
- Servers.

A Transport Converter is a network function that relays all data exchanged over one upstream connection to one downstream connection and vice versa (Figure 1). The Transport Converter, thus, maintains state that associates one upstream connection to a corresponding downstream connection.

A connection can be initiated from both sides of the Transport Converter (Internet-facing interface, client-facing interface).

```
<p>| |</p>
<table>
<thead>
<tr>
<th align="center"></th>
</tr>
</thead>
<tbody>
<tr>
<td align="center">client &lt;- upstream -&gt; Transport Converter &lt;- downstream -&gt; server</td>
</tr>
<tr>
<td align="center">:----------------:</td>
</tr>
<tr>
<td align="center">client-facing interface : Internet-facing interface</td>
</tr>
</tbody>
</table>
```

Figure 1: A Transport Converter Relays Data between Pairs of TCP Connections

Transport Converters can be operated by network operators or third parties. Nevertheless, this document focuses on the single administrative deployment case where the entity offering the connectivity service to a client is also the entity which owns and operates the Transport Converter.

A Transport Converter can be embedded in a standalone device or be activated as a service on a router. How such function is enabled is deployment-specific. A sample deployment is depicted in Figure 2.
Figure 2: A Transport Converter Can Be Installed Anywhere in the Network

The architecture assumes that new software will be installed on the Client hosts to interact with one or more Transport Converters. Further, the architecture allows for making use of new TCP extensions even if those are not supported by a given server.

The Client is configured, through means that are outside the scope of this document, with the names and/or the addresses of one or more Transport Converters and the TCP extensions that they support. The procedure for selecting a Transport Converter among a list of configured Transport Converters is outside the scope of this document.

One of the benefits of this design is that different transport protocol extensions can be used on the upstream and the downstream connections. This encourages the deployment of new TCP extensions until they are widely supported by servers, in particular.

The architecture does not mandate anything on the server side.

Similar to address sharing mechanisms, the architecture does not interfere with end-to-end TLS connections [RFC8446] between the Client and the Server (Figure 3). In other words, end-to-end TLS is supported in the presence of a Converter.
**3.2. Theory of Operation**

At a high level, the objective of the Transport Converter is to allow the use a specific extension, e.g., Multipath TCP, on a subset of the path even if the peer does not support this extension. This is illustrated in Figure 4 where the Client initiates a Multipath TCP connection with the Transport Converter (packets belonging to the Multipath TCP connection are shown with "===") while the Transport Converter uses a regular TCP connection with the Server.

The packets belonging to the pair of connections between the Client and Server passing through a Transport Converter may follow a different path than the packets directly exchanged between the Client and the Server. Deployments should minimize the possible additional delay by carefully selecting the location of the Transport Converter used to reach a given destination.

When establishing a connection, the Client can, depending on local policies, either contact the Server directly (e.g., by sending a TCP SYN towards the Server) or create the connection via a Transport Converter.
Converter. In the latter case (that is, the conversion service is used), the Client initiates a connection towards the Transport Converter and indicates the IP address and port number of the Server within the connection establishment packet. Doing so enables the Transport Converter to immediately initiate a connection towards that Server, without experiencing an extra delay. The Transport Converter waits until the receipt of the confirmation that the Server agrees to establish the connection before confirming it to the Client.

The client places the destination address and port number of the Server in the payload of the SYN sent to the Transport Converter to minimize connection establishment delays. In accordance with [RFC1919], the Transport Converter maintains two connections that are combined together:

- the upstream connection is the one between the Client and the Transport Converter.
- the downstream connection is between the Transport Converter and the Server.

Any user data received by the Transport Converter over the upstream (or downstream) connection is relayed over the downstream (or upstream) connection. In particular, if the initial SYN message contains data in its payload (e.g., [RFC7413]), that data MUST be placed right after the Convert TLVs when generating the relayed SYN.

The Converter associates a lifetime with state entries used to bind an upstream connection with its downstream connection.

Figure 5 illustrates the establishment of an outbound TCP connection by a Client through a Transport Converter. The information shown between brackets denotes Convert Protocol messages described in Section 4.

```
<table>
<thead>
<tr>
<th>Client</th>
<th>Transport Converter</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN [-&gt;Server:port]</td>
<td>SYN</td>
<td>SYN+ACK</td>
</tr>
<tr>
<td>&lt;-----------------</td>
<td>-------------------</td>
<td>--------</td>
</tr>
<tr>
<td>SYN+ACK [ ]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 5: Establishment of a TCP Connection Through a Transport Converter (1)
The Client sends a SYN destined to the Transport Converter. The payload of this SYN contains the address and port number of the Server. The Transport Converter does not reply immediately to this SYN. It first tries to create a TCP connection towards the target Server. If this upstream connection succeeds, the Transport Converter confirms the establishment of the connection to the Client by returning a SYN+ACK and the first bytes of the bytestream contain information about the TCP options that were negotiated with the Server. This information is sent at the beginning of the bytestream, either directly in the SYN+ACK or in a subsequent packet. For graphical reasons, the figures in this section show that the Transport Converter returns this information in the SYN+ACK packet. An implementation could also place this information in a packet that it sent shortly after the SYN+ACK.

The connection can also be established from the Internet towards a Client via a Transport Converter. This is typically the case when an application on the Client listens to a specific port (the Client hosts a server, typically).

A Transport Converter MAY operate in address preservation or address sharing modes as discussed in Section 5.4 of [I-D.nam-mptcp-deployment-considerations]. Which behavior to use by a Transport Converter is deployment-specific. If address sharing mode is enabled, the Transport Converter MUST adhere to REQ-2 of [RFC6888] which implies a default "IP address pooling" behavior of "Paired" (as defined in Section 4.1 of [RFC4787]) must be supported. This behavior is meant to avoid breaking applications that depend on the external address remaining constant.

Standard TCP ([RFC0793], Section 3.4) allows a SYN packet to carry data inside its payload but forbids the receiver from delivering it to the application until completion of the three-way-handshake. To enable applications to exchange data in a TCP handshake, this specification follows an approach similar to TCP Fast Open [RFC7413] and thus removes the constraint by allowing data in SYN packets to be delivered to the Transport Converter application.

As discussed in [RFC7413], such change to TCP semantic raises two issues. First, duplicate SYNs can cause problems for some applications that rely on TCP. Second, TCP suffers from SYN flooding attacks [RFC4987]. TFO solves these two problems for applications that can tolerate replays by using the TCP Fast Open option that includes a cookie. However, the utilization of this option consumes space in the limited TCP header. Furthermore, there are situations, as noted in Section 7.3 of [RFC7413] where it is possible to accept the payload of SYN packets without creating additional security risks such as a network where addresses cannot be spoofed and the Transport
Converter only serves a set of hosts that are identified by these addresses.

For these reasons, this specification does not mandate the use of the TCP Fast Open option when the Client sends a connection establishment packet towards a Transport Converter. The Convert protocol includes an optional Cookie TLV that provides similar protection as the TCP Fast Open option without consuming space in the extended TCP header.

If the downstream (or upstream) connection fails for some reason (excessive retransmissions, reception of a RST segment, etc.), then the Converter should force the teardown of the upstream (or downstream) connection.

The same reasoning applies when the upstream connection ends. In this case, the Converter should also terminate the downstream connection by using FIN segments. If the downstream connection terminates with the exchange of FIN segments, the Converter should initiate a graceful termination of the upstream connection.

3.3. Sample Examples of Outgoing Converter-Assisted Multipath TCP Connections

As an example, let us consider how the Convert protocol can help the deployment of Multipath TCP. We assume that both the Client and the Transport Converter support Multipath TCP, but consider two different cases depending on whether the Server supports Multipath TCP or not.

As a reminder, a Multipath TCP connection is created by placing the MP\_CAPABLE (MPC) option in the SYN sent by the Client.

Figure 6 describes the operation of the Transport Converter if the Server does not support Multipath TCP.
The Client tries to initiate a Multipath TCP connection by sending a SYN with the MP_CAPABLE option (MPC in Figure 6). The SYN includes the address and port number of the target Server, that are extracted and used by the Transport Converter to initiate a Multipath TCP connection towards this Server. Since the Server does not support Multipath TCP, it replies with a SYN+ACK that does not contain the MP_CAPABLE option. The Transport Converter notes that the connection with the Server does not support Multipath TCP and returns the extended TCP header received from the Server to the Client.

Note that, if the TCP connection fails for some reason, the Converter tears down the Multipath TCP connection by transmitting a MP_FASTCLOSE. Likewise, if the Multipath TCP connection ends with the transmission of DATA_FINs, the Converter terminates the TCP connection by using FIN segments.

Figure 7 considers a Server that supports Multipath TCP. In this case, it replies to the SYN sent by the Transport Converter with the MP_CAPABLE option. Upon reception of this SYN+ACK, the Transport Converter confirms the establishment of the connection to the Client and indicates to the Client that the Server supports Multipath TCP. With this information, the Client has discovered that the Server supports Multipath TCP natively. This will enable the Client to bypass the Transport Converter for the subsequent Multipath TCP connections that it will initiate towards this Server.
3.4. Sample Example of Incoming Converter-Assisted Multipath TCP Connection

An example of an incoming Converter-assisted Multipath TCP connection is depicted in Figure 8. In order to support incoming connections from remote hosts, the Client may use PCP [RFC6887] to instruct the Transport Converter to create dynamic mappings. Those mappings will be used by the Transport Converter to intercept an incoming TCP connection destined to the Client and convert it into a Multipath TCP connection.

Typically, the Client sends a PCP request to the Converter asking to create an explicit TCP mapping for (internal IP address, internal port number). The Converter accepts the request by creating a TCP mapping (internal IP address, internal port number, external IP address, external port number). The external IP address and external port number will be then advertised using an out-of-band mechanism so that remote hosts can initiate TCP connections to the Client via the Converter. Note that the external and internal information may be the same.

Then, when the Converter receives an incoming SYN, it checks its mapping table to verify if there is an active mapping matching the destination IP address and destination port of that SYN. If an entry is found, the Converter inserts an MP_CAPABLE option and Connect TLV in the SYN packet, rewrites the source IP address to one of its IP addresses and, eventually, the destination IP address and port number in accordance with the information stored in the mapping. SYN-ACK and ACK will be then exchanged between the Client and the Converter.
to confirm the establishment of the initial subflow. The Client can add new subflows following normal Multipath TCP procedures.

Figure 8: Establishment of an Incoming TCP Connection through a Transport Converter

It is out of scope of this document to define specific Convert TLVs to manage incoming connections. These TLVs can be defined in a separate document.

4. The Convert Protocol (Convert)

This section describes the messages that are exchanged between a Client and a Transport Converter. The Convert Protocol (Convert, for short) uses a 32 bits long fixed header that is sent by both the Client and the Transport Converter over each established connection. This header indicates both the version of the protocol used and the length of the Convert message.

4.1. The Convert Fixed Header

The Fixed Header is used to convey information about the version and length of the messages exchanged between the Client and the Transport Converter.

The Client and the Transport Converter MUST send the fixed-sized header, shown in Figure 9, as the first four bytes of the bytestream.
The Version is encoded as an 8 bits unsigned integer value. This document specifies version 1. Version 0 is reserved by this document and MUST NOT be used.

The Total Length is the number of 32 bits word, including the header, of the bytestream that are consumed by the Convert messages. Since Total Length is also an 8 bits unsigned integer, those messages cannot consume more than 1020 bytes of data. This limits the number of bytes that a Transport Converter needs to process. A Total Length of zero is invalid and the connection MUST be reset upon reception of a header with such total length.

The Unassigned field MUST be set to zero in this version of the protocol. These bits are available for future use [RFC8126].

Data added by the Convert protocol to the TCP bytestream in the upstream connection is unambiguously distinguished from payload data in the downstream connection by the Total Length field in the Convert messages.

4.2. Convert TLVs

4.2.1. Generic Convert TLV Format

The Convert protocol uses variable length messages that are encoded using the generic TLV (Type, Length, Value) format depicted in Figure 10.

The length of all TLVs used by the Convert protocol is always a multiple of four bytes. All TLVs are aligned on 32 bits boundaries. All TLV fields are encoded using the network byte order.

Figure 9: The Fixed-Sized Header of the Convert Protocol

<table>
<thead>
<tr>
<th>Version</th>
<th>Total Length</th>
<th>Unassigned</th>
</tr>
</thead>
</table>

+---------------+---------------+-------------------------------+
| Version       | Total Length  | Unassigned                   |
|---------------+---------------+-------------------------------+ 
The Length field is expressed in units of 32 bits words. In general zero padding MUST be added if the value’s length in bytes can not be expressed as $2+(4 \times n)$.

A given TLV MUST only appear once on a connection. If two or more instances of the same TLV are exchanged over a Convert connection, the associated TCP connections MUST be closed.

4.2.2. Summary of Supported Convert TLVs

This document specifies the following Convert TLVs:

<table>
<thead>
<tr>
<th>Type</th>
<th>Hex</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x1</td>
<td>1</td>
<td>Info TLV</td>
</tr>
<tr>
<td>10</td>
<td>0xA</td>
<td>Variable</td>
<td>Connect TLV</td>
</tr>
<tr>
<td>20</td>
<td>0x14</td>
<td>Variable</td>
<td>Extended TCP Header TLV</td>
</tr>
<tr>
<td>21</td>
<td>0x15</td>
<td>Variable</td>
<td>Supported TCP Extensions TLV</td>
</tr>
<tr>
<td>22</td>
<td>0x16</td>
<td>Variable</td>
<td>Cookie TLV</td>
</tr>
<tr>
<td>30</td>
<td>0x1E</td>
<td>Variable</td>
<td>Error TLV</td>
</tr>
</tbody>
</table>

Figure 11: The TLVs used by the Convert Protocol

Type 0x0 is a reserved valued. Implementations MUST discard messages with such TLV.

The Client can request the establishment of connections to servers by using the Connect TLV (Section 4.2.5). If the connection can be established with the final server, the Transport Converter replies with the Extended TCP Header TLV (Section 4.2.4). If not, the Transport Converter returns an Error TLV (Section 4.2.8) and then closes the connection.

As a general rule, when an error is encountered an Error TLV with the appropriate error code MUST be returned by the Transport Converter.
4.2.3. The Info TLV

The Info TLV (Figure 12) is an optional TLV which can be sent by a Client to request the TCP extensions that are supported by a Transport Converter. It is typically sent on the first connection that a Client establishes with a Transport Converter to learn its capabilities. Assuming a Client is entitled to invoke the Transport Converter, the latter replies with the Supported TCP Extensions TLV described in Section 4.2.4.

![Figure 12: The Info TLV](image)

4.2.4. Supported TCP Extensions TLV

The Supported TCP Extensions TLV (Figure 13) is used by a Transport Converter to announce the TCP options for which it provides a conversion service. A Transport Converter SHOULD include in this list the TCP options that it accepts from Clients and that it includes the SYN packets that it sends to initiate connections.

Each supported TCP option is encoded with its TCP option Kind listed in the "TCP Parameters" registry maintained by IANA.

![Figure 13: The Supported TCP Extensions TLV](image)

TCP option Kinds 0, 1, and 2 defined in [RFC0793] are supported by all TCP implementations and thus MUST NOT appear in this list.

The list of Supported TCP Extension is padded with 0 to end on a 32 bits boundary.
For example, if the Transport Converter supports Multipath TCP, Kind=30 will be present in the Supported TCP Extensions TLV that it returns in response to Info TLV.

4.2.5. Connect TLV

The Connect TLV (Figure 14) is used to request the establishment of a connection via a Transport Converter. This connection can be from or to a client.

The `Remote Peer Port` and `Remote Peer IP Address` fields contain the destination port number and IP address of the Server, for outgoing connections. For incoming connections destined to a Client serviced via a Transport Converter, these fields convey the source port number and IP address.

The Remote Peer IP Address MUST be encoded as an IPv6 address. IPv4 addresses MUST be encoded using the IPv4-Mapped IPv6 Address format defined in [RFC4291]. Further, Remote Peer IP address field MUST NOT include multicast, broadcast, and host loopback addresses [RFC6890]. Connect TLVs with such messages MUST be discarded by the Transport Converter.

We distinguish two types of Connect TLV based on their length: (1) the base Connect TLV has a length of 20 bytes and contains a remote address and a remote port, (2) the extended Connect TLV spans more than 20 bytes and also includes the optional `TCP Options` field. This field is used to specify how specific TCP options should be advertised by the Transport Converter to the server.

```
+---------------+---------------+-------------------------------+
|     Type=0xA  |     Length    |      Remote Peer Port         |
|                                                               |
|         Remote Peer IP Address (128 bits)                     |
|                                                               |
|                                                               |
|---------------------------------------------------------------+
|                          TCP Options (Variable)               |
|                              ...                              |
|---------------------------------------------------------------+
```

Figure 14: The Connect TLV

The `TCP Options` field is a variable length field that carries a list of TCP option fields (Figure 15). Each TCP option field is
encoded as a block of 2+n bytes where the first byte is the TCP option Kind and the second byte is the length of the TCP option as specified in [RFC0793]. The minimum value for the TCP option Length is 2. The TCP options that do not include a length subfield, i.e., option types 0 (EOL) and 1 (NOP) defined in [RFC0793] MUST NOT be placed inside the TCP options field of the Connect TLV. The optional Value field contains the variable-length part of the TCP option. A length of two indicates the absence of the Value field. The TCP options field always ends on a 32 bits boundary after being padded with zeros.

```
+---------------+---------------+---------------+---------------+
|  TCPOpt kind  | TCPOpt Length | Value  (opt)  |  ....         |
+---------------+---------------+---------------+---------------+
```

Figure 15: The TCP Options Field

Upon reception of a Connect TLV, and absent any policy (e.g., rate-limit) or resource exhaustion conditions, a Transport Converter attempts to establish a connection to the address and port that it contains. The Transport Converter MUST use by default the TCP options that correspond to its local policy to establish this connection. These are the options that it advertises in the Supported TCP Extensions TLV.

Upon reception of an extended Connect TLV, and absent any rate limit policy or resource exhaustion conditions, a Transport Converter MUST attempt to establish a connection to the address and port that it contains. It MUST include the options of the ‘TCP Options’ subfield in the SYN sent to the Server in addition to the TCP options that it would have used according to its local policies. For the TCP options that are listed without an optional value, the Transport Converter MUST generate its own value. For the TCP options that are included in the ‘TCP Options’ field with an optional value, it MUST copy the entire option for use in the connection with the destination peer. This feature is required to support TCP Fast Open.

The Transport Converter may discard a Connect TLV request for various reasons (e.g., authorization failed, out of resources, invalid address type). An error message indicating the encountered error is returned to the requesting Client (Section 4.2.8). In order to
prevent denial-of-service attacks, error messages sent to a Client SHOULD be rate-limited.

4.2.6. Extended TCP Header TLV

The Extended TCP Header TLV (Figure 16) is used by the Transport Converter to send to the Client the extended TCP header that was returned by the Server in the SYN+ACK packet. This TLV is only sent if the Client sent a Connect TLV to request the establishment of a connection.

```
+---------------+---------------+-------------------------------+
|     Type=0x14 |     Length    |           Unassigned          |
+---------------+---------------+-------------------------------+
|               Returned Extended TCP header                    |
|                              ...                              |
+---------------------------------------------------------------+
```

Figure 16: The Extended TCP Header TLV

The Returned Extended TCP header field is a copy of the extended header that was received in the SYN+ACK by the Transport Converter.

The Unassigned field MUST be set to zero by the transmitter and ignored by the receiver. These bits are available for future use [RFC8126].

4.2.7. The Cookie TLV

The Cookie TLV (Figure 17 is an optional TLV which use is similar to the TCP Fast Open Cookie [RFC7413]. A Transport Converter may want to verify that its Clients can receive the packets that it sends to prevent attacks from spoofed addresses. This verification can be done by using a Cookie that is bound to, for example, the IP address(es) of the Client. This Cookie can be configured on the Client by means that are outside of this document or provided by the Transport Converter as follows.

A Transport Converter that has been configured to use the optional Cookie TLV MUST verify the presence of this TLV in the payload of the received SYN. If this TLV is present, the Transport Converter MUST validate the Cookie by means similar to those in Section 4.1.2 of [RFC7413] (i.e., IsCookieValid). If the Cookie is valid, the connection establishment procedure can continue. Otherwise, the Transport Converter MUST return an Error TLV set to "Not Authorized" and close the connection.
If the received SYN did not contain a Cookie TLV, and cookie validation is required, the Transport Converter should compute a Cookie bound to this Client address and return a Convert message containing the fixed header, an Error TLV set to "Missing Cookie" and the computed Cookie and close the connection. The Client will react to this error by storing the received Cookie in its cache and attempt to reestablish a new connection to the Transport Converter that includes the Cookie.

The format of the Cookie TLV is shown in the below figure.

```
+---------------+---------------+-------------------------------+
|     Type=0x16 |     Length    |             Zero              |
|               |               |-------------------------------|
|                        Opaque  Cookie                         |
|                              ...                              |
+---------------------------------------------------------------+
```

Figure 17: The Cookie TLV

4.2.8. Error TLV

The Error TLV (Figure 18) is used by the Transport Converter to provide information about some errors that occurred during the processing of Convert message. This TLV has a variable length. It appears after the Convert fixed-header in the bytestream returned by the Transport Converter. Upon reception of an Error TLV, a Client MUST close the associated connection.

```
+---------------+---------------+----------------+--------------+
|     Type=0x1E |     Length    |    Error code  |  Value       |
|               |               |----------------+--------------|
+---------------------------------------------------------------+
```

Figure 18: The Error TLV

Different types of errors can occur while processing Convert messages. Each error is identified by an Error code represented as an unsigned integer. Four classes of Error codes are defined:

- Message validation and processing errors (0-31 range): returned upon reception of an invalid message (including valid messages but with invalid or unknown TLVs).
o Client-side errors (32-63 range): the Client sent a request that could not be accepted by the Transport Converter (e.g., unsupported operation).

o Converter-side errors (64-95 range): problems encountered on the Transport Converter (e.g., lack of resources) which prevent it from fulfilling the Client’s request.

o Errors caused by the destination server (96-127 range): the final destination could not be reached or it replied with a reset.

The following error codes are defined in this document:

o Unsupported Version (0): The version number indicated in the fixed header of a message received from a peer is not supported.

This error code MUST be generated by a Transport Converter when it receives a request having a version number that it does not support.

The value field MUST be set to the version supported by the Transport Converter. When multiple versions are supported by the Transport Converter, it includes the list of supported version in the value field; each version is encoded in 8 bits. The list of supported versions should be padded with zeros to end on a 32 bits boundary.

Upon receipt of this error code, the client checks whether it supports one of the versions returned by the Transport Converter. The highest common supported version MUST be used by the client in subsequent exchanges with the Transport Converter.

o Malformed Message (1): This error code is sent to indicate that a message can not be successfully parsed and validated.

Typically, this error code is sent by the Transport Converter if it receives a Connect TLV enclosing a multicast, broadcast, or loopback IP address.

To ease troubleshooting, the value field MUST echo the received message shifted by one byte to keep to original alignment of the message.

o Unsupported Message (2): This error code is sent to indicate that a message type is not supported by the Transport Converter.
To ease troubleshooting, the value field MUST echo the received message shifted by one byte to keep to original alignment of the message.

- **Missing Cookie (3):** If a Transport Converter requires the utilization of Cookies to prevent spoofing attacks and a Cookie TLV was not included in the Convert message, the Transport Converter MUST return this error to the requesting client. The first byte of the value field MUST be set to zero and the remaining bytes of the Error TLV contain the Cookie computed by the Transport Converter for this Client.

  A Client which receives this error code MUST cache the received Cookie and include it in subsequent Convert messages sent to that Transport Converter.

- **Not Authorized (32):** This error code indicates that the Transport Converter refused to create a connection because of a lack of authorization (e.g., administratively prohibited, authorization failure, invalid Cookie TLV, etc.). The Value field MUST be set to zero.

  This error code MUST be sent by the Transport Converter when a request cannot be successfully processed because the authorization failed.

- **Unsupported TCP Option (33):** A TCP option that the Client requested to advertise to the final Server cannot be safely used.

  The Value field is set to the type of the unsupported TCP option. If several unsupported TCP options were specified in the Connect TLV, then the list of unsupported TCP options is returned. The list of unsupported TCP options MUST be padded with zeros to end on a 32 bits boundary.

- **Resource Exceeded (64):** This error indicates that the Transport Converter does not have enough resources to perform the request.

  This error MUST be sent by the Transport Converter when it does not have sufficient resources to handle a new connection. The Transport Converter may indicate in the Value field the suggested delay (in seconds) that the Client SHOULD wait before soliciting the Transport Converter for a new proxied connection. A Value of zero corresponds to a default delay of at least 30 seconds.

- **Network Failure (65):** This error indicates that the Transport Converter is experiencing a network failure to relay the request.
The Transport Converter MUST send this error code when it experiences forwarding issues to relay a connection. The Transport Converter may indicate in the Value field the suggested delay (in seconds) that the Client SHOULD wait before soliciting the Transport Converter for a new proxied connection. A Value of zero corresponds to a default delay of at least 30 seconds.

- **Connection Reset (96):** This error indicates that the final destination responded with a RST packet. The Value field MUST be set to zero.

- **Destination Unreachable (97):** This error indicates that an ICMP destination unreachable, port unreachable, or network unreachable was received by the Transport Converter. The Value field MUST echo the Code field of the received ICMP message.

Figure 19 summarizes the different error codes.

```
+-------+-------+-----------------------------------------------+
<table>
<thead>
<tr>
<th>Error</th>
<th>Hex</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x00</td>
<td>Unsupported Version</td>
</tr>
<tr>
<td>1</td>
<td>0x01</td>
<td>Malformed Message</td>
</tr>
<tr>
<td>2</td>
<td>0x02</td>
<td>Unsupported Message</td>
</tr>
<tr>
<td>3</td>
<td>0x03</td>
<td>Missing Cookie</td>
</tr>
<tr>
<td>32</td>
<td>0x20</td>
<td>Not Authorized</td>
</tr>
<tr>
<td>33</td>
<td>0x21</td>
<td>Unsupported TCP Option</td>
</tr>
<tr>
<td>64</td>
<td>0x40</td>
<td>Resource Exceeded</td>
</tr>
<tr>
<td>65</td>
<td>0x41</td>
<td>Network Failure</td>
</tr>
<tr>
<td>96</td>
<td>0x60</td>
<td>Connection Reset</td>
</tr>
<tr>
<td>97</td>
<td>0x61</td>
<td>Destination Unreachable</td>
</tr>
</tbody>
</table>
```

Figure 19: Convert Error Values

5. Compatibility of Specific TCP Options with the Conversion Service

In this section, we discuss how several standard track TCP options can be supported through the Convert protocol. The non-standard track options and the experimental options will be discussed in other documents.

5.1. Base TCP Options

Three TCP options were initially defined in [RFC0793]: End-of-Option List (Kind=0), No-Operation (Kind=1) and Maximum Segment Size (Kind=2). The first two options are mainly used to pad the TCP header. There is no reason for a client to request a Transport
Converter to specifically send these options towards the final destination.

The Maximum Segment Size option (Kind=2) is used by a host to indicate the largest segment that it can receive over each connection. This value is function of the stack that terminates the TCP connection. There is no reason for a Client to request a Transport Converter to advertise a specific MSS value to a remote server.

A Transport Converter MUST ignore options with Kind=0, 1 or 2 if they appear in a Connect TLV. It MUST NOT announce them in a Supported TCP Extensions TLV.

5.2. Window Scale (WS)

The Window Scale option (Kind=3) is defined in [RFC7323]. As for the MSS option, the window scale factor that is used for a connection strongly depends on the TCP stack that handles the connection. When a Transport Converter opens a TCP connection towards a remote server on behalf of a Client, it SHOULD use a WS option with a scaling factor that corresponds to the configuration of its stack. A local configuration MAY allow for WS option in the proxied message to be function of the scaling factor of the incoming connection.

There is no benefit from a deployment viewpoint in enabling a Client of a Transport Converter to specifically request the utilisation of the WS option (Kind=3) with a specific scaling factor towards a remote Server. For this reason, a Transport Converter MUST ignore option Kind=3 if it appears in a Connect TLV. It MUST NOT announce it in a Supported TCP Extensions TLV.

5.3. Selective Acknowledgements

Two distinct TCP options were defined to support selective acknowledgements in [RFC2018]. This first one, SACK Permitted (Kind=4), is used to negotiate the utilisation of selective acknowledgements during the three-way handshake. The second one, SACK (Kind=5), carries the selective acknowledgements inside regular segments.

The SACK Permitted option (Kind=4) MAY be advertised by a Transport Converter in the Supported TCP Extensions TLV. Clients connected to this Transport Converter MAY include the SACK Permitted option in the Connect TLV.

The SACK option (Kind=5) cannot be used during the three-way handshake. For this reason, a Transport Converter MUST ignore option
Kind=5 if it appears in a Connect TLV. It MUST NOT announce it in a TCP Supported Extensions TLV.

5.4. Timestamp

The Timestamp option was initially defined in [RFC1323] and later refined in [RFC7323]. It can be used during the three-way handshake to negotiate the utilization of timestamps during the TCP connection. It is notably used to improve round-trip-time estimations and to provide protection against wrapped sequence numbers (PAWS). As for the WS option, the timestamps are a property of a connection and there is limited benefit in enabling a client to request a Transport Converter to use the timestamp option when establishing a connection to a remote server. Furthermore, the timestamps that are used by TCP stacks are specific to each stack and there is no benefit in enabling a client to specify the timestamp value that a Transport Converter could use to establish a connection to a remote server.

A Transport Converter MAY advertise the Timestamp option (Kind=8) in the TCP Supported Extensions TLV. The clients connected to this Transport Converter MAY include the Timestamp option in the Connect TLV but without any timestamp.

5.5. Multipath TCP

The Multipath TCP options are defined in [RFC6824]. [RFC6824] defines one variable length TCP option (Kind=30) that includes a subtype field to support several Multipath TCP options. There are several operational use cases where clients would like to use Multipath TCP through a Transport Converter [IETFJ16]. However, none of these use cases require the Client to specify the content of the Multipath TCP option that the Transport Converter should send to a remote server.

A Transport Converter which supports Multipath TCP conversion service MUST advertise the Multipath TCP option (Kind=30) in the Supported TCP Extensions TLV. Clients serviced by this Transport Converter may include the Multipath TCP option in the Connect TLV but without any content.

5.6. TCP Fast Open

The TCP Fast Open cookie option (Kind=34) is defined in [RFC7413]. There are two different usages of this option that need to be supported by Transport Converters. The first utilization of the TCP Fast Open cookie option is to request a cookie from the server. In this case, the option is sent with an empty cookie by the client and the server returns the cookie. The second utilization of the TCP
Fast Open cookie option is to send a cookie to the server. In this case, the option contains a cookie.

A Transport Converter MAY advertise the TCP Fast Open cookie option (Kind=34) in the Supported TCP Extensions TLV. If a Transport Converter has advertised the support for TCP Fast Open in its Supported TCP Extensions TLV, it needs to be able to process two types of Connect TLV. If such a Transport Converter receives a Connect TLV with the TCP Fast Open cookie option that does not contain a cookie, it MUST add an empty TCP Fast Open cookie option in the SYN sent to the remote server. If such a Transport Converter receives a Connect TLV with the TCP Fast Open cookie option that contains a cookie, it MUST copy the TCP Fast Open cookie option in the SYN sent to the remote server.

5.7. TCP User Timeout

The TCP User Timeout option is defined in [RFC5482]. The associated TCP option (Kind=28) does not appear to be widely deployed.

5.8. TCP-AO

TCP-AO [RFC5925] provides a technique to authenticate all the packets exchanged over a TCP connection. Given the nature of this extension, it is unlikely that the applications that require their packets to be authenticated end-to-end would want their connections to pass through a converter. For this reason, we do not recommend the support of the TCP-AO option by Transport Converters. The only use cases where it could make sense to combine TCP-AO and the solution in this document are those where the TCP-AO-NAT extension [RFC6978] is in use.

A Transport Converter MUST NOT advertise the TCP-AO option (Kind=29) in the Supported TCP Extensions TLV. If a Transport Converter receives a Connect TLV that contains the TCP-AO option, it MUST reject the establishment of the connection with error code set to "Unsupported TCP Option", except if the TCP-AO-NAT option is used.

5.9. TCP Experimental Options

The TCP Experimental options are defined in [RFC4727]. Given the variety of semantics for these options and their experimental nature, it is impossible to discuss them in details in this document.

6. Interactions with Middleboxes

The Convert Protocol is designed to be used in networks that do not contain middleboxes that interfere with TCP. Under such conditions, it is assumed that the network provider ensures that all involved on-
path nodes are not breaking TCP signals (e.g., strip TCP options, discard some SYN, etc.).

Nevertheless, and in order to allow for a robust service, this section describes how a Client can detect middlebox interference and stop using the Transport Converter affected by this interference.

Internet measurements [IMC11] have shown that middleboxes can affect the deployment of TCP extensions. In this section, we only discuss the middleboxes that modify SYN and SYN+ACK packets since the Convert Protocol places its messages in such packets.

Consider a middlebox that removes the SYN payload. The Client can detect this problem by looking at the acknowledgement number field of the SYN+ACK returned by the Transport Converter. The Client MUST stop to use this Transport Converter given the middlebox interference.

As explained in [RFC7413], some CGNs (Carrier Grade NATs) can affect the operation of TFO if they assign different IP addresses to the same end host. Such CGNs could affect the operation of the TFO Option used by the Convert Protocol. As a reminder CGNs, enabled on the path between a Client and a Transport Converter, must adhere to the address preservation defined in [RFC6888]. See also the discussion in Section 7.1 of [RFC7413].

7. Security Considerations

7.1. Privacy & Ingress Filtering

The Transport Converter may have access to privacy-related information (e.g., subscriber credentials). The Transport Converter is designed to not leak such sensitive information outside a local domain.

Given its function and its location in the network, a Transport Converter has access to the payload of all the packets that it processes. As such, it MUST be protected as a core IP router (e.g., [RFC1812]).

Furthermore, ingress filtering policies MUST be enforced at the network boundaries [RFC2827].

This document assumes that all network attachments are managed by the same administrative entity. Therefore, enforcing anti-spoofing filters at these network ensures that hosts are not sending traffic with spoofed source IP addresses.
7.2. Authorization

The Convert Protocol is intended to be used in managed networks where end hosts can be identified by their IP address.

Stronger mutual authentication schemes MUST be defined to use the Convert Protocol in more open network environments. One possibility is to use TLS to perform mutual authentication between the client and the Converter. That is, use TLS when a Client retrieves a Cookie from the Converter and rely on certificate-based client authentication, pre-shared key based [RFC4279] or raw public key based client authentication [RFC7250] to secure this connection.

If the authentication succeeds, the Converter returns a cookie to the Client. Subsequent Connect messages will be authorized as a function of the content of the Cookie TLV.

In deployments where network-assisted connections are not allowed between hosts of a domain (i.e., hairpinning), the Converter may be instructed to discard such connections. Hairpinned connections are thus rejected by the Transport Converter by returning an Error TLV set to "Not Authorized". Absent explicit configuration otherwise, hairpinning is enabled by the Converter (see Figure 20.

Note: X2':x2' may be equal to X2:x2

Figure 20: Hairpinning Example

See below for authorization considerations that are specific for Multipath TCP.
7.3. Denial of Service

Another possible risk is the amplification attacks since a Transport Converter sends a SYN towards a remote Server upon reception of a SYN from a Client. This could lead to amplification attacks if the SYN sent by the Transport Converter were larger than the SYN received from the Client or if the Transport Converter retransmits the SYN. To mitigate such attacks, the Transport Converter SHOULD rate limit the number of pending requests for a given Client. It SHOULD also avoid sending to remote Servers SYNs that are significantly longer than the SYN received from the Client. Finally, the Transport Converter SHOULD only retransmit a SYN to a Server after having received a retransmitted SYN from the corresponding Client. Means to protect against SYN flooding attacks MUST also be enabled [RFC4987].

7.4. Traffic Theft

Traffic theft is a risk if an illegitimate Converter is inserted in the path. Indeed, inserting an illegitimate Converter in the forwarding path allows traffic interception and can therefore provide access to sensitive data issued by or destined to a host. Converter discovery and configuration are out of scope of this document.

7.5. Multipath TCP-specific Considerations

Multipath TCP-related security threats are discussed in [RFC6181] and [RFC6824].

The operator that manages the various network attachments (including the Transport Converters) can enforce authentication and authorization policies using appropriate mechanisms. For example, a non-exhaustive list of methods to achieve authorization is provided hereafter:

- The network provider may enforce a policy based on the International Mobile Subscriber Identity (IMSI) to verify that a user is allowed to benefit from the aggregation service. If that authorization fails, the Packet Data Protocol (PDP) context/bearer will not be mounted. This method does not require any interaction with the Transport Converter.

- The network provider may enforce a policy based upon Access Control Lists (ACLs), e.g., at a Broadband Network Gateway (BNG) to control the hosts that are authorized to communicate with a Transport Converter. These ACLs may be installed as a result of RADIUS exchanges, e.g., [I-D.boucadair-radext-tcpm-converter]. This method does not require any interaction with the Transport Converter.
A device that embeds a Transport Converter may also host a RADIUS client that will solicit an AAA server to check whether connections received from a given source IP address are authorized or not [I-D.boucadair-radext-tcpm-converter].

A first safeguard against the misuse of Transport Converter resources by illegitimate users (e.g., users with access networks that are not managed by the same provider that operates the Transport Converter) is the Transport Converter to reject Multipath TCP connections received on its Internet-facing interfaces. Only Multipath TCP connections received on the customer-facing interfaces of a Transport Converter will be accepted.

8. IANA Considerations

8.1. Convert Service Port Number

IANA is requested to assign a TCP port number (TBA) for the Convert Protocol from the "Service Name and Transport Protocol Port Number Registry" available at https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.

8.2. The Convert Protocol (Convert) Parameters

IANA is requested to create a new "The Convert Protocol (Convert) Parameters" registry.

The following subsections detail new registries within "The Convert Protocol (Convert) Parameters" registry.

8.2.1. Convert Versions

IANA is requested to create the "Convert versions" sub-registry. New values are assigned via IETF Review (Section 4.8 of [RFC8126]).

The initial values to be assigned at the creation of the registry are as follows:

<table>
<thead>
<tr>
<th>Version</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved by this document</td>
<td>[This-RFC]</td>
</tr>
<tr>
<td>1</td>
<td>Assigned by this document</td>
<td>[This-RFC]</td>
</tr>
</tbody>
</table>
8.2.2. Convert TLVs

IANA is requested to create the "Convert TLVs" sub-registry. The procedure for assigning values from this registry is as follows:

- The values in the range 1-127 can be assigned via IETF Review.
- The values in the range 128-191 can be assigned via Specification Required.
- The values in the range 192-255 can be assigned for Private Use.

The initial values to be assigned at the creation of the registry are as follows:

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
<td>[This-RFC]</td>
</tr>
<tr>
<td>1</td>
<td>Info TLV</td>
<td>[This-RFC]</td>
</tr>
<tr>
<td>10</td>
<td>Connect TLV</td>
<td>[This-RFC]</td>
</tr>
<tr>
<td>20</td>
<td>Extended TCP Header TLV</td>
<td>[This-RFC]</td>
</tr>
<tr>
<td>21</td>
<td>Supported TCP Extension TLV</td>
<td>[This-RFC]</td>
</tr>
<tr>
<td>22</td>
<td>Cookie TLV</td>
<td>[This-RFC]</td>
</tr>
<tr>
<td>30</td>
<td>Error TLV</td>
<td>[This-RFC]</td>
</tr>
</tbody>
</table>

8.2.3. Convert Error Messages

IANA is requested to create the "Convert Errors" sub-registry. Codes in this registry are assigned as a function of the error type. Four types are defined; the following ranges are reserved for each of these types:

- Message validation and processing errors: 0-31
- Client-side errors: 32-63
- Transport Converter-side errors: 64-95
- Errors caused by destination server: 96-127

The procedure for assigning values from this sub-registry is as follows:

- 0-191: Values in this range are assigned via IETF Review.
192-255: Values in this range are assigned via Specification Required.

The initial values to be assigned at the creation of the registry are as follows:

<table>
<thead>
<tr>
<th>Error</th>
<th>Hex</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x00</td>
<td>Unsupported Version</td>
<td>[This-RFC]</td>
</tr>
<tr>
<td>1</td>
<td>0x01</td>
<td>Malformed Message</td>
<td>[This-RFC]</td>
</tr>
<tr>
<td>2</td>
<td>0x02</td>
<td>Unsupported Message</td>
<td>[This-RFC]</td>
</tr>
<tr>
<td>3</td>
<td>0x03</td>
<td>Missing Cookie</td>
<td>[This-RFC]</td>
</tr>
<tr>
<td>32</td>
<td>0x20</td>
<td>Not Authorized</td>
<td>[This-RFC]</td>
</tr>
<tr>
<td>33</td>
<td>0x21</td>
<td>Unsupported TCP Option</td>
<td>[This-RFC]</td>
</tr>
<tr>
<td>64</td>
<td>0x40</td>
<td>Resource Exceeded</td>
<td>[This-RFC]</td>
</tr>
<tr>
<td>65</td>
<td>0x41</td>
<td>Network Failure</td>
<td>[This-RFC]</td>
</tr>
<tr>
<td>96</td>
<td>0x60</td>
<td>Connection Reset</td>
<td>[This-RFC]</td>
</tr>
<tr>
<td>97</td>
<td>0x61</td>
<td>Destination Unreachable</td>
<td>[This-RFC]</td>
</tr>
</tbody>
</table>

Figure 21: The Convert Error Codes

9. Acknowledgements

Although they could disagree with the contents of the document, we would like to thank Joe Touch and Juliusz Chroboczek whose comments on the MPTCP mailing list have forced us to reconsider the design of the solution several times.

We would like to thank Raphael Bauduin, Stefano Secci, Anandatirtha Nandugudi and Gregory Vander Schueren for their help in preparing this document. Nandini Ganesh provided valuable feedback about the handling of TFO and the error codes. Thanks to them.

Thanks to Yuchung Cheng and Praveen Balasubramanian for the discussion on supplying data in SYN.

This document builds upon earlier documents that proposed various forms of Multipath TCP proxies [I-D.boucadair-mptcp-plain-mode], [I-D.peirens-mptcp-transparent] and [HotMiddlebox13b].

From [I-D.boucadair-mptcp-plain-mode]:

Many thanks to Chi Dung Phung, Mingui Zhang, Rao Shoaib, Yoshifumi Nishida, and Christoph Paasch for their valuable comments.
Thanks to Ian Farrer, Mikael Abrahamsson, Alan Ford, Dan Wing, and Sri Gundavelli for the fruitful discussions in IETF#95 (Buenos Aires).

Special thanks to Pierrick Seite, Yannick Le Goff, Fred Klamm, and Xavier Grall for their inputs.

Thanks also to Olaf Schleusing, Martin Gysi, Thomas Zasowski, Andreas Burkhard, Silka Simmen, Sandro Berger, Michael Melloul, Jean-Yves Flahaut, Adrien Desportes, Gregory Detal, Benjamin David, Arun Srinivasan, and Raghavendra Mallya for the discussion.

9.1. Contributors

Bart Peirens contributed to an early version of the document.

As noted above, this document builds on two previous documents.

The authors of [I-D.boucadair-mptcp-plain-mode] were:

- Mohamed Boucadair
- Christian Jacquenet
- Olivier Bonaventure
- Denis Behaghel
- Stefano Secci
- Wim Henderickx
- Robert Skog
- Suresh Vinapamula
- SungHoon Seo
- Wouter Cloetens
- Ullrich Meyer
- Luis M. Contreras
- Bart Peirens

The authors of [I-D.peirens-mptcp-transparent] were:
10. Change Log

This section to be removed before publication.

- 00: initial version, designed to support Multipath TCP and TFO only
- 00 to -01: added section Section 5 describing the support of different standard tracks TCP options by Transport Converters, clarification of the IANA section, moved the SOCKS comparison to the appendix and various minor modifications
- 01 to -02: Minor modifications
- 02 to -03: Minor modifications
- 03 to -04: Minor modifications
- 04 to -05: Integrate a lot of feedback from implementors who have worked on client and server side implementations. The main modifications are the following:
  * TCP Fast Open is not strictly required anymore. Several implementors expressed concerns about this requirement. The TFO Cookie protects from some attack scenarios that affect open servers like web servers. The Convert protocol is different and as discussed in RFC7413, there are different ways to protect from such attacks. Instead of using a TFO cookie inside the TCP options, which consumes precious space in the extended TCP header, this version supports the utilisation of a Cookie that is placed in the SYN payload. This provides the same level of protection as a TFO Cookie in environments where such protection is required.
  * the Bootstrap procedure has been simplified based on feedback from implementors
  * Error messages are not included in RST segments anymore but sent in the bytestream. Implementors have indicated that
processing such segments on clients was difficult on some platforms. This change simplifies client implementations.

* Many minor editorial changes to clarify the text based on implementors feedback.

o 05 to -06: Many clarifications to integrate the comments from the chairs in preparation to the WGLC:

* Updated IANA policy to require "IETF Review" instead of "Standard Action"
* Call out explicitly that data in SYNs are relayed by the Converter
* Reiterate the scope
* Hairpinning behavior can be disabled (policy-based)
* Fix nits

o 07:

* Update the text about supplying data in SYNs to make it clear that a constraint defined in RFC793 is relaxed following the same rationale as in RFC7413.
* Nits
* Added Appendix A on example Socket API changes

o 08:

* Added short discussion on the termination of connections

11. Example Socket API Changes to Support the 0-RTT Convert Protocol

11.1. Active Open (Client Side)

On the client side, the support of the 0-RTT Converter protocol does not require any other changes than those identified in Appendix A of [RFC7413]. Those modifications are already supported by multiple TCP stacks.

As an example, on Linux, a client can send the 0-RTT Convert message inside a SYN by using sendto with the MSG_FASTOPEN flag as shown in the example below:
s = socket(AF_INET, SOCK_STREAM, 0);
sendto(s, buffer, buffer_len, MSG_FASTOPEN,
       (struct sockaddr *) &server_addr, addr_len);

The client side of the Linux TCP TFO can be used in two different
modes depending on the host configuration (sysctl tcp_fastopen
variable):

- 0x1: (client) enables sending data in the opening SYN on the
   client.
- 0x4: (client) send data in the opening SYN regardless of cookie
   availability and without a cookie option.

By setting this configuration variable to 0x5, a Linux client using
the above code would send data inside the SYN without using a TFO
option.

11.2. Passive Open (Converter Side)

The Converter needs to enable the reception of data inside the SYN
independently of the utilisation of the TFO option. This implies
that the Transport Converter application cannot rely on the TFO
cookies to validate the reachability of the IP address that sent the
SYN. It must rely on other techniques, such as the Cookie TLV
described in this document, to verify this reachability.

[RFC7413] suggested the utilisation of a TCP_FASTOPEN socket option
to enable the reception of SYNs containing data. Later, Appendix A
of [RFC7413], mentionned:

Traditionally, accept() returns only after a socket is connected.
But, for a Fast Open connection, accept() returns upon receiving
SYN with a valid Fast Open cookie and data, and the data is available
to be read through, e.g., recvmsg(), read().

To support the 0-RTT Convert protocol, this behaviour should be
modified as follows:

Traditionally, accept() returns only after a socket is connected.
But, for a Fast Open connection, accept() returns upon receiving a
SYN with data, and the data is available to be read through, e.g.,
recvmsg(), read(). The application that receives such SYNs with data
must be able to validate the reachability of the source of the SYN
and also deal with replayed SYNs.

The Linux server side can be configured with the following sysctls:
0x2: (server) enables the server support, i.e., allowing data in a SYN packet to be accepted and passed to the application before 3-way handshake finishes.

0x200: (server) accept data-in-SYN w/o any cookie option present.

However, this configuration is system-wide. This is convenient for typical Transport Converter deployments where no other applications relying on TFO are collocated on the same device.

Recently, the TCP_FASTOPEN_NO_COOKIE socket option has been added to provide the same behaviour on a per socket basis. This enables a single host to support both servers that require the TFO cookie and servers that do not use it.

12. Differences with SOCKSv5

At a first glance, the solution proposed in this document could seem similar to the SOCKS v5 protocol [RFC1928] which is used to proxy TCP connections. The Client creates a connection to a SOCKS proxy, exchanges authentication information and indicates the destination address and port of the final server. At this point, the SOCKS proxy creates a connection towards the final server and relays all data between the two proxied connections. The operation of an implementation based on SOCKSv5 is illustrated in Figure 22.
The Convert protocol also relays data between an upstream and a downstream connection, but there are important differences with SOCKSv5.

A first difference is that the Convert protocol exchanges all control information during the three-way handshake. This reduces the connection establishment delay compared to SOCKS that requires two or more round-trip-times before the establishment of the downstream connection towards the final destination. In today’s Internet, latency is a important metric and various protocols have been tuned
to reduce their latency [I-D.arkko-arch-low-latency]. A recently proposed extension to SOCKS also leverages the TFO option [I-D.olteanu-intarea-socks-6].

A second difference is that the Convert protocol explicitly takes the TCP extensions into account. By using the Convert protocol, the Client can learn whether a given TCP extension is supported by the destination Server. This enables the Client to bypass the Transport Converter when the destination supports the required TCP extension. Neither SOCKS v5 [RFC1928] nor the proposed SOCKS v6 [I-D.olteanu-intarea-socks-6] provide such a feature.

A third difference is that a Transport Converter will only accept the connection initiated by the Client provided that the downstream connection is accepted by the Server. If the Server refuses the connection establishment attempt from the Transport Converter, then the upstream connection from the Client is rejected as well. This feature is important for applications that check the availability of a Server or use the time to connect as a hint on the selection of a Server [RFC8305].

A fourth difference is that the Convert protocol only allows the client to specify the address/port of the destination server and not a DNS name. We evaluated an alternate design for the Connect TLV that included the DNS name of the remote peer instead of its IP address as in SOCKS [RFC1928]. However, that design was not adopted because it induces both an extra load and increased delays on the Transport Converter to handle and manage DNS resolution requests.

13. References

13.1. Normative References


13.2. Informative References


Peirens, B., Detal, G., Barre, S., and O. Bonaventure, "Link bonding with transparent Multipath TCP", draft-peirens-mptcp-transparent-00 (work in progress), July 2016.

Bonaventure, O. and S. Seo, "Multipath TCP Deployment", IETF Journal, Fall 2016, n.d..


[RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and
L. Jones, "SOCKS Protocol Version 5", RFC 1928,
DOI 10.17487/RFC1928, March 1996,

Selective Acknowledgment Options", RFC 2018,
DOI 10.17487/RFC2018, October 1996,

[RFC2827] Ferguson, P. and D. Senie, "Network Ingress Filtering:
Defeating Denial of Service Attacks which employ IP Source
Address Spoofing", BCP 38, RFC 2827, DOI 10.17487/RFC2827,

Shelby, "Performance Enhancing Proxies Intended to
Mitigate Link-Related Degradations", RFC 3135,
DOI 10.17487/RFC3135, June 2001,

[RFC6181] Bagnulo, M., "Threat Analysis for TCP Extensions for
Multipath Operation with Multiple Addresses", RFC 6181,
DOI 10.17487/RFC6181, March 2011,

[RFC6887] Wing, D., Ed., Cheshire, S., Boucadair, M., Penno, R., and
P. Selkirk, "Port Control Protocol (PCP)", RFC 6887,
DOI 10.17487/RFC6887, April 2013,

[RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis,
"Increasing TCP’s Initial Window", RFC 6928,
DOI 10.17487/RFC6928, April 2013,

[RFC6978] Touch, J., "A TCP Authentication Option Extension for NAT
Traversal", RFC 6978, DOI 10.17487/RFC6978, July 2013,

Scheffenegger, Ed., "TCP Extensions for High Performance",
RFC 7323, DOI 10.17487/RFC7323, September 2014,

Authors' Addresses

Olivier Bonaventure (editor)
Tessares
Email: Olivier.Bonaventure@tessares.net

Mohamed Boucadair (editor)
Orange
Email: mohamed.boucadair@orange.com
Sri Gundavelli
Cisco
Email: sgundave@cisco.com

SungHoon Seo
Korea Telecom
Email: sh.seo@kt.com

Benjamin Hesmans
Tessares
Email: Benjamin.Hesmans@tessares.net
ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control Packets
draft-ietf-tcpm-generalized-ecn-03

Abstract

This document describes an experimental modification to ECN when used with TCP. It allows the use of ECN on the following TCP packets: SYNs, pure ACKs, Window probes, FINs, RSTs and retransmissions.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 25, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction ........................................... 3
   1.1. Motivation ........................................ 4
   1.2. Experiment Goals .................................. 5
   1.3. Document Structure ................................ 6
2. Terminology ............................................. 6
3. Specification ........................................... 7
   3.1. Network (e.g. Firewall) Behaviour ................. 7
   3.2. Sender Behaviour ................................. 8
       3.2.1. SYN (Send) .................................... 9
       3.2.2. SYN-ACK (Send) ............................... 12
       3.2.3. Pure ACK (Send) ............................... 13
       3.2.4. Window Probe (Send) ......................... 14
       3.2.5. FIN (Send) ............................... 15
       3.2.6. RST (Send) ............................... 15
       3.2.7. Retransmissions (Send) ..................... 16
       3.2.8. General Fall-back for any Control Packet or Retransmission .......................... 16
   3.3. Receiver Behaviour ................................ 16
       3.3.1. Receiver Behaviour for Any TCP Control Packet or Retransmission ..................... 17
       3.3.2. SYN (Receive) ............................... 17
       3.3.3. Pure ACK (Receive) ........................... 18
       3.3.4. FIN (Receive) ............................... 18
       3.3.5. RST (Receive) ............................... 18
       3.3.6. Retransmissions (Receive) ................... 19
4. Rationale ............................................... 19
   4.1. The Reliability Argument .......................... 19
4.2. SYNs ................................................ 20
   4.2.1. Argument 1a: Unrecognized CE on the SYN ........ 20
   4.2.2. Argument 1b: ECT Considered Invalid on the SYN .... 21
   4.2.3. Caching Strategies for ECT on SYNs .............. 23
   4.2.4. Argument 2: DoS Attacks ....................... 25
4.3. SYN-ACKs ........................................... 26
   4.3.1. Possibility of Unrecognized CE on the SYN-ACK .... 26
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.

ECN++ uses a sender-only deployment model. It works whether the two ends of the TCP connection use classic ECN feedback [RFC3168] or experimental Accurate ECN feedback (AccECN [I-D.ietf-tcpm-accurate-ecn]). Nonetheless, if the client does not implement AccECN, it cannot use ECN++ on the one packet that offers most benefit from it - the initial SYN. Therefore, implementers of ECN++ are RECOMMENDED to also implement AccECN.
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 [I-D.ietf-tsvwg-l4s-arch]), 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.

Non-ECN control packets particularly harm 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 [I-D.ietf-tsvwg-l4s-arch]. In such an approach, L4S packets identify themselves using an ECN codepoint [I-D.ietf-tsvwg-ecn-l4s-id]. 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 enjoy ultra-low latency forwarding, while the packets controlling and retransmitting these data packets would
still get stuck behind the queue induced by legacy (‘Classic’) 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.

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. Whenever data input is needed to assist in a design choice, it is spelled out throughout the 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 keywords 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] when and only when they appear in all capitals.

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 [RFC0793], a window probe is a TCP segment with only one byte of data 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.

ECT: ECN-Capable Transport. One of the two codepoints ECT(0) or ECT(1) in the ECN field [RFC3168] of the IP header (v4 or v6). An ECN-capable sender sets one of these to indicate that both transport end-points support ECN. When this specification says the sender sets an ECT codepoint, by default it means ECT(0). Optionally, it could
mean ECT(1), which is in the process of being redefined for use by L4S experiments [RFC8311] [I-D.ietf-tsvwg-ecn-l4s-id].

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 allow implementers to check the receive side code while they are altering the send-side code. All changes can be deployed at each end-point 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.

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

Window probe and retransmission are abbreviated to W Probe an Re-XMT.

* For a SYN, "negotiated" means "requested".

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 the sender cannot set ECT on the SYN if it is not requesting AccECN feedback. Therefore it is RECOMMENDED that the experimental AccECN specification [I-D.ietf-tcpm-accurate-ecn] is implemented (as well as the present specification), 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 the other end of the TCP connection does not (AccECN can only be used for a connection if both ends support it).

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. It can ignore the prohibition in section 6.1.1 of RFC 3168 against setting ECT on such a SYN, as per Section 4.3 of [RFC8311].

3.2.1.1.2. ECN++ TCP Client does not Support AccECN

A TCP initiator MUST NOT set ECT on a SYN if it does not also attempt to negotiate Accurate ECN feedback in the same SYN.

If the TCP initiator does not support AccECN, the rest of Section 3.2.1 does not apply. It solely applies to the case where the TCP initiator supports AccECN as well as ECN++.

3.2.1.2. Caching where to use ECT on SYNs

As explained above, this subsection only applies if the ECN++ TCP client also supports AccECN.

Until AccECN servers become widely deployed, a TCP initiator that sets ECT on a SYN (which implies the same SYN also requests AccECN, as required 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 4.2.1).

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). The initiator will discover a server that has upgraded to support AccECN 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 4.2.1. However, Section 4.2.1 gives two other strategies and the choice between them depends on the implementer’s goals 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 also supports AccECN.

If the SYN-ACK returned to the TCP initiator confirms that the server supports AccECN, it will also indicate whether or not the SYN was CE-marked. If the SYN was CE-marked, the initiator MUST reduce its Initial Window (IW) and SHOULD reduce it to 1 SMSS (sender maximum segment size).
If the initiator has set ECT on the SYN and if the SYN-ACK shows that the server does not support AccECN, 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 a non-AccECN 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 10 (IW10 [RFC6928]) is implemented, 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 supports AccECN.

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 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. It can ignore the requirement in section 6.1.1 of RFC 3168 to
set not-ECT on a SYN-ACK, as per Section 4.3 of [RFC8311].

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.

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 the
initial window of 10 experiment (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. A client-based alternative to caching at the server is given in Section 4.3.3. 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). If a cache is used, it SHOULD be arranged to expire so that the server 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, if AccECN has been successfully negotiated, either end of the connection will set ECT on Pure ACKs. They can ignore the requirement in section 6.1.4 of RFC 3168 to set not-ECT on a pure ACK, as per Section 4.3 of [RFC8311].

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

As explained above, this subsection only applies if AccECN has been successfully negotiated for the TCP connection.

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 4.4.1.1).

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. It can ignore the prohibition in section 6.1.6 of RFC 3168 against setting ECT on a window probe, as per Section 4.3 of [RFC8311].

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 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.

3.2.7. Retransmissions (Send)

For the ECN++ experiment, the TCP sender will set ECT on retransmitted segments. It can ignore the prohibition in section 6.1.5 of RFC 3168 against setting ECT on retransmissions, as per Section 4.3 of [RFC8311].

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 where ECT on the initial packet of a half-connection (SYN or SYN-ACK) 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 this experiment to be useful, the following requirements follow from RFC8311:

- 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.

These measures are derived from the robustness principle of "... be liberal in what you accept from others", in order to ensure compatibility with 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. When RFC3168 says that "A host MUST NOT set ECT on SYN ... packets." it is silent as to what a TCP server ought to do if it receives a SYN packet with a non-zero IP/ECN field.

Some implementations of TCP servers (e.g. current Linux) 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 finds that this type of network mangling seems to be virtually non-existent so it would be preferable to report any such mangling so it can be fixed.

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

- Any TCP implementation SHOULD accept receipt of a pure ACK with a non-zero ECN field, despite current RFCs precluding the sending of such packets.

- 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]). Currently AccECN feedback is required to count CE marking of any control packet including pure ACKs. Whereas RFC 3168 is silent on this point, so feedback of CE-markings might be implementation specific (see Section 4.4.1.1).

3.3.4. FIN (Receive)

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 "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 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 intofeeding 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. First it addresses overarching 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 because TCP does not detect the loss of most 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. In particular, setting the CE codepoint on the very same packet that would otherwise have been dropped fulfills this criterion, since either the packet is delivered and the CE signal is delivered to the endpoint, or the packet is dropped and the original congestion signal (packet loss) is delivered to the endpoint.
The concern about a CE marking being dropped at a subsequent node might be motivated by the idea that ECN-marking a packet at the first node does not remove the packet, so it could go on to worsen congestion at a subsequent node. However, it is not useful to reason about congestion by considering single packets. The departure rate from the first node will generally be the same (fully utilized) with or without ECN, so this argument does not apply.

4.2. SYN

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 'ECN' bits 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 doesn’t support AccECN.

This still does not seem to allow a client 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.

Happily, the appropriate conservative congestion response is to reduce the initial window, and it is extremely rare for a TCP client to send more than one packet as its initial request anyway. 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).

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

Given, until now, ECT-marked SYN packets have been 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 was increasing 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 those sites that do not support ECN-capable SYN (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 "contra-Postel" ECN test, because it is 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 ‘contra-Postel’ test proves that RFC 5652 was correct to expect that ECT would be considered invalid on SYNs. Nonetheless, it is not an insurmountable problem - caching can work round it. The prevalence of these "contra-Postel" 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 caching strategies for setting ECT on a SYN:

(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 ‘contra-Postel’ 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.

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 ‘contra-Postel’ 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. For instance, if a user’s Internet access bottleneck supported L4S ECN but not Classic ECN, strategy (S2A) would make most sense and there would be no point trying to avoid the 'contra-Postel' test and negotiate Classic ECN.

- 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 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).

- 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 'contra-Postel' 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, caching successes (S1) starts off small then grows, while caching failures (S2B) becomes 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 than 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, it does not seem sensible to add a number of packets at once. 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.
4.3.3. Fall-Back if ECT SYN-ACK Fails

An alternative to the server caching failed connection attempts would be for the server to rely on the client caching 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 the server receives a SYN that requests AccECN support but is set to not-ECT, it replies with a SYN-ACK also set to not-ECT. 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.

4.4. Pure ACKs

Section 5.2 of RFC 3168 gives the following arguments for not allowing the ECT marking of pure ACKs (ACKs not piggy-backed on data):

"To ensure the reliable delivery of the congestion indication of the CE codepoint, an ECT codepoint MUST NOT be set in a packet unless the loss of that packet in the network would be detected by the end nodes and interpreted as an indication of congestion.

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."

Bagnulo & Briscoe Expires April 25, 2019
We next address each of the arguments presented above.

The first argument 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 unless there is a mechanism to respond to it. This argument actually comprises three sub-arguments:

Mechanism feasibility: If ECN is enabled on Pure ACKs, are there, or could there be, suitable mechanisms to detect, feed back and respond to ECN-marked Pure ACKs?

Do no extra harm: There has never been a mechanism to respond to loss of non-ECN Pure ACKs. So it seems that adding ECN without a response mechanism will do no extra harm to others, while improving a connection’s own performance (because loss of an ACK holds back new data). However, if the end systems have no response mechanism, ECN Pure ACKs do slightly more harm than non-ECN, because the AQM doesn’t immediately clear ECT packets from the queue until it reaches overload and disables ECN.

Standards policy: Even if there were no harm to others, does it set an undesirable precedent to allow a flow to use ECN to protect its Pure ACKs from loss, when there is no mechanism to respond to ECN-marking?

The last two arguments involve value judgements, but they both depend on the concrete technical question of mechanism feasibility, which will therefore be addressed first in Section 4.4.1 below. Then Section 4.4.2 draws conclusions by addressing the value judgements in the other two questions.

4.4.1. Mechanisms to Respond to CE-Marked Pure ACKs

The question of whether the receiver of pure ACKs is required to detect and feedback any CE-marking is outside the scope of the present specification – it is a matter for the relevant feedback specification (classic ECN [RFC3168] and AccECN [I-D.ietf-tcpm-accurate-ecn]). The response to congestion feedback is also out of scope, because it would be defined in the base TCP congestion control specification [RFC5681] or its variants.

Nonetheless, in order to decide whether the present ECN++ experimental specification should require a host to set ECT on pure ACKs, we only need to know whether a response mechanism would be feasible – we do not have to standardize it. So the bullets below
assess, for each type of feedback, whether the three stages of the congestion response mechanism could all work.

Detection: Can the receiver of a pure ACK detect a CE marking on it?:

* Classic feedback: RFC 3168 is silent on this point. The implementer of the receiver would not expect CE marks on pure ACKs, but the implementation might happen to check for CE marks before it looks for the data. So detection will be implementation-dependent.

* AccECN feedback: the AccECN specification requires the receiver of any TCP packets to count any CE marks on them (whether or not it sends ECN-capable control packets itself).

Feedback: TCP never ACKs a pure ACK, but the receiver of a CE-mark on a pure ACK could feed it back when it sends a subsequent data segment (if it ever does):

* Classic feedback: RFC 3168 is silent on this point, so feedback of CE-markings might be implementation specific. If the receiver (of the pure ACKs) did generate feedback, it would set the echo congestion experienced (ECE) flag in the TCP header of subsequent packets in the round, as it would to feed back CE on data packets.

* AccECN feedback: the receiver continually feeds back a count of the number of CE-marked packets that it has received and, optionally, a count of CE-marked bytes. For either metric, AccECN includes pure ACKs and indeed all types of packets.

Congestion response: In either case (classic or AccECN feedback), if the TCP sender does receive feedback about CE-markings on pure ACKs, it will be able to reduce the congestion window (cwnd) and/or the ACK rate.

Therefore a congestion response mechanism is clearly feasible if AccECN has been negotiated, but the position is unknown for the installed base of classic ECN feedback.

4.4.1.1. 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 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 it is purely receiving data and sending only Pure ACKs, reducing cwnd will have caused it no harm, having no effect on its ACK rate (the next subsection addresses that).

However, 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. For instance, one could calculate the fraction of CE-marked bytes (headers and data) over each round trip (say) as follows:

\[
\frac{\text{(CE-marked header bytes + CE-marked data bytes)}}{\text{(all header bytes + all data bytes)}}
\]

Header bytes can 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.

4.4.1.2. ACK Rate Response to CE-Marked Pure ACKs

Reducing the congestion window will have no effect on the rate of pure ACKs. The worst case here is if the bottleneck is congested solely with pure ACKs, but it could also be problematic if a large
fraction of the load was from unresponsive ACKs, leaving little or no capacity for the load from responsive data.

Since RFC 3168 was published, experimental Acknowledgement Congestion Control (AckCC) techniques have been documented in [RFC5690] (informational). So any pair of TCP end-points can choose to agree to regulate the delayed ACK ratio in response to lost or CE-marked pure ACKs. However, the protocol has a number of open issues concerning deployment (e.g. it requires support from both ends, it relies on two new TCP options, one of which is required on the SYN where option space is at a premium and, if either option is blocked by a middlebox, no fall-back behaviour is specified).

The new TCP options address two problems, namely that TCP had: i) no mechanism to allow ECT to be set on pure ACKs; and ii) no mechanism to feed back loss or CE-marking of pure ACKs. A combination of the present specification and AccECN addresses both these problems, at least for CE-marking. So it might now be possible to design an ECN-specific ACK congestion control scheme without the extra TCP options proposed in RFC 5690. However, such a mechanism is out of scope of the present document.

Setting aside the practicality 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.

4.4.2. Summary: Enabling ECN on Pure ACKs

In the case when AccECN has been negotiated, it provides a feasible congestion response mechanism, so the arguments for ECT on pure ACKs heavily 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 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 has deployment challenges. In summary, a congestion response mechanism is unlikely to be feasible with the installed base of classic ECN.

During review of this specification, it was decided that allowing hosts to set ECT on Pure ACKs without a feasible response mechanism would set an undesirable precedent. It would certainly improve the flow’s own performance, but it would slightly increase potential harm to others. Therefore, Section 3.2.3 allows ECT on Pure ACKs if AccECN feedback has been negotiated, but not with classic RFC 3168 ECN feedback.

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 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.


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 over-reaction to congestion. 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 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.

A consequence is that, for those retransmitted packets that arrive at the receiver after the original packet has been properly received (so-called spurious retransmissions), any CE marking will be ignored. There is no problem with that because the fact that the original packet has been delivered implies that the sender’s original congestion response (when it deemed the packet lost and retransmitted it) was unnecessary.

Finally, the third argument is about over-reacting to congestion. The argument goes that, if a retransmitted packet is dropped, the sender will not detect it, so it will not react again to congestion (it would have reduced its congestion window already when it retransmitted the packet). Whereas, if retransmitted packets can be CE tagged instead of dropped, senders could potentially react more than once to congestion. However, we argue that it is legitimate to respond again to congestion if it still persists in subsequent round trip(s).

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 each 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; 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 an inferior queue [I-D.ietf-tsvwg-ecn-l4s-id] or over a different path, because some
broken multipath equipment (erroneously) routes based on all 8 bits of the Diffserv field.

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 following subsections discuss any interactions between setting ECT on all packets and using the following popular variants of TCP: IW10 and TFO. It also 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.

The designs of the following TCP variants have also 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]), TCP Fast Open (TFO [RFC7413]) and L4S [I-D.ietf-tsvwg-l4s-arch].

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 can 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 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, the 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 [28; Fig 3] - IW10 is primarily exploited by TCP servers.

If a responder receives feedback that the SYN-ACK was CE-marked, Section 3.2.2.2 mandates that it reduces its initial window to 1 SMSS. When the responder also implements IW10, it is particularly important to adhere to this requirement in order to avoid overflowing a queue that is clearly already congested.

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. TCP Derivatives

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

Stream Control Transmission Protocol (SCTP [RFC4960]) is a standards track transport protocol derived from TCP. SCTP currently does not include ECN support, but Appendix A of RFC 4960 broadly describes how it would be supported and a (long-expired) draft on the addition of ECN to SCTP has been produced [I-D.stewart-tsvwg-sctpecn]. This draft avoided setting ECT on control packets and retransmissions, closely following the arguments in RFC 3168.

QUIC [I-D.ietf-quic-transport] 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(0) on all packets.

6. Security Considerations

Section 3.2.6 considers the question of whether ECT on RSTs will allow RST attacks to be intensified. 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 addressed 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.

7. IANA Considerations

There are no IANA considerations in this memo.

8. Acknowledgments

Thanks to Mirja Kuehlewind, David Black, Padma Bhooma, Gorry Fairhurst, Michael Scharf, Yuchung Cheng and Christophe Paasch for their useful reviews.

The work of Marcelo Bagnulo has been performed in the framework of the H2020-ICT-2014-2 project 5G NORMA. His contribution reflects the consortium’s view, but the consortium is not liable for any use that may be made of any of the information contained therein.
Bob Briscoe’s contribution was partly funded by the Research Council of Norway through the TimeIn project. The views expressed here are solely those of the authors.

9. References

9.1. Normative References

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


9.2. Informative References

[ecn-overload]  

[ecn-pam]  
[ECN-PLUS]

[I-D.ietf-quic-transport]

[I-D.ietf-tsvwg-ecn-l4s-id]

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

[I-D.stewart-tsvwg-sctpecn]

[judd-nsdi]

[Kuehlewind18]

[Mandalari18]


[strict-ecn] Dumazet, E., "tcp: be more strict before accepting ECN negociation", Linux netdev patch list, May 2012, <https://github.com/torvalds/linux/commit/bd14b1b2e29bd6812597f896dde06eaf7c6d2f24#diff-5c7c60ed5f9efb6bbe97ff5233f17282>.

Authors’ Addresses

Marcelo Bagnulo
Universidad Carlos III de Madrid
Av. Universidad 30
Leganes, Madrid 28911
SPAIN

Phone: 34 91 6249500
Email: marcelo@it.uc3m.es
URI: http://www.it.uc3m.es
Bob Briscoe
CableLabs
UK

Email: ietf@bobbriscoe.net
URI: http://bobbriscoe.net/
Transmission Control Protocol (TCP) YANG Model  
draft-scharf-tcpm-yang-tcp-01  

Abstract  

This document specifies a base YANG model for TCP on devices that are configured by network management protocols. The YANG model is loosely based on the standard TCP-MIB [RFC4022].  

Status of This Memo  

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

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

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

This Internet-Draft will expire on August 28, 2019.  

Copyright Notice  

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

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

The Transmission Control Protocol (TCP) [RFC0793] is used by several control and management protocols in the Internet. Therefore, TCP is implemented on network elements that can be configured via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. This document specifies a YANG data model [RFC6020][RFC7950] for configuring and operating TCP on network elements that support YANG data models.

Many protocol stacks on Internet hosts use other methods to configure TCP, such as operating system configuration or policies. Such TCP stacks typically cannot be configured by network management protocols such as NETCONF or RESTCONF. This document only applies to network elements that use YANG data models.

This document is related to other standardization efforts. A Management Information Base (MIB) for the Transmission Control Protocol (TCP) has been standardized [RFC4022], and various managed network devices support the TCP-MIB. A MIB providing extended statistics for TCP is also available [RFC4898]. In addition, there are also MIBs for UDP [RFC4113] and SCTP [RFC3873].

The document defines a YANG data model for configuration and operational state of a TCP implementation. The model focuses on fundamental and standard TCP functions. The model can be augmented to address more advanced or implementation-specific TCP features.
1.1. Why a YANG Model for TCP?

This section summarizes reasons why the specification of a YANG model for TCP could be useful to maintain TCP's utility.

As more and more network elements can be managed by management protocols such as NETCONF or RESTCONF, network elements may need a YANG model for TCP stack configuration and operation. This could in particular apply to network elements that support the TCP-MIB [RFC4022] but migrate to management by NETCONF or RESTCONF. In such cases, one option would be to translate the TCP-MIB into a YANG model, for instance using the translation described in [RFC6643]. However, such a translated YANG model neither allows configuration, nor is the content up-to-date. For instance, the TCP-MIB refers to standards that have been obsoleted.

Given the advantages of YANG, there are also many ongoing efforts to define YANG models. An increasing number of YANG models include TCP and TCP-related parameters for specific usage scenarios. Examples are:

- TCP connection properties such as use of keep-alives can be configured by [I-D.ietf-netconf-netconf-client-server].
- TCP header attributes are modeled in [I-D.ietf-netmod-acl-model].
- TCP-related configuration of a NAT is defined in [I-D.ietf-opsawg-nat-yang].

It is possible that further YANG models would include aspects of or relate to TCP stack configuration. An example could be YANG models for other TCP-based protocols running on network elements. Instead of specifying TCP configuration separately for each use cases, a better alternative may be to define a common YANG model for TCP, from which definitions and objects could be reused as needed.

The intention of this document is to explore whether a YANG model for TCP is useful, and if so, what such a model would include (and what not).

This document targets the TCMP working group as the TCMP charter includes work on "interfaces that maintain TCP’s utility". The YANG model deals with the network management interface of a TCP implementation. The interface used for data transfer between an application and the TCP stack is outside the scope of this document.
1.2. Open Issues

There are many open questions on the scope of this document that need discussion and community feedback:

- **Scope:** TCP stacks can typically be customized by configuration, including implementation-specific internal behavior. A standard YANG model should focus on fundamental TCP parameters that are independent of the internals of an implementation, that are available in all or most implementations, and that matter for network management. This set of TCP configuration parameters needs to be determined. Additional implementation-specific configuration could be added by augmentation of the YANG model and would not require standardization.

- **Backward compatibility:** There may be implementations that support the TCP-MIB [RFC4022], possibly in addition to YANG models. In such cases, one option could be to translate the TCP-MIB [RFC4022] into an equivalent YANG model. The TCP-MIB is not up-to-date. Additions could be needed to reflect e.g. the progress of TCP standards. It is an open question whether a YANG model would need "backward-compatibility" to TCP-MIB entries, and, if so, if this is needed for all TCP-MIB entries.

- **General definitions:** A TCP model could perhaps include general YANG type definitions and groupings for TCP. They could be reused by other YANG models. Having a common definition of TCP-related attributes could ensure consistency. It is an open question whether definitions in a TCP model would indeed be used by other YANG models.

- **Extended statistics:** It needs to be decided whether extended statistics [RFC4898] would be in scope of a TCP YANG model.

- **Suggested MIB additions:** Extensions to the TCP-MIB [RFC4022] have been suggested, e.g., in [RFC5482]. A YANG model could possibly take into account such proposed extensions.

- **Cross-layer functionality:** The TCP configuration may have dependencies on other protocols. It needs to be figured out if and how this can be modelled. An example could be enabling of TCP keep-alives, which should be off by default [RFC1122].
Alignment: Further discussion is needed on alignment of a TCP YANG model with other transport protocols. This would include UDP and SCTP.

Status of [RFC4022]: Standardization of a YANG model does not affect an existing MIB. As a result, updating or deprecating [RFC4022] may not be necessary even if a YANG model for TCP is standardized. However, more analysis is needed on how future versions of this I-D would relate to or reference [RFC4022].

Implementation aspects: Implementation aspects of a YANG model for TCP need further analysis.

Authoring: If there is a need for similarity to the TCP-MIB, authors and contributors to [RFC4022] could/should be added as authors or contributors to this document.

2. Requirements Language

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

3. Overview of the YANG Model for TCP

[[Editor’s node: This section should to be improved in follow-up versions of this document.]]

3.1. Model Design

[[Editor’s node: In potential follow-up versions of this document, this section will discuss the design of the TCP YANG model.]]

A standard YANG data model for TCP should follow the Network Management Datastore Architecture (NMDA) [RFC8342].

The YANG model in the -00 version of this document has been translated from the TCP-MIB [RFC4022] without modifications (apart from removing outdated meta-data). The motivation of using the standard TCP-MIB definitions as a baseline for -00 is to facilitate the tracking of potential changes in later versions of this specification, as far as this is needed.

The TCP-MIB defined in [RFC4022] consists of several objects and two tables:
Parameters of a TCP protocol engine. These include parameters such as the retransmission algorithm in use and the retransmission timeout values.

Statistics of a TCP protocol engine. These include counters for the number of active/passive opens, input/output segments, and errors.

A TCP connection table provides access to status information for all TCP connections handled by a TCP protocol engine. In addition, the table reports identification of the operating system level processes that handle the TCP connections.

A TCP listener table provides access to information about all TCP listening endpoints known by a TCP protocol engine. And as with the connection table, this table also reports the identification of the operating system level processes that handle this listening TCP endpoint.

The extended statistics defined in [RFC4898] have been omitted in the initial version of this specification. They can be added as needed.

3.2. Tree Diagram

[[Editor’s node: This section is TBD.]]
module: TCP-MIB
  +--rw tcp
    +--ro tcpRtoAlgorithm? enumeration
    +--ro tcpRtoMin? int32
    +--ro tcpRtoMax? int32
    +--ro tcpMaxConn? int32
    +--ro tcpActiveOpens? yang:counter32
    +--ro tcpPassiveOpens? yang:counter32
    +--ro tcpAttemptFails? yang:counter32
    +--ro tcpEstabResets? yang:counter32
    +--ro tcpCurrEstab? yang:gauge32
    +--ro tcpInSegs? yang:counter32
    +--ro tcpOutSegs? yang:counter32
    +--ro tcpRetransSegs? yang:counter32
  x--rw tcpConnEntry* [tcpConnLocalAddress tcpConnLocalPort tcpConnRemAddress tcpConnRemPort]
    x--rw tcpConnState? enumeration
    x--ro tcpConnLocalAddress inet:ipv4-address
    x--ro tcpConnLocalPort int32
    x--ro tcpConnRemAddress inet:ipv4-address
    x--ro tcpConnRemPort int32
    +--ro tcpInErrs? yang:counter32
    +--ro tcpOutRsts? yang:counter32
    +--ro tcpHCInSegs? yang:counter64
    +--ro tcpHCOutSegs? yang:counter64
  +--rw tcpConnectionEntry* [tcpConnectionLocalAddressType tcpConnectionLocalAddress tcpConnectionLocalPort tcpConnectionRemAddressType tcpConnectionRemAddress tcpConnectionRemPort]
    +--ro tcpConnectionLocalAddressType inet-address:InetAddressType
    +--ro tcpConnectionLocalAddress inet-address:InetAddress
    +--ro tcpConnectionLocalPort inet-address:InetPortNumber
    +--ro tcpConnectionRemAddressType inet-address:InetAddressType
    +--ro tcpConnectionRemAddress inet-address:InetAddress
    +--ro tcpConnectionRemPort inet-address:InetPortNumber
    +--rw tcpConnectionState? enumeration
    +--ro tcpConnectionProcess? uint32
  +--rw tcpListenerEntry* [tcpListenerLocalAddressType tcpListenerLocalAddress tcpListenerLocalPort]
    +--ro tcpListenerLocalAddressType inet-address:InetAddressType
    +--ro tcpListenerLocalAddress inet-address:InetAddress
    +--ro tcpListenerLocalPort inet-address:InetPortNumber
    +--ro tcpListenerProcess? uint32

4. TCP YANG Model

[[Editor’s node: This section is TBD.]]

The following YANG module has been generated by smidump 0.4.8 using the command "smidump -f yang TCP-MIB". Improvements to the translated YANG model are planned for future versions of this specification. The -00 version of this model is just published as a
baseline and to enable tracking of changes in later versions of the memo.

<CODE BEGINS> file "ietf-tcp-translated@2005-02-18.yang"
module TCP-MIB {

    /*** NAMESPACE / PREFIX DEFINITION ***/
    prefix "tcp-mib";

    /*** LINKAGE (IMPORTS / INCLUDES) ***/
    import INET-ADDRESS-MIB { prefix "inet-address"; } 
    import inet-types       { prefix "inet"; }
    import yang-types       { prefix "yang"; }

    /*** META INFORMATION ***/
    organization
        "TBD";

    contact
        "TBD";

    description
        "TBD";

    revision "2005-02-18" {
        description
            "IP version neutral revision, published as RFC 4022.";
    }
    revision "1994-11-01" {
        description
            "Initial SMIv2 version, published as RFC 2012.";
    }
    revision "1991-03-31" {
        description
            "The initial revision of this MIB module was part of MIB-II.";
    }

    container tcp {

        leaf tcpRtoAlgorithm {
            type enumeration {
                enum other    { value 1; }
                enum constant { value 2; }
            }
        }

Scharf                Expires August 28, 2019                [Page 8]
enum rsre { value 3; }
enum vanj { value 4; }
enum rfc2988 { value 5; }

cfg false;
description
"The algorithm used to determine the timeout value used for
retransmitting unacknowledged octets."
}

leaf tcpRtoMin {
type int32 {
    range "0..2147483647";
}
units "milliseconds";
cfg false;
description
"The minimum value permitted by a TCP implementation for
the retransmission timeout, measured in milliseconds.
More refined semantics for objects of this type depend
on the algorithm used to determine the retransmission
timeout; in particular, the IETF standard algorithm
rfc2988(5) provides a minimum value."
}

leaf tcpRtoMax {
type int32 {
    range "0..2147483647";
}
units "milliseconds";
cfg false;
description
"The maximum value permitted by a TCP implementation for
the retransmission timeout, measured in milliseconds.
More refined semantics for objects of this type depend
on the algorithm used to determine the retransmission
timeout; in particular, the IETF standard algorithm
rfc2988(5) provides an upper bound (as part of an
adaptive backoff algorithm)."
}

leaf tcpMaxConn {
type int32 {
    range "-1..2147483647";
}

cfg false;
description
"The limit on the total number of TCP connections the entity
Scharf                             Expires August 28, 2019
can support. In entities where the maximum number of
connections is dynamic, this object should contain the
value -1.

leaf tcpActiveOpens {
  type yang:counter32;
  config false;
  description
  "The number of times that TCP connections have made a direct
  transition to the SYN-SENT state from the CLOSED state.

  Discontinuities in the value of this counter are
  indicated via discontinuities in the value of sysUpTime."
}

leaf tcpPassiveOpens {
  type yang:counter32;
  config false;
  description
  "The number of times TCP connections have made a direct
  transition to the SYN-RCVD state from the LISTEN state.

  Discontinuities in the value of this counter are
  indicated via discontinuities in the value of sysUpTime."
}

leaf tcpAttemptFails {
  type yang:counter32;
  config false;
  description
  "The number of times that TCP connections have made a direct
  transition to the CLOSED state from either the SYN-SENT
  state or the SYN-RCVD state, plus the number of times that
  TCP connections have made a direct transition to the
  LISTEN state from the SYN-RCVD state.

  Discontinuities in the value of this counter are
  indicated via discontinuities in the value of sysUpTime."
}

leaf tcpEstabResets {
  type yang:counter32;
  config false;
  description
  "The number of times that TCP connections have made a direct
  transition to the CLOSED state from either the ESTABLISHED
  state or the CLOSE-WAIT state.
Discontinuities in the value of this counter are indicated via discontinuities in the value of sysUpTime.

leaf tcpCurrEstab {
    type yang:gauge32;
    config false;
    description
    "The number of TCP connections for which the current state is either ESTABLISHED or CLOSE-WAIT."
}

leaf tcpInSegs {
    type yang:counter32;
    config false;
    description
    "The total number of segments received, including those received in error. This count includes segments received on currently established connections. Discontinuities in the value of this counter are indicated via discontinuities in the value of sysUpTime."
}

leaf tcpOutSegs {
    type yang:counter32;
    config false;
    description
    "The total number of segments sent, including those on current connections but excluding those containing only retransmitted octets. Discontinuities in the value of this counter are indicated via discontinuities in the value of sysUpTime."
}

leaf tcpRetransSegs {
    type yang:counter32;
    config false;
    description
    "The total number of segments retransmitted; that is, the number of TCP segments transmitted containing one or more previously transmitted octets. Discontinuities in the value of this counter are indicated via discontinuities in the value of sysUpTime."
}
/* XXX table comments here XXX */

list tcpConnEntry {

    key "tcpConnLocalAddress tcpConnLocalPort
tcpConnRemAddress tcpConnRemPort";

    status deprecated;

description
"A conceptual row of the tcpConnTable containing information
about a particular current IPv4 TCP connection. Each row
of this table is transient in that it ceases to exist when
(or soon after) the connection makes the transition to the
CLOSED state."

leaf tcpConnState {
    type enumeration {
        enum closed      { value 1; }
        enum listen      { value 2; }
        enum synSent     { value 3; }
        enum synReceived { value 4; }
        enum established { value 5; }
        enum finWait1    { value 6; }
        enum finWait2    { value 7; }
        enum closeWait   { value 8; }
        enum lastAck     { value 9; }
        enum closing     { value 10; }
        enum timeWait    { value 11; }
        enum deleteTCB   { value 12; }
    }

    config true;

    status deprecated;

description
"The state of this TCP connection.

The only value that may be set by a management station is
deleteTCB(12). Accordingly, it is appropriate for an agent
to return a 'badValue' response if a management station
attempts to set this object to any other value.

If a management station sets this object to the value
deleteTCB(12), then the TCB (as defined in [RFC793]) of
the corresponding connection on the managed node is
deleted, resulting in immediate termination of the
connection.

As an implementation-specific option, a RST segment may be
sent from the managed node to the other TCP endpoint (note,
however, that RST segments are not sent reliably)."
}

leaf tcpConnLocalAddress {
  type inet:ipv4-address;
  config false;
  status deprecated;
  description
  "The local IP address for this TCP connection. In the case
  of a connection in the listen state willing to
  accept connections for any IP interface associated with the
  node, the value 0.0.0.0 is used.";
}

leaf tcpConnLocalPort {
  type int32 {
    range "0..65535";
  }
  config false;
  status deprecated;
  description
  "The local port number for this TCP connection.";
}

leaf tcpConnRemAddress {
  type inet:ipv4-address;
  config false;
  status deprecated;
  description
  "The remote IP address for this TCP connection.";
}

leaf tcpConnRemPort {
  type int32 {
    range "0..65535";
  }
  config false;
  status deprecated;
  description
  "The remote port number for this TCP connection.";
}

leaf tcpInErrs {
  type yang:counter32;
  config false;
  description
  "The total number of segments received in error (e.g., bad
TCP checksums).

Discontinuities in the value of this counter are indicated via discontinuities in the value of sysUpTime.

leaf tcpOutRsts {
  type yang:counter32;
  config false;
  description "The number of TCP segments sent containing the RST flag. Discontinuities in the value of this counter are indicated via discontinuities in the value of sysUpTime."
}

leaf tcpHCInSegs {
  type yang:counter64;
  config false;
  description "The total number of segments received, including those received in error. This count includes segments received on currently established connections. This object is the 64-bit equivalent of tcpInSegs. Discontinuities in the value of this counter are indicated via discontinuities in the value of sysUpTime."
}

leaf tcpHCOutSegs {
  type yang:counter64;
  config false;
  description "The total number of segments sent, including those on current connections but excluding those containing only retransmitted octets. This object is the 64-bit equivalent of tcpOutSegs. Discontinuities in the value of this counter are indicated via discontinuities in the value of sysUpTime."
}

/* XXX table comments here XXX */

list tcpConnectionEntry {

Scharf Expires August 28, 2019 [Page 14]
key "tcpConnectionLocalAddressType"
  tcpConnectionLocalAddress tcpConnectionLocalPort
  tcpConnectionRemAddressType tcpConnectionRemAddress
  tcpConnectionRemPort";

description
"A conceptual row of the tcpConnectionTable containing
information about a particular current TCP connection.
Each row of this table is transient in that it ceases to
exist when (or soon after) the connection makes the
transition to the CLOSED state.";

leaf tcpConnectionLocalAddressType {
  type inet-address:InetAddressType;
  config false;
  description
  "The address type of tcpConnectionLocalAddress.";
}

leaf tcpConnectionLocalAddress {
  type inet-address:InetAddress;
  config false;
  description
  "The local IP address for this TCP connection. The type
  of this address is determined by the value of
tcpConnectionLocalAddressType.

  As this object is used in the index for the
tcpConnectionTable, implementors should be
careful not to create entries that would result in OIDs
with more than 128 subidentifiers; otherwise the information
cannot be accessed by using SNMPv1, SNMPv2c, or SNMPv3.";
}

leaf tcpConnectionLocalPort {
  type inet-address:InetPortNumber;
  config false;
  description
  "The local port number for this TCP connection.";
}

leaf tcpConnectionRemAddressType {
  type inet-address:InetAddressType;
  config false;
  description
  "The address type of tcpConnectionRemAddress.";
}
leaf tcpConnectionRemAddress {
  type inet-address:InetAddress;
  config false;
  description
   "The remote IP address for this TCP connection. The type
   of this address is determined by the value of
   tcpConnectionRemAddressType.

   As this object is used in the index for the
   tcpConnectionTable, implementors should be
   careful not to create entries that would result in OIDs
   with more than 128 subidentifiers; otherwise the information
   cannot be accessed by using SNMPv1, SNMPv2c, or SNMPv3.";
}

leaf tcpConnectionRemPort {
  type inet-address:InetPortNumber;
  config false;
  description
   "The remote port number for this TCP connection.";
}

leaf tcpConnectionState {
  type enumeration {
    enum closed     { value 1; }
    enum listen     { value 2; }
    enum synSent    { value 3; }
    enum synReceived { value 4; }
    enum established { value 5; }
    enum finWait1   { value 6; }
    enum finWait2   { value 7; }
    enum closeWait  { value 8; }
    enum lastAck    { value 9; }
    enum closing    { value 10; }
    enum timeWait   { value 11; }
    enum deleteTCB  { value 12; }
  }
  config true;
  description
   "The state of this TCP connection.

   The value listen(2) is included only for parallelism to the
   old tcpConnTable and should not be used. A connection in
   LISTEN state should be present in the tcpListenerTable.

   The only value that may be set by a management station is
   deleteTCB(12). Accordingly, it is appropriate for an agent
   to return a ‘badValue’ response if a management station
attempts to set this object to any other value.

If a management station sets this object to the value deleteTCB(12), then the TCB (as defined in [RFC793]) of the corresponding connection on the managed node is deleted, resulting in immediate termination of the connection.

As an implementation-specific option, a RST segment may be sent from the managed node to the other TCP endpoint (note, however, that RST segments are not sent reliably)."

leaf tcpConnectionProcess {
  type uint32;
  config false;
  description
  "The system’s process ID for the process associated with this connection, or zero if there is no such process. This value is expected to be the same as HOST-RESOURCES-MIB::hrSWRunIndex or SYSAPPL-MIB::sysApplElmtRunIndex for some row in the appropriate tables.";
}

leaf tcpConnectionProcess {
  type uint32;
  config false;
  description
  "The system’s process ID for the process associated with this connection, or zero if there is no such process. This value is expected to be the same as HOST-RESOURCES-MIB::hrSWRunIndex or SYSAPPL-MIB::sysApplElmtRunIndex for some row in the appropriate tables.";
}

list tcpListenerEntry {
  key "tcpListenerLocalAddressType tcpListenerLocalAddress tcpListenerLocalPort";
  description
  "A conceptual row of the tcpListenerTable containing information about a particular TCP listener.";

leaf tcpListenerLocalAddressType {
  type inet-address:InetAddressType;
  config false;
  description
  "The address type of tcpListenerLocalAddress. The value should be unknown (0) if connection initiations to all local IP addresses are accepted.";
}

leaf tcpListenerLocalAddress {
  type inet-address:InetAddress;
The value of this object can be represented in three possible ways, depending on the characteristics of the listening application:

1. For an application willing to accept both IPv4 and IPv6 datagrams, the value of this object must be ''h (a zero-length octet-string), with the value of the corresponding tcpListenerLocalAddressType object being unknown (0).

2. For an application willing to accept only IPv4 or IPv6 datagrams, the value of this object must be '0.0.0.0' or '::' respectively, with tcpListenerLocalAddressType representing the appropriate address type.

3. For an application which is listening for data destined only to a specific IP address, the value of this object is the specific local address, with tcpListenerLocalAddressType representing the appropriate address type.

As this object is used in the index for the tcpListenerTable, implementors should be careful not to create entries that would result in OIDs with more than 128 subidentifiers; otherwise the information cannot be accessed, using SNMPv1, SNMPv2c, or SNMPv3.
5. IANA Considerations

[[Editor’s node: This section will be completed in follow-up versions of this document.]]

6. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

7. References

7.1. Normative References


7.2.  Informative References

[I-D.ietf-netconf-netconf-client-server]

[I-D.ietf-netmod-acl-model]

[I-D.ietf-opsawg-nat-yang]


Appendix A. Acknowledgements

Michael Scharf is supported by the StandICT.eu project, which is funded by the European Commission under the Horizon 2020 Programme.

The YANG model used in version -00 of the document has been converted from [RFC4022] by the LIBSMI library [LIBSMI].

Appendix B. Changes compared to previous versions

Changes compared to draft-scharf-tcpm-yang-tcp-00

- Editorial improvements

Author’s Address

Michael Scharf
Hochschule Esslingen – University of Applied Sciences
Flandernstr. 101
Esslingen  73732
Germany

Email: michael.scharf@hs-esslingen.de