

Domain Name System Operations  
Internet-Draft  
Updates: 1123, 1536 (if approved)  
Intended status: Best Current Practice  
Expires: 10 July 2022

J.T. Kristoff  
DataPlane.org  
D. Wessels  
Verisign  
6 January 2022

DNS Transport over TCP - Operational Requirements  
draft-ietf-dnsop-dns-tcp-requirements-15

Abstract

This document updates RFC 1123 and RFC 1536. This document requires the operational practice of permitting DNS messages to be carried over TCP on the Internet as a Best Current Practice. This operational requirement is aligned with the implementation requirements in RFC 7766. The use of TCP includes both DNS over unencrypted TCP, as well as over an encrypted TLS session. The document also considers the consequences of this form of DNS communication and the potential operational issues that can arise when this Best Current 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 10 July 2022.

Copyright Notice

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

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

and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction . . . . .	4
1.1. Requirements Language . . . . .	4
2. History of DNS over TCP . . . . .	4
2.1. Uneven Transport Usage and Preference . . . . .	5
2.2. Waiting for Large Messages and Reliability . . . . .	5
2.3. EDNS(0) . . . . .	6
2.4. Fragmentation and Truncation . . . . .	6
2.5. "Only Zone Transfers Use TCP" . . . . .	8
2.6. Reuse, Pipelining, and Out-of-Order Processing . . . . .	8
3. DNS over TCP Requirements . . . . .	9
4. Network and System Considerations . . . . .	10
4.1. Connection Establishment and Admission . . . . .	10
4.2. Connection Management . . . . .	12
4.3. Connection Termination . . . . .	13
4.4. DNS-over-TLS . . . . .	13
4.5. Defaults and Recommended Limits . . . . .	14
5. DNS over TCP Filtering Risks . . . . .	15
5.1. Truncation, Retries, and Timeouts . . . . .	15
5.2. DNS Root Zone KSK Rollover . . . . .	16
6. Logging and Monitoring . . . . .	16
7. IANA Considerations . . . . .	17
8. Security Considerations . . . . .	17
9. Privacy Considerations . . . . .	18
10. Acknowledgments . . . . .	18
11. References . . . . .	18
11.1. Normative References . . . . .	18
11.2. Informative References . . . . .	19
Appendix A. Standards Related to DNS Transport over TCP . . . . .	27
A.1. IETF RFC 1035 - DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION . . . . .	27
A.2. IETF RFC 1536 - Common DNS Implementation Errors and Suggested Fixes . . . . .	27
A.3. IETF RFC 1995 - Incremental Zone Transfer in DNS . . . . .	27
A.4. IETF RFC 1996 - A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY) . . . . .	27
A.5. IETF RFC 2181 - Clarifications to the DNS Specification . . . . .	27
A.6. IETF RFC 2694 - DNS extensions to Network Address Translators (DNS_ALG) . . . . .	28
A.7. IETF RFC 3225 - Indicating Resolver Support of DNSSEC . . . . .	28

A.8.	IETF RFC 3226 - DNSSEC and IPv6 A6 aware server/resolver message size requirements . . . . .	28
A.9.	IETF RFC 4472 - Operational Considerations and Issues with IPv6 DNS . . . . .	28
A.10.	IETF RFC 5452 - Measures for Making DNS More Resilient against Forged Answers . . . . .	28
A.11.	IETF RFC 5507 - Design Choices When Expanding the DNS . .	29
A.12.	IETF RFC 5625 - DNS Proxy Implementation Guidelines . . .	29
A.13.	IETF RFC 5936 - DNS Zone Transfer Protocol (AXFR) . . . .	29
A.14.	IETF RFC 7534 - AS112 Nameserver Operations . . . . .	29
A.15.	IETF RFC 6762 - Multicast DNS . . . . .	29
A.16.	IETF RFC 6891 - Extension Mechanisms for DNS (EDNS(0)) .	29
A.17.	IETF RFC 6950 - Architectural Considerations on Application Features in the DNS . . . . .	30
A.18.	IETF RFC 7477 - Child-to-Parent Synchronization in DNS .	30
A.19.	IETF RFC 7720 - DNS Root Name Service Protocol and Deployment Requirements . . . . .	30
A.20.	IETF RFC 7766 - DNS Transport over TCP - Implementation Requirements . . . . .	30
A.21.	IETF RFC 7828 - The edns-tcp-keepalive EDNS(0) Option . .	30
A.22.	IETF RFC 7858 - Specification for DNS over Transport Layer Security (TLS) . . . . .	31
A.23.	IETF RFC 7873 - Domain Name System (DNS) Cookies . . . .	31
A.24.	IETF RFC 7901 - CHAIN Query Requests in DNS . . . . .	31
A.25.	IETF RFC 8027 - DNSSEC Roadblock Avoidance . . . . .	31
A.26.	IETF RFC 8094 - DNS over Datagram Transport Layer Security (DTLS) . . . . .	32
A.27.	IETF RFC 8162 - Using Secure DNS to Associate Certificates with Domain Names for S/MIME . . . . .	32
A.28.	IETF RFC 8324 - DNS Privacy, Authorization, Special Uses, Encoding, Characters, Matching, and Root Structure: Time for Another Look? . . . . .	32
A.29.	IETF RFC 8467 - Padding Policies for Extension Mechanisms for DNS (EDNS(0)) . . . . .	32
A.30.	IETF RFC 8482 - Providing Minimal-Sized Responses to DNS Queries That Have QTYPE=ANY . . . . .	32
A.31.	IETF RFC 8483 - Yeti DNS Testbed . . . . .	33
A.32.	IETF RFC 8484 - DNS Queries over HTTPS (DoH) . . . . .	33
A.33.	IETF RFC 8490 - DNS Stateful Operations . . . . .	33
A.34.	IETF RFC 8501 - Reverse DNS in IPv6 for Internet Service Providers . . . . .	33
A.35.	IETF RFC 8806 - Running a Root Server Local to a Resolver . . . . .	33
A.36.	IETF RFC 8906 - A Common Operational Problem in DNS Servers: Failure to Communicate . . . . .	33
A.37.	IETF RFC 8932 - Recommendations for DNS Privacy Service Operators . . . . .	34

A.38. IETF RFC 8945 - Secret Key Transaction Authentication for	
DNS (TSIG) . . . . .	34
Authors' Addresses . . . . .	34

## 1. Introduction

DNS messages are 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 declare that support for TCP is a required part of the DNS implementation specifications [RFC7766]. 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. It updates [RFC1123] Section 6.1.3.2 to clarify that all DNS resolvers and recursive servers MUST support and service both TCP and UDP queries, and also updates [RFC1536] to remove the misconception that TCP is only useful for zone transfers.

### 1.1. 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.

## 2. History of DNS over TCP

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

## 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, whose operational considerations are originally given in [RFC2541]. The former suggested "requestors who require an accurate response code must use TCP," while the latter 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 in the late 1990s. 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. EDNS(0)

In 1999 the IETF published the Extension Mechanisms for DNS (EDNS(0)) in [RFC2671] (superseded in 2013 by an update in [RFC6891]). That 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 EDNS(0)-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 over UDP. In practice, typical values are a subset of the 512- to 4096-byte range. EDNS(0) became widely deployed over the next several years, and numerous surveys ([CASTRO2010], [NETALYZR]) have shown that many systems support larger UDP MTUs with EDNS(0).

The natural effect of EDNS(0) 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 EDNS(0) or fell back to TCP when necessary, DNS clients still strongly prefer UDP to TCP. For example, as of 2014, DNS over TCP transactions remained a very small fraction of overall DNS traffic received by root name servers [VERISIGN].

### 2.4. Fragmentation and Truncation

Although EDNS(0) 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 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, although tunnel encapsulation may reduce that maximum message size in some cases.

For IPv6, the situation is a little more complicated. First, IPv6 headers are 40 bytes (versus 20 without options in IPv4). Second, approximately one third of DNS recursive resolvers use the minimum MTU of 1280 bytes [APNIC]. 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. When the originating host receives a signal that fragmentation is required, it is expected to populate its Path MTU cache for that destination. The application, then, will retry the query after a timeout since the host does not generally retain copies of messages sent over UDP for potential retransmission.

The practical consequence of all this is that DNS requestors must be prepared to retry queries with different EDNS(0) maximum message size values. Administrators of [BIND] are likely to be familiar with seeing "success resolving ... after reducing the advertised EDNS(0) UDP packet size to 512 octets" messages in their system logs.

Often, reducing the EDNS(0) 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.

Note that a receiver is unable to differentiate between packets lost due to congestion and packets (fragments) intentionally dropped by firewalls or middleboxes. Over network paths with non-trivial amounts of packet loss, larger, fragmented DNS responses are more likely to never arrive and time out compared to smaller, unfragmented responses. Clients might be misled into retrying queries with different EDNS(0) UDP packet size values for the wrong reason.

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 Key Signing Key (KSK) Rollover design team [DESIGNTEAM] 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 [ECDSA].

More recently, renewed security concerns about fragmented DNS messages ([AVOID\_FRAGS], [FRAG\_POISON]) are leading implementors to consider smaller responses and lower default EDNS(0) UDP payload size values for both queriers and responders [FLAGDAY2020].

## 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 [FLAGDAY2020]. 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 is 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 historical 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.

## 2.6. Reuse, Pipelining, and Out-of-Order Processing

The idea that a TCP connection can support multiple transactions goes back as far as [RFC0883], which states: "Multiple messages may be sent over a virtual circuit." Although [RFC1035], which updates the former, omits this particular detail, it has been generally accepted that a TCP connection can be used for more than one query and response.

[RFC5966] clarified that servers are not required to preserve the order of queries and responses over any transport. [RFC7766], which updates the former, further encourages query pipelining over TCP to achieve performance on par with UDP. A server that sends out-of-



order responses to pipelined queries avoids head-of-line blocking when the response for a later query is ready before the response to an earlier query.

However, TCP can potentially suffer from a different head-of-line blocking problem due to packet loss. Since TCP itself enforces ordering, a single lost segment delays delivery of data in any following segments until the lost segment is retransmitted and successfully received.

### 3. DNS over TCP Requirements

An average increase in DNS message size (e.g., due to DNSSEC), the continued development of new DNS features (Appendix A), and a denial of service mitigation technique (Section 8), all show 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 research that argues connection-oriented DNS transactions may provide security and privacy advantages over UDP transport [TDNS]. In fact, the standard for DNS over TLS [RFC7858] 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.

- \* DNS servers (including forwarders) MUST support and service TCP for receiving queries, so that clients can reliably receive responses that are larger than what either side considers too large for UDP.
- \* DNS clients MUST support TCP for sending queries, so that they can retry truncated UDP responses as necessary.

Furthermore, the requirement in Section 6.1.3.2 of [RFC1123] around limiting the resources a server devotes to queries is hereby updated:

OLD:

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.

NEW:

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.

Lastly, Section 1 of [RFC1536] is updated to eliminate the misconception that TCP is only useful for zone transfers:

OLD:

DNS implements the classic request-response scheme of client-server interaction. UDP is, therefore, the chosen protocol for communication though TCP is used for zone transfers.

NEW:

DNS implements the classic request-response scheme of client-server interaction.

Filtering of DNS over TCP is 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, 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 Establishment and Admission

Resolvers and other DNS clients should be aware that some servers might not be reachable over TCP. For this reason, clients MAY track and limit the number of TCP connections and connection attempts to a single server. Reachability problems can be caused by network elements close to the server, close to the client, or anywhere along the path between them. Mobile clients that cache connection failures MAY do so on a per-network basis, or MAY clear such a cache upon change of network.

Additionally, DNS clients MAY enforce a short timeout on unestablished connections, rather than rely on the host operating system's TCP connection timeout, which is often around 60-120 seconds

(i.e., due to an initial retransmission timeout of 1 second, the exponential back off rules of [RFC6298], and a limit of six retries as is the default in Linux).

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, described in Section 3.6 of [RFC4987], 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 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. See also [RFC5358].

The FreeBSD and NetBSD operating systems have an "accept filter" feature ([accept\_filter]) 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. The Linux `TCP_DEFER_ACCEPT` feature, while more limited in scope, can provide some of the same benefits as the BSD accept filter feature. These features are implemented as low-level socket options, and are not activated automatically. If applications wish to use these features, they need to make specific calls to set the right options, and administrators may also need to configure the applications to appropriately use the features.

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 either use the same TFO server key on all instances, or disable TFO for all members of the cluster.

DNS clients MAY also enable TFO. At the time of this writing, on some operating systems it is not implemented, or is 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 clients and servers SHOULD 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.

Operators of DNS server software SHOULD be aware that operating system and application vendors MAY impose a limit on the total number of established connections. These limits may be designed to protect against DDoS attacks or performance degradation. Operators SHOULD understand how to increase these limits if necessary, and the consequences of doing so. Limits imposed by the application SHOULD be lower than limits imposed by the operating system, so that the application can apply its own policy to connection management, such as closing the oldest idle connections first.

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 cannot consume all TCP resources and deny service to other users. Note, however, that if this limit is enabled, it possibly limits client performance while leaving some TCP resources unutilized. Operators SHOULD be aware of these tradeoffs and ensure this limit, if configured, is set appropriately based on the number and diversity of their users, and whether users connect from unique IP addresses or through a shared Network Address Translator [RFC3022].

DNS server software SHOULD provide a configurable timeout for idle TCP connections. This can be used to free up resources for new connections and to ensure that idle connections are eventually closed. At the same time, it possibly limits client performance while leaving some TCP resources unutilized. 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 can help protect against unfair connection use (e.g., not releasing connection slots to other clients) and network evasion attacks.

Similarly, DNS server software MAY provide a configurable limit on the total duration of a TCP connection. This can help protect against unfair connection use, slow read attacks, and network evasion attacks.

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

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. It is generally preferable for clients to initiate the close of a TCP connection so that busy servers do not accumulate many sockets in the TIME\_WAIT state, which can cause performance problems or even denial of service. The edns-tcp-keepalive EDNS(0) option [RFC7828] can be used to encourage clients to close connections.

On systems where large numbers of sockets in TIME\_WAIT are observed (either as client or server), and are affecting an application's performance, it may be tempting to tune local TCP parameters. For example, the Linux kernel has a "sysctl" parameter named net.ipv4.tcp\_tw\_reuse which allows connections in the TIME\_WAIT state to be reused in specific circumstances. Note, however, this affects only outgoing (client) connections and has no impact on servers. In most cases it is NOT RECOMMENDED to change parameters related to the TIME\_WAIT state. It should only be done by those with detailed knowledge of both TCP and the affected application.

#### 4.4. 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 DNS-over-TLS utilizes TCP port 853 instead of port 53, this document applies equally well to DNS-over-TLS. Note, however, DNS-over-TLS is only defined between stubs and recursives at the time of this writing.

The use of TLS places even stronger operational burdens on DNS clients and servers. Cryptographic functions for authentication and encryption require additional processing. Unoptimized connection setup with TLS 1.3 [RFC8446] takes one additional round-trip compared

to TCP. Connection setup times can be reduced with TCP Fast Open, and TLS False Start [RFC7918] for TLS 1.2. TLS 1.3 session resumption does not reduce round-trip latency because no application profile for use of TLS 0-RTT data with DNS has been published at the time of this writing. However, TLS session resumption can reduce the number of cryptographic operations, and in TLS 1.2, session resumption does reduce the number of additional round trips from two to one.

#### 4.5. Defaults and Recommended Limits

A survey of features and defaults was conducted for popular open source DNS server implementations at the time of writing. This section documents those defaults and makes recommendations for configurable limits that can be used in the absence of any other information. Any recommended values in this document are only intended as a starting point for administrators that are unsure what sorts of limits might be reasonable. Operators SHOULD use application-specific monitoring, system logs, and system monitoring tools to gauge whether their service is operating within or exceeding these limits, and adjust accordingly.

Most open source DNS server implementations provide a configurable limit on the total number of established connections. Default values range from 20 to 150. In most cases, where the majority of queries take place over UDP, 150 is a reasonable limit. For services or environments where most queries take place over TCP or TLS, 5000 is a more appropriate limit.

Only some open source implementations provide a way to limit the number of connections per source IP address or subnet, but the default is to have no limit. For environments or situations where it may be necessary to enable this limit, 25 connections per source IP address is a reasonable starting point. The limit should be increased when aggregated by subnet, or for services where most queries take place over TCP or TLS.

Most open source implementations provide a configurable idle timeout on connections. Default values range from 2 to 30 seconds. In most cases, 10 seconds is a reasonable default for this limit. Longer timeouts improve connection reuse, but busy servers may need to use a lower limit.

Only some open source implementations provide a way to limit the number of transactions per connection, but the default is to have no limit. This document does not offer advice on particular values for such a limit.

Only some open source implementations provide a way to limit the duration of connection, but the default is to have no limit. This document does not offer advice on particular values for such a limit.

## 5. DNS over TCP Filtering Risks

Networks that filter DNS over TCP risk losing access to significant or important pieces of the DNS namespace. For a variety of reasons a DNS answer may require a DNS over TCP query. This may include large message sizes, lack of EDNS(0) support, DDoS mitigation techniques (including [RRL]), 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 effectiveness 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 at the time of this writing when networks filter DNS over TCP.

### 5.1. Truncation, Retries, and Timeouts

Networks that filter DNS over TCP may inadvertently cause problems for third-party resolvers as experienced by [TOYAMA]. For example, a resolver receives queries for a moderately popular domain. The resolver forwards the queries to the domain's authoritative name servers, but those servers respond with the TC bit set. The resolver retries over TCP, but the authoritative server blocks DNS over TCP. The pending connections consume resources on the resolver until they time out. If the number and frequency of these truncated-and-then-blocked queries is sufficiently high, the resolver wastes valuable resources on queries that can never be answered. This condition is generally not easily or completely mitigated by the affected DNS resolver operator.

## 5.2. DNS Root Zone KSK Rollover

The plans for deploying a new root zone DNSSEC KSK highlighted a potential problem in retrieving the root zone key set [LEWIS]. During some phases of the KSK rollover process, root zone DNSKEY responses were larger than 1280 bytes, the IPv6 minimum MTU for links carrying IPv6 traffic [RFC8200]. There was some concern [KSK\_ROLLOVER\_ARCHIVES] that any DNS server unable to receive large DNS messages over UDP, or any DNS message over TCP, would experience disruption while performing DNSSEC validation.

However, during the year-long postponement of the KSK rollover there were no reported problems that could be attributed to the 1414 octet DNSKEY response when both the old and new keys were published in the zone. Additionally, there were no reported problems during the two-month period when the old key was published as revoked and the DNSKEY response was 1425 octets in size [ROLL\_YOUR\_ROOT].

## 6. Logging and Monitoring

Developers of applications that log or monitor DNS SHOULD NOT ignore TCP due to the perception that it is rarely used or is hard to process. Operators SHOULD ensure that their monitoring and logging applications properly capture DNS messages over TCP. 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 might attempt to hide certain messages by forcing them over very small TCP segments. Applications that capture network packets (e.g., with libpcap [libpcap]) SHOULD implement and perform full TCP stream reassembly and analyze the reassembled stream instead of the individual packets. Otherwise, they are vulnerable to network evasion attacks [phrack]. Furthermore, such applications need to protect themselves from resource exhaustion attacks by limiting the amount of memory allocated to tracking unacknowledged connection state data. dnscap [dnscap] is an open-source example of a DNS logging program that implements TCP stream reassembly.

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

As an alternative to packet capture, some DNS server software supports dnstap [dnstap] as an integrated monitoring protocol intended to facilitate wide-scale DNS monitoring.



## 7. IANA Considerations

This memo includes no request to IANA.

## 8. Security Considerations

This document, providing operational requirements, is the companion to the implementation requirements of DNS over TCP, provided in [RFC7766]. The security considerations from [RFC7766] still apply.

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. In fact, [DAI21] details a class of IP fragmentation attacks on DNS transactions if the IP Identifier field (16 bits in IPv4 and 32 bits in IPv6) can be predicted and a system is coerced to fragment rather than retransmit messages. 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 similar to 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].

As mentioned in Section 6, applications that implement TCP stream reassembly need to limit the amount of memory allocated to connection tracking. A failure to do so could lead to a total failure of the logging or monitoring application. Imposition of resource limits creates a tradeoff between allowing some stream reassembly to continue and allowing some evasion attacks to succeed.

This document recommends that DNS Servers enable TFO when possible. [RFC7413] recommends that a pool of servers behind a load balancer with shared server IP address also share the key used to generate Fast Open cookies, to prevent inordinate fallback to the 3WHS. This guidance remains accurate, but comes with a caveat: compromise of one server would reveal this group-shared key, and allow for attacks involving the other servers in the pool by forging invalid Fast Open cookies.

## 9. Privacy Considerations

Since DNS over both UDP and TCP uses the same underlying message format, the use of one transport instead of the other does not change the privacy characteristics of the message content (i.e., the name being queried). A number of protocols have recently been developed to provide DNS privacy, including DNS over TLS [RFC7858], DNS over DTLS [RFC8094], DNS over HTTPS [RFC8484], with even more on the way.

Because TCP is somewhat more complex than UDP, some characteristics of a TCP conversation may enable DNS client fingerprinting and tracking that is not possible with UDP. For example, the choice of initial sequence numbers, window size, and options might be able to identify a particular TCP implementation, or even individual hosts behind shared resources such as network address translators (NATs).

## 10. 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 between 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: Joe Abley, Piet Barber, Sara Dickinson, Tony Finch, Bob Harold, Paul Hoffman, Geoff Huston, Tatuja Jinmei, Puneet Sood, and Richard Wilhelm. 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.

## 11. References

### 11.1. Normative References

- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <<https://www.rfc-editor.org/info/rfc1035>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

- [RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, DOI 10.17487/RFC2181, July 1997, <<https://www.rfc-editor.org/info/rfc2181>>.
- [RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, DOI 10.17487/RFC6891, April 2013, <<https://www.rfc-editor.org/info/rfc6891>>.
- [RFC7766] Dickinson, J., Dickinson, S., Bellis, R., Mankin, A., and D. Wessels, "DNS Transport over TCP - Implementation Requirements", RFC 7766, DOI 10.17487/RFC7766, March 2016, <<https://www.rfc-editor.org/info/rfc7766>>.
- [RFC7828] Wouters, P., Abley, J., Dickinson, S., and R. Bellis, "The edns-tcp-keepalive EDNS0 Option", RFC 7828, DOI 10.17487/RFC7828, April 2016, <<https://www.rfc-editor.org/info/rfc7828>>.
- [RFC7873] Eastlake 3rd, D. and M. Andrews, "Domain Name System (DNS) Cookies", RFC 7873, DOI 10.17487/RFC7873, May 2016, <<https://www.rfc-editor.org/info/rfc7873>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

## 11.2. Informative References

- [accept\_filter] FreeBSD, "FreeBSD accept\_filter(9)", 7 May 2018, <[https://www.freebsd.org/cgi/man.cgi?query=accept\\_filter](https://www.freebsd.org/cgi/man.cgi?query=accept_filter)>.
- [APNIC] Huston, G., "DNS XL", October 2020, <<https://labs.apnic.net/?p=1380>>.
- [AVOID\_FRAGS] Fujiwara, K. and P. Vixie, "Fragmentation Avoidance in DNS", Work in Progress, draft-ietf-dnsop-avoid-fragmentation-05, February 2021.
- [BIND] Internet Systems Consortium, "BIND 9 - ISC", April 2021, <<https://www.isc.org/bind/>>.
- [CASTRO2010] Castro, S., Zhang, M., John, W., Wessels, D., and k.c. claffy, "Understanding and preparing for DNS evolution", 2010.

- [CHES94] Cheswick, W.R. and S.M. Bellovin, "Firewalls and Internet Security: Repelling the Wily Hacker", 1994.
- [CLOUDFLARE]  
Grant, D., "Economical With The Truth: Making DNSSEC Answers Cheap", 24 June 2016,  
<<https://blog.cloudflare.com/black-lies/>>.
- [DAI21] Tianxiang, T., Shulman, H., and M. Waidner, "DNS-over-TCP Considered Vulnerable", 2021.
- [DESIGNTEAM]  
Design Team Report, "Root Zone KSK Rollover Plan", 18 December 2015, <<https://www.iana.org/reports/2016/root-ksk-rollover-design-20160307.pdf>>.
- [DJBDNS] D.J. Bernstein, "When are TCP queries sent?", 2002,  
<<https://cr.yp.to/djbdns/tcp.html#why>>.
- [dnscap] DNS-OARC, "DNSCAP", 7 May 2018,  
<<https://www.dns-oarc.net/tools/dnscap>>.
- [dnstap] Edmonds, R. and P. Vixie, "dnstap", 7 May 2018,  
<<https://dnstap.info>>.
- [ECDSA] Rijswijk-Deij, R., Sperotto, A., and A. Pras, "Making the Case for Elliptic Curves in DNSSEC", September 2015,  
<<https://dl.acm.org/doi/10.1145/2831347.2831350>>.
- [FLAGDAY2020]  
Various DNS software and service providers, "DNS Flag Day 2020", October 2020, <<https://dnsflagday.net/2020/>>.
- [FRAG\_POISON]  
Herzberg, A. and H. Shulman, "Fragmentation Considered Poisonous", May 2012,  
<<https://u.cs.biu.ac.il/~herzbea/security/13-03-frag.pdf>>.
- [HUSTON] Huston, G., "Dealing with IPv6 fragmentation in the DNS", 22 August 2017, <<https://blog.apnic.net/2017/08/22/dealing-ipv6-fragmentation-dns/>>.
- [KSK\_ROLLOVER\_ARCHIVES]  
Internet Corporation for Assigned Names and Numbers, "KSK Rollover List Archives", January 2019,  
<<https://mm.icann.org/pipermail/ksk-rollover/2019-January/date.html>>.

- [LEWIS] Lewis, E., "2017 DNSSEC KSK Rollover", RIPE 74 Budapest, Hungary, 8 May 2017, <<https://ripe74.ripe.net/presentations/25-RIPE74-lewis-submission.pdf>>.
- [libpcap] Tcpdump/Libpcap, "Tcpdump and Libpcap", 7 May 2018, <<https://www.tcpdump.org>>.
- [NETALYZR] Kreibich, C., Weaver, N., Nechaev, B., and V. Paxson, "Netalyzer: Illuminating The Edge Network", 2010.
- [phrack] horizon, "Defeating Sniffers and Intrusion Detection Systems", December 1998, <<http://phrack.org/issues/54/10.html>>.
- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <<https://www.rfc-editor.org/info/rfc768>>.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC0883] Mockapetris, P., "Domain names: Implementation specification", RFC 883, DOI 10.17487/RFC0883, November 1983, <<https://www.rfc-editor.org/info/rfc883>>.
- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <<https://www.rfc-editor.org/info/rfc1034>>.
- [RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <<https://www.rfc-editor.org/info/rfc1123>>.
- [RFC1536] Kumar, A., Postel, J., Neuman, C., Danzig, P., and S. Miller, "Common DNS Implementation Errors and Suggested Fixes", RFC 1536, DOI 10.17487/RFC1536, October 1993, <<https://www.rfc-editor.org/info/rfc1536>>.
- [RFC1995] Ohta, M., "Incremental Zone Transfer in DNS", RFC 1995, DOI 10.17487/RFC1995, August 1996, <<https://www.rfc-editor.org/info/rfc1995>>.
- [RFC1996] Vixie, P., "A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)", RFC 1996, DOI 10.17487/RFC1996, August 1996, <<https://www.rfc-editor.org/info/rfc1996>>.

- [RFC2136] Vixie, P., Ed., Thomson, S., Rekhter, Y., and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)", RFC 2136, DOI 10.17487/RFC2136, April 1997, <<https://www.rfc-editor.org/info/rfc2136>>.
- [RFC2541] Eastlake 3rd, D., "DNS Security Operational Considerations", RFC 2541, DOI 10.17487/RFC2541, March 1999, <<https://www.rfc-editor.org/info/rfc2541>>.
- [RFC2671] Vixie, P., "Extension Mechanisms for DNS (EDNS0)", RFC 2671, DOI 10.17487/RFC2671, August 1999, <<https://www.rfc-editor.org/info/rfc2671>>.
- [RFC2694] Srisuresh, P., Tsirtsis, G., Akkiraju, P., and A. Heffernan, "DNS extensions to Network Address Translators (DNS\_ALG)", RFC 2694, DOI 10.17487/RFC2694, September 1999, <<https://www.rfc-editor.org/info/rfc2694>>.
- [RFC3022] Srisuresh, P. and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)", RFC 3022, DOI 10.17487/RFC3022, January 2001, <<https://www.rfc-editor.org/info/rfc3022>>.
- [RFC3225] Conrad, D., "Indicating Resolver Support of DNSSEC", RFC 3225, DOI 10.17487/RFC3225, December 2001, <<https://www.rfc-editor.org/info/rfc3225>>.
- [RFC3226] Gudmundsson, O., "DNSSEC and IPv6 A6 aware server/resolver message size requirements", RFC 3226, DOI 10.17487/RFC3226, December 2001, <<https://www.rfc-editor.org/info/rfc3226>>.
- [RFC4472] Durand, A., Ihren, J., and P. Savola, "Operational Considerations and Issues with IPv6 DNS", RFC 4472, DOI 10.17487/RFC4472, April 2006, <<https://www.rfc-editor.org/info/rfc4472>>.
- [RFC4953] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, DOI 10.17487/RFC4953, July 2007, <<https://www.rfc-editor.org/info/rfc4953>>.
- [RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <<https://www.rfc-editor.org/info/rfc4987>>.

- [RFC5358] Damas, J. and F. Neves, "Preventing Use of Recursive Nameservers in Reflector Attacks", BCP 140, RFC 5358, DOI 10.17487/RFC5358, October 2008, <<https://www.rfc-editor.org/info/rfc5358>>.
- [RFC5452] Hubert, A. and R. van Mook, "Measures for Making DNS More Resilient against Forged Answers", RFC 5452, DOI 10.17487/RFC5452, January 2009, <<https://www.rfc-editor.org/info/rfc5452>>.
- [RFC5507] IAB, Faltstrom, P., Ed., Austein, R., Ed., and P. Koch, Ed., "Design Choices When Expanding the DNS", RFC 5507, DOI 10.17487/RFC5507, April 2009, <<https://www.rfc-editor.org/info/rfc5507>>.
- [RFC5625] Bellis, R., "DNS Proxy Implementation Guidelines", BCP 152, RFC 5625, DOI 10.17487/RFC5625, August 2009, <<https://www.rfc-editor.org/info/rfc5625>>.
- [RFC5927] Gont, F., "ICMP Attacks against TCP", RFC 5927, DOI 10.17487/RFC5927, July 2010, <<https://www.rfc-editor.org/info/rfc5927>>.
- [RFC5936] Lewis, E. and A. Hoenes, Ed., "DNS Zone Transfer Protocol (AXFR)", RFC 5936, DOI 10.17487/RFC5936, June 2010, <<https://www.rfc-editor.org/info/rfc5936>>.
- [RFC5961] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", RFC 5961, DOI 10.17487/RFC5961, August 2010, <<https://www.rfc-editor.org/info/rfc5961>>.
- [RFC5966] Bellis, R., "DNS Transport over TCP - Implementation Requirements", RFC 5966, DOI 10.17487/RFC5966, August 2010, <<https://www.rfc-editor.org/info/rfc5966>>.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <<https://www.rfc-editor.org/info/rfc6298>>.
- [RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <<https://www.rfc-editor.org/info/rfc6762>>.

- [RFC6950] Peterson, J., Kolkman, O., Tschofenig, H., and B. Aboba, "Architectural Considerations on Application Features in the DNS", RFC 6950, DOI 10.17487/RFC6950, October 2013, <<https://www.rfc-editor.org/info/rfc6950>>.
- [RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <<https://www.rfc-editor.org/info/rfc7413>>.
- [RFC7477] Hardaker, W., "Child-to-Parent Synchronization in DNS", RFC 7477, DOI 10.17487/RFC7477, March 2015, <<https://www.rfc-editor.org/info/rfc7477>>.
- [RFC7534] Abley, J. and W. Sotomayor, "AS112 Nameserver Operations", RFC 7534, DOI 10.17487/RFC7534, May 2015, <<https://www.rfc-editor.org/info/rfc7534>>.
- [RFC7720] Blanchet, M. and L-J. Liman, "DNS Root Name Service Protocol and Deployment Requirements", BCP 40, RFC 7720, DOI 10.17487/RFC7720, December 2015, <<https://www.rfc-editor.org/info/rfc7720>>.
- [RFC7858] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", RFC 7858, DOI 10.17487/RFC7858, May 2016, <<https://www.rfc-editor.org/info/rfc7858>>.
- [RFC7901] Wouters, P., "CHAIN Query Requests in DNS", RFC 7901, DOI 10.17487/RFC7901, June 2016, <<https://www.rfc-editor.org/info/rfc7901>>.
- [RFC7918] Langley, A., Modadugu, N., and B. Moeller, "Transport Layer Security (TLS) False Start", RFC 7918, DOI 10.17487/RFC7918, August 2016, <<https://www.rfc-editor.org/info/rfc7918>>.
- [RFC8027] Hardaker, W., Gudmundsson, O., and S. Krishnaswamy, "DNSSEC Roadblock Avoidance", BCP 207, RFC 8027, DOI 10.17487/RFC8027, November 2016, <<https://www.rfc-editor.org/info/rfc8027>>.
- [RFC8094] Reddy, T., Wing, D., and P. Patil, "DNS over Datagram Transport Layer Security (DTLS)", RFC 8094, DOI 10.17487/RFC8094, February 2017, <<https://www.rfc-editor.org/info/rfc8094>>.



- [RFC8162] Hoffman, P. and J. Schlyter, "Using Secure DNS to Associate Certificates with Domain Names for S/MIME", RFC 8162, DOI 10.17487/RFC8162, May 2017, <<https://www.rfc-editor.org/info/rfc8162>>.
- [RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <<https://www.rfc-editor.org/info/rfc8200>>.
- [RFC8324] Klensin, J., "DNS Privacy, Authorization, Special Uses, Encoding, Characters, Matching, and Root Structure: Time for Another Look?", RFC 8324, DOI 10.17487/RFC8324, February 2018, <<https://www.rfc-editor.org/info/rfc8324>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [RFC8467] Mayrhofer, A., "Padding Policies for Extension Mechanisms for DNS (EDNS(0))", RFC 8467, DOI 10.17487/RFC8467, October 2018, <<https://www.rfc-editor.org/info/rfc8467>>.
- [RFC8482] Abley, J., Gudmundsson, O., Majkowski, M., and E. Hunt, "Providing Minimal-Sized Responses to DNS Queries That Have QTYPE=ANY", RFC 8482, DOI 10.17487/RFC8482, January 2019, <<https://www.rfc-editor.org/info/rfc8482>>.
- [RFC8483] Song, L., Ed., Liu, D., Vixie, P., Kato, A., and S. Kerr, "Yeti DNS Testbed", RFC 8483, DOI 10.17487/RFC8483, October 2018, <<https://www.rfc-editor.org/info/rfc8483>>.
- [RFC8484] Hoffman, P. and P. McManus, "DNS Queries over HTTPS (DoH)", RFC 8484, DOI 10.17487/RFC8484, October 2018, <<https://www.rfc-editor.org/info/rfc8484>>.
- [RFC8490] Bellis, R., Cheshire, S., Dickinson, J., Dickinson, S., Lemon, T., and T. Pusateri, "DNS Stateful Operations", RFC 8490, DOI 10.17487/RFC8490, March 2019, <<https://www.rfc-editor.org/info/rfc8490>>.
- [RFC8501] Howard, L., "Reverse DNS in IPv6 for Internet Service Providers", RFC 8501, DOI 10.17487/RFC8501, November 2018, <<https://www.rfc-editor.org/info/rfc8501>>.
- [RFC8806] Kumari, W. and P. Hoffman, "Running a Root Server Local to a Resolver", RFC 8806, DOI 10.17487/RFC8806, June 2020, <<https://www.rfc-editor.org/info/rfc8806>>.

- [RFC8906] Andrews, M. and R. Bellis, "A Common Operational Problem in DNS Servers: Failure to Communicate", BCP 231, RFC 8906, DOI 10.17487/RFC8906, September 2020, <<https://www.rfc-editor.org/info/rfc8906>>.
- [RFC8932] Dickinson, S., Overeinder, B., van Rijswijk-Deij, R., and A. Mankin, "Recommendations for DNS Privacy Service Operators", BCP 232, RFC 8932, DOI 10.17487/RFC8932, October 2020, <<https://www.rfc-editor.org/info/rfc8932>>.
- [RFC8945] Dupont, F., Morris, S., Vixie, P., Eastlake 3rd, D., Gudmundsson, O., and B. Wellington, "Secret Key Transaction Authentication for DNS (TSIG)", STD 93, RFC 8945, DOI 10.17487/RFC8945, November 2020, <<https://www.rfc-editor.org/info/rfc8945>>.
- [ROLL\_YOUR\_ROOT] Müller, M., Thomas, M., Wessels, D., Hardaker, W., Chung, T., Toorop, W., and R.v. Rijswijk-Deij, "Roll, Roll, Roll Your Root: A Comprehensive Analysis of the First Ever DNSSEC Root KSK Rollover", October 2019, <<https://dl.acm.org/doi/10.1145/3355369.3355570>>.
- [RRL] Vixie, P. and V. Schryver, "DNS Response Rate Limiting (DNS RRL)", ISC-TN 2012-1 Draft1, April 2012.
- [TDNS] Zhu, L., Heidemann, J., Wessels, D., Mankin, A., and N. Somaiya, "Connection-oriented DNS to Improve Privacy and Security", 2015.
- [TOYAMA] Toyama, K., Ishibashi, K., Ishino, M., Yoshimura, C., and K. Fujiwara, "DNS Anomalies and Their Impacts on DNS Cache Servers", NANOG 32 Reston, VA USA, 2004.
- [VERISIGN] Thomas, M. and D. Wessels, "An Analysis of TCP Traffic in Root Server DITL Data", DNS-OARC 2014 Fall Workshop Los Angeles, 2014.
- [WIKIPEDIA\_TFO] Wikipedia, "TCP Fast Open", 4 May 2018, <[https://en.wikipedia.org/wiki/TCP\\_Fast\\_Open](https://en.wikipedia.org/wiki/TCP_Fast_Open)>.

## Appendix A. Standards Related to DNS Transport over TCP

This section enumerates all known IETF RFC documents that are currently of status Internet Standard, Draft Standard, Proposed Standard, Informational, Best Current 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

This 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

This Proposed Standard [RFC1995] 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.

### A.4. IETF RFC 1996 - A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)

This Proposed Standard [RFC1996] suggests a primary server 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 primary server, and therefore a secondary server ought to be able to operate over both UDP and TCP.

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

This Proposed Standard [RFC2181] includes clarifying text on how a client should react to the TC bit set on responses. It is advised that the response should be discarded and the query resent using TCP.

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

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

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

This Proposed Standard [RFC3225] 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 EDNS(0) support for DNSSEC.

A.8. IETF RFC 3226 - DNSSEC and IPv6 A6 aware server/resolver message size requirements

Although updated by later DNSSEC RFCs, the Proposed Standard [RFC3226] strongly argues in favor of UDP messages instead of TCP, largely for performance reasons. The document declares EDNS(0) a requirement for DNSSEC servers and advocates that 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 than when this document was written, it refers to implementations that truncate data without setting the TC bit to encourage 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)

This Proposed Standard [RFC5936] 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 7534 - AS112 Nameserver Operations

[RFC7534] 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.15. IETF RFC 6762 - Multicast DNS

In this Proposed Standard [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 TC bit set is received, "[...] the querier SHOULD reissue its query using TCP in order to receive the larger response."

#### A.16. IETF RFC 6891 - Extension Mechanisms for DNS (EDNS(0))

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

A.17. 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.18. IETF RFC 7477 - Child-to-Parent Synchronization in DNS

This Proposed Standard [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.19. IETF RFC 7720 - DNS Root Name Service Protocol and Deployment Requirements

This Best Current Practice [RFC7720] declares root name service "MUST support UDP [RFC0768] and TCP [RFC0793] transport of DNS queries and responses."

A.20. IETF RFC 7766 - DNS Transport over TCP - Implementation Requirements

This Proposed Standard [RFC7766] instructs DNS implementers to provide support for carrying DNS over TCP messages in their software, and 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, but makes no recommendations to operators, which we seek to address here.

A.21. IETF RFC 7828 - The edns-tcp-keepalive EDNS(0) Option

This Proposed Standard [RFC7828] defines an EDNS(0) 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.22. IETF RFC 7858 - Specification for DNS over Transport Layer Security (TLS)

This Proposed Standard [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.23. IETF RFC 7873 - Domain Name System (DNS) Cookies

This Proposed Standard [RFC7873] describes an EDNS(0) option to provide additional protection against query and answer forgery. This specification mentions DNS over TCP as an alternative 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.24. IETF RFC 7901 - CHAIN Query Requests in DNS

This Experimental specification [RFC7901] describes an EDNS(0) 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.25. IETF RFC 8027 - DNSSEC Roadblock Avoidance

This Best Current Practice [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", the scope of the document is limited to detection and mitigation techniques to avoid so-called DNSSEC roadblocks.

A.26. 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.27. 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.28. 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.29. 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.30. IETF RFC 8482 - Providing Minimal-Sized Responses to DNS Queries That Have QTYPE=ANY

[RFC8482] is a Proposed Standard that describes alternative ways that DNS servers can respond to queries of type ANY, which are sometimes used to provide amplification in DDoS attacks. The specification notes that responders may behave differently, depending on the transport. For example, minimal-sized responses may be used over UDP transport, while full responses may be given over TCP.



## 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 Proposed Standard [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 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 Proposed Standard [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.

## A.34. IETF RFC 8501 - Reverse DNS in IPv6 for Internet Service Providers

This Informational document [RFC8501] identifies potential operational challenges with Dynamic DNS including denial-of-service threats. The document suggests TCP may provide some advantages, but that updating hosts would need to be explicitly configured to use TCP instead of UDP.

## A.35. IETF RFC 8806 - Running a Root Server Local to a Resolver

This Informational document [RFC8806] describes how to obtain and operate a local copy of the root zone with examples showing how to pull from authoritative sources using a DNS over TCP zone transfer.

## A.36. IETF RFC 8906 - A Common Operational Problem in DNS Servers: Failure to Communicate

This Best Current Practice document [RFC8906] discusses a number of DNS operational failure scenarios and how to avoid them. This includes discussions involving DNS over TCP queries, EDNS over TCP, and a testing methodology that includes a section on verifying DNS over TCP functionality.

## A.37. IETF RFC 8932 - Recommendations for DNS Privacy Service Operators

This Best Current Practice document [RFC8932] presents privacy considerations to DNS privacy service operators. These mechanisms sometimes include the use of TCP and are therefore susceptible to information leakage such as TCP-based fingerprinting. This document also references a draft version of this document.

## A.38. IETF RFC 8945 - Secret Key Transaction Authentication for DNS (TSIG)

This Internet Standard [RFC8945] recommends a client use TCP if truncated TSIG messages are received.

## Authors' Addresses

John Kristoff  
DataPlane.org  
Chicago, IL 60605  
United States of America

Phone: +1 312 493 0305  
Email: [jtk@dataplane.org](mailto:jtk@dataplane.org)  
URI: <https://dataplane.org/jtk/>

Duane Wessels  
Verisign  
12061 Bluemont Way  
Reston, VA 20190  
United States of America

Phone: +1 703 948 3200  
Email: [dwessels@verisign.com](mailto:dwessels@verisign.com)  
URI: <https://verisign.com>

QUIC  
Internet-Draft  
Intended status: Standards Track  
Expires: 19 July 2021

J. Iyengar, Ed.  
Fastly  
I. Swett, Ed.  
Google  
15 January 2021

QUIC Loss Detection and Congestion Control  
draft-ietf-quic-recovery-34

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](mailto:quic@ietf.org) (<mailto:quic@ietf.org>)), which is archived at [https://mailarchive.ietf.org/arch/search/?email\\_list=quic](https://mailarchive.ietf.org/arch/search/?email_list=quic).

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

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 19 July 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

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

## Table of Contents

1. Introduction . . . . .	4
2. Conventions and Definitions . . . . .	4
3. Design of the QUIC Transmission Machinery . . . . .	5
4. Relevant Differences Between QUIC and TCP . . . . .	6
4.1. Separate Packet Number Spaces . . . . .	6
4.2. Monotonically Increasing Packet Numbers . . . . .	6
4.3. Clearer Loss Epoch . . . . .	7
4.4. No Reneging . . . . .	7
4.5. More ACK Ranges . . . . .	7
4.6. Explicit Correction For Delayed Acknowledgments . . . . .	7
4.7. Probe Timeout Replaces RTO and TLP . . . . .	7
4.8. The Minimum Congestion Window is Two Packets . . . . .	8
5. Estimating the Round-Trip Time . . . . .	8
5.1. Generating RTT samples . . . . .	8
5.2. Estimating min_rtt . . . . .	9
5.3. Estimating smoothed_rtt and rttvar . . . . .	10
6. Loss Detection . . . . .	12
6.1. Acknowledgment-Based Detection . . . . .	12
6.1.1. Packet Threshold . . . . .	13
6.1.2. Time Threshold . . . . .	13
6.2. Probe Timeout . . . . .	14
6.2.1. Computing PTO . . . . .	14
6.2.2. Handshakes and New Paths . . . . .	16
6.2.3. Speeding Up Handshake Completion . . . . .	17
6.2.4. Sending Probe Packets . . . . .	18
6.3. Handling Retry Packets . . . . .	19
6.4. Discarding Keys and Packet State . . . . .	19
7. Congestion Control . . . . .	20
7.1. Explicit Congestion Notification . . . . .	20
7.2. Initial and Minimum Congestion Window . . . . .	21
7.3. Congestion Control States . . . . .	21
7.3.1. Slow Start . . . . .	22
7.3.2. Recovery . . . . .	22
7.3.3. Congestion Avoidance . . . . .	23
7.4. Ignoring Loss of Undecryptable Packets . . . . .	23
7.5. Probe Timeout . . . . .	24
7.6. Persistent Congestion . . . . .	24

7.6.1. Duration . . . . .	24
7.6.2. Establishing Persistent Congestion . . . . .	25
7.6.3. Example . . . . .	26
7.7. Pacing . . . . .	27
7.8. Under-utilizing the Congestion Window . . . . .	28
8. Security Considerations . . . . .	28
8.1. Loss and Congestion Signals . . . . .	28
8.2. Traffic Analysis . . . . .	28
8.3. Misreporting ECN Markings . . . . .	28
9. IANA Considerations . . . . .	29
10. References . . . . .	29
10.1. Normative References . . . . .	29
10.2. Informative References . . . . .	30
Appendix A. Loss Recovery Pseudocode . . . . .	32
A.1. Tracking Sent Packets . . . . .	32
A.1.1. Sent Packet Fields . . . . .	32
A.2. Constants of Interest . . . . .	33
A.3. Variables of interest . . . . .	33
A.4. Initialization . . . . .	34
A.5. On Sending a Packet . . . . .	34
A.6. On Receiving a Datagram . . . . .	35
A.7. On Receiving an Acknowledgment . . . . .	35
A.8. Setting the Loss Detection Timer . . . . .	37
A.9. On Timeout . . . . .	39
A.10. Detecting Lost Packets . . . . .	39
A.11. Upon Dropping Initial or Handshake Keys . . . . .	40
Appendix B. Congestion Control Pseudocode . . . . .	41
B.1. Constants of interest . . . . .	41
B.2. Variables of interest . . . . .	41
B.3. Initialization . . . . .	42
B.4. On Packet Sent . . . . .	42
B.5. On Packet Acknowledgment . . . . .	42
B.6. On New Congestion Event . . . . .	43
B.7. Process ECN Information . . . . .	44
B.8. On Packets Lost . . . . .	44
B.9. Removing Discarded Packets From Bytes In Flight . . . . .	44
Appendix C. Change Log . . . . .	45
C.1. Since draft-ietf-quic-recovery-32 . . . . .	45
C.2. Since draft-ietf-quic-recovery-31 . . . . .	45
C.3. Since draft-ietf-quic-recovery-30 . . . . .	45
C.4. Since draft-ietf-quic-recovery-29 . . . . .	45
C.5. Since draft-ietf-quic-recovery-28 . . . . .	46
C.6. Since draft-ietf-quic-recovery-27 . . . . .	46
C.7. Since draft-ietf-quic-recovery-26 . . . . .	46
C.8. Since draft-ietf-quic-recovery-25 . . . . .	46
C.9. Since draft-ietf-quic-recovery-24 . . . . .	46
C.10. Since draft-ietf-quic-recovery-23 . . . . .	46
C.11. Since draft-ietf-quic-recovery-22 . . . . .	47

C.12. Since draft-ietf-quic-recovery-21 . . . . .	47
C.13. Since draft-ietf-quic-recovery-20 . . . . .	47
C.14. Since draft-ietf-quic-recovery-19 . . . . .	47
C.15. Since draft-ietf-quic-recovery-18 . . . . .	48
C.16. Since draft-ietf-quic-recovery-17 . . . . .	48
C.17. Since draft-ietf-quic-recovery-16 . . . . .	49
C.18. Since draft-ietf-quic-recovery-14 . . . . .	49
C.19. Since draft-ietf-quic-recovery-13 . . . . .	49
C.20. Since draft-ietf-quic-recovery-12 . . . . .	50
C.21. Since draft-ietf-quic-recovery-11 . . . . .	50
C.22. Since draft-ietf-quic-recovery-10 . . . . .	50
C.23. Since draft-ietf-quic-recovery-09 . . . . .	50
C.24. Since draft-ietf-quic-recovery-08 . . . . .	50
C.25. Since draft-ietf-quic-recovery-07 . . . . .	50
C.26. Since draft-ietf-quic-recovery-06 . . . . .	51
C.27. Since draft-ietf-quic-recovery-05 . . . . .	51
C.28. Since draft-ietf-quic-recovery-04 . . . . .	51
C.29. Since draft-ietf-quic-recovery-03 . . . . .	51
C.30. Since draft-ietf-quic-recovery-02 . . . . .	51
C.31. Since draft-ietf-quic-recovery-01 . . . . .	51
C.32. Since draft-ietf-quic-recovery-00 . . . . .	51
C.33. Since draft-iyengar-quic-loss-recovery-01 . . . . .	51
Appendix D. Contributors . . . . .	52
Acknowledgments . . . . .	52
Authors' Addresses . . . . .	52

## 1. Introduction

QUIC is a secure general-purpose transport protocol, described in [QUIC-TRANSPORT]). This document describes loss detection and congestion control mechanisms for QUIC.

## 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-eliciting frames: All frames other than ACK, PADDING, and CONNECTION\_CLOSE are considered ack-eliciting.

Ack-eliciting packets: Packets that contain ack-eliciting frames elicit an ACK from the receiver within the maximum acknowledgment delay and are called ack-eliciting packets.

In-flight packets: Packets are considered in-flight when they are ack-eliciting or contain a PADDING frame, and they have been sent but are not acknowledged, declared lost, or discarded along with old keys.

### 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 Section 12.3 in [QUIC-TRANSPORT]. Packet numbers never repeat within a packet number space for the lifetime of a connection. Packet numbers are sent in monotonically increasing order within a space, preventing ambiguity. It is permitted for some packet numbers to never be used, leaving intentional gaps.

This design obviates the need for disambiguating between transmissions and retransmissions; this 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 acknowledgment.
- \* Packets containing frames besides ACK or CONNECTION\_CLOSE frames count toward congestion control limits and are considered in-flight.
- \* PADDING frames cause packets to contribute toward bytes in flight without directly causing an acknowledgment to be sent.

#### 4. 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. However, protocol differences between QUIC and TCP contribute to algorithmic differences. These protocol differences are briefly described below.

##### 4.1. 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 acknowledgment 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.

##### 4.2. Monotonically Increasing Packet Numbers

TCP conflates transmission order at the sender with delivery order at the receiver, resulting in the retransmission ambiguity problem ([RETRANSMISSION]). QUIC separates transmission order from delivery order: packet numbers indicate transmission order, and delivery order is determined by the stream offsets in 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 includes 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.



#### 4.3. Clearer Loss Epoch

QUIC starts a loss epoch when a packet is lost. The loss epoch ends when any packet sent after the start of the epoch is acknowledged. TCP waits for the gap in the sequence number space to be filled, and so if a segment is lost multiple times in a row, the loss epoch may not end for several round trips. Because both should reduce their congestion windows only once per epoch, QUIC will do it once for every round trip that experiences loss, while TCP may only do it once across multiple round trips.

#### 4.4. No Reneging

QUIC ACK frames contain information similar to that in TCP Selective Acknowledgements (SACKs, [RFC2018]). However, QUIC does not allow a packet acknowledgement to be reneged, greatly simplifying implementations on both sides and reducing memory pressure on the sender.

#### 4.5. 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.

#### 4.6. Explicit Correction For Delayed Acknowledgments

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 13.2 of [QUIC-TRANSPORT].

#### 4.7. Probe Timeout Replaces RTO and TLP

QUIC uses a probe timeout (PTO; see Section 6.2), with a timer based on TCP's RTO computation; see [RFC6297]. QUIC's PTO includes the peer's maximum expected acknowledgment delay instead of using a fixed minimum timeout.

Similar to the RACK-TLP loss detection algorithm for TCP ([RACK]), QUIC does not collapse the congestion window when the PTO expires, since a single packet loss at the tail does not indicate persistent congestion. Instead, QUIC collapses the congestion window when persistent congestion is declared; see Section 7.6. In doing this, QUIC avoids unnecessary congestion window reductions, obviating the need for correcting mechanisms such as F-RTO ([RFC5682]). Since QUIC does not collapse the congestion window on a PTO expiration, a QUIC

sender is not limited from sending more in-flight packets after a PTO expiration if it still has available congestion window. This occurs when a sender is application-limited and the PTO timer expires. This is more aggressive than TCP's RTO mechanism when application-limited, but identical when not application-limited.

QUIC allows probe packets to temporarily exceed the congestion window whenever the timer expires.

#### 4.8. The Minimum Congestion Window is Two Packets

TCP uses a minimum congestion window of one packet. However, loss of that single packet means that the sender needs to waiting for a PTO (Section 6.2) to recover, which can be much longer than a round-trip time. Sending a single ack-eliciting packet also increases the chances of incurring additional latency when a receiver delays its acknowledgment.

QUIC therefore recommends that the minimum congestion window be two packets. While this increases network load, it is considered safe, since the sender will still reduce its sending rate exponentially under persistent congestion (Section 6.2).

### 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 (see Section 13.2 of [QUIC-TRANSPORT]) to generate a statistical description of the network path's RTT. An endpoint computes the following three values for each path: the minimum value over a period of time (`min_rtt`), an exponentially-weighted moving average (`smoothed_rtt`), and the mean deviation (referred to as "variation" in the rest of this document) 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:

```
latest_rtt = ack_time - 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 acknowledgment delays for only the largest acknowledged packet in an ACK frame. While the reported acknowledgment 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 for a single 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 usually does not send an ACK frame when only non-ack-eliciting packets are received. Therefore an ACK frame that contains acknowledgments for only non-ack-eliciting packets could include an arbitrarily large ACK Delay value. 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 sender's estimate of the minimum RTT observed for a given network path over a period of time. In this document, `min_rtt` is used by loss detection to reject implausibly small `rtt` samples.

`min_rtt` MUST be set to the `latest_rtt` on the first RTT sample. `min_rtt` MUST be set to the lesser of `min_rtt` and `latest_rtt` (Section 5.1) on all other samples.

An endpoint uses only locally observed times in computing the `min_rtt` and does not adjust for acknowledgment delays reported by the peer. 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.

The RTT for a network path may change over time. If a path's actual RTT decreases, the `min_rtt` will adapt immediately on the first low sample. If the path's actual RTT increases however, the `min_rtt` will not adapt to it, allowing future RTT samples that are smaller than the new RTT to be included in `smoothed_rtt`.

Endpoints SHOULD set the `min_rtt` to the newest RTT sample after persistent congestion is established. This is to allow a connection to reset its estimate of `min_rtt` and `smoothed_rtt` (Section 5.3) after a disruptive network event, and because it is possible that an increase in path delay resulted in persistent congestion being incorrectly declared.

Endpoints MAY re-establish the `min_rtt` at other times in the connection, such as when traffic volume is low and an acknowledgment is received with a low acknowledgment delay. Implementations SHOULD NOT refresh the `min_rtt` value too often, since the actual minimum RTT of the path is not frequently observable.

### 5.3. Estimating `smoothed_rtt` and `rttvar`

`smoothed_rtt` is an exponentially-weighted moving average of an endpoint's RTT samples, and `rttvar` estimates the variation in the RTT samples using a mean variation.

The calculation of `smoothed_rtt` uses RTT samples after adjusting them for acknowledgment delays. These delays are decoded from the ACK Delay field of ACK frames as described in Section 19.3 of [QUIC-TRANSPORT].

The peer might report acknowledgment delays that are larger than the peer's `max_ack_delay` during the handshake (Section 13.2.1 of [QUIC-TRANSPORT]). To account for this, the endpoint SHOULD ignore `max_ack_delay` until the handshake is confirmed, as defined in Section 4.1.2 of [QUIC-TLS]. When they occur, these large acknowledgment delays are likely to be non-repeating and limited to the handshake. The endpoint can therefore use them without limiting them to the `max_ack_delay`, avoiding unnecessary inflation of the RTT estimate.

Note that a large acknowledgment delay can result in a substantially inflated `smoothed_rtt`, if there is either an error in the peer's reporting of the acknowledgment delay or in the endpoint's `min_rtt` estimate. Therefore, prior to handshake confirmation, an endpoint MAY ignore RTT samples if adjusting the RTT sample for acknowledgment delay causes the sample to be less than the `min_rtt`.

After the handshake is confirmed, any acknowledgment delays reported by the peer that are greater than the peer's `max_ack_delay` are attributed to unintentional but potentially repeating delays, such as scheduler latency at the peer or loss of previous acknowledgments. Excess delays could also be due to a non-compliant receiver. Therefore, these extra delays are considered effectively part of path delay and incorporated into the RTT estimate.

Therefore, when adjusting an RTT sample using peer-reported acknowledgment delays, an endpoint:

- \* MAY ignore the acknowledgment delay for Initial packets, since these acknowledgments are not delayed by the peer (Section 13.2.1 of [QUIC-TRANSPORT]);
- \* SHOULD ignore the peer's `max_ack_delay` until the handshake is confirmed;
- \* MUST use the lesser of the acknowledgment delay and the peer's `max_ack_delay` after the handshake is confirmed; and
- \* MUST NOT subtract the acknowledgment delay from the RTT sample if the resulting value is smaller than the `min_rtt`. This limits the underestimation of the `smoothed_rtt` due to a misreporting peer.

Additionally, an endpoint might postpone the processing of acknowledgments when the corresponding decryption keys are not immediately available. For example, a client might receive an acknowledgment for a 0-RTT packet that it cannot decrypt because 1-RTT packet protection keys are not yet available to it. In such cases, an endpoint SHOULD subtract such local delays from its RTT sample until the handshake is confirmed.

Similar to [RFC6298], `smoothed_rtt` and `rttvar` are computed as follows.

An endpoint initializes the RTT estimator during connection establishment and when the estimator is reset during connection migration; see Section 9.4 of [QUIC-TRANSPORT]. Before any RTT samples are available for a new path or when the estimator is reset, the estimator is initialized using the initial RTT; see Section 6.2.2.

`smoothed_rtt` and `rttvar` are initialized as follows, where `kInitialRtt` contains the initial RTT value:

```
smoothed_rtt = kInitialRtt
rttvar = kInitialRtt / 2
```

RTT samples for the network path are recorded in `latest_rtt`; see Section 5.1. On the first RTT sample after initialization, the estimator is reset using that sample. This ensures that the estimator retains no history of past samples.

On the first RTT sample after initialization, `smoothed_rtt` and `rttvar` are set as follows:

```
smoothed_rtt = latest_rtt
rttvar = latest_rtt / 2
```

On subsequent RTT samples, smoothed\_rtt and rttvar evolve as follows:

```
ack_delay = decoded acknowledgment delay from ACK frame
if (handshake confirmed):
    ack_delay = min(ack_delay, max_ack_delay)
adjusted_rtt = latest_rtt
if (min_rtt + ack_delay < latest_rtt):
    adjusted_rtt = latest_rtt - ack_delay
smoothed_rtt = 7/8 * smoothed_rtt + 1/8 * adjusted_rtt
rttvar_sample = abs(smoothed_rtt - adjusted_rtt)
rttvar = 3/4 * rttvar + 1/4 * rttvar_sample
```

## 6. Loss Detection

QUIC senders use acknowledgments to detect lost packets, and a probe time out (see Section 6.2) to ensure acknowledgments are received. 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 discarding the frame. For more information, see Section 13.3 of [QUIC-TRANSPORT].

Loss detection is separate per packet number space, unlike RTT measurement and congestion control, because RTT and congestion control are properties of the path, whereas loss detection also relies upon key availability.

### 6.1. Acknowledgment-Based Detection

Acknowledgment-based loss detection implements the spirit of TCP's Fast Retransmit ([RFC5681]), Early Retransmit ([RFC5827]), FACK ([FACK]), SACK loss recovery ([RFC6675]), and RACK-TLP ([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.
- \* The packet was sent kPacketThreshold packets before an acknowledged packet (Section 6.1.1), or it was sent long enough in the past (Section 6.1.2).

The acknowledgment indicates that a packet sent later was delivered, and 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 can detect spurious retransmissions and increase the reordering threshold in packets or time to reduce future spurious retransmissions and loss events. Implementations with adaptive time thresholds 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]). In order to remain similar to TCP, implementations SHOULD NOT use a packet threshold less than 3; see [RFC5681].

Some networks may exhibit higher degrees of packet reordering, causing a sender to detect spurious losses. Additionally, packet reordering could be more common with QUIC than TCP, because network elements that could observe and reorder TCP packets cannot do that for QUIC, because QUIC packet numbers are encrypted. Algorithms that increase the reordering threshold after spuriously detecting losses, such as RACK [RACK], have proven to be useful in TCP and are expected to be at least as useful in QUIC.

#### 6.1.2. Time Threshold

Once a later packet within the same packet number space has been acknowledged, an endpoint SHOULD declare an earlier packet lost if it was sent a threshold amount of time in the past. To avoid declaring packets as lost too early, this time threshold MUST be set to at least the local timer granularity, as indicated by the `kGranularity` constant. The time threshold is:

$$\max(kTimeThreshold * \max(smoothed\_rtt, latest\_rtt), kGranularity)$$

If packets sent prior to the largest acknowledged packet cannot yet be declared lost, then a timer SHOULD be set for the remaining time.

Using  $\max(smoothed\_rtt, latest\_rtt)$  protects from the two following cases:

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

The RECOMMENDED time threshold (`kTimeThreshold`), expressed as a round-trip time multiplier, is  $9/8$ . The RECOMMENDED value of the timer granularity (`kGranularity`) is 1ms.

Note: TCP's RACK ([RACK]) specifies a slightly larger threshold, equivalent to  $5/4$ , for a similar purpose. Experience with QUIC shows that  $9/8$  works well.

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

## 6.2. Probe Timeout

A Probe Timeout (PTO) triggers sending one or two probe datagrams when ack-eliciting packets are not acknowledged within the expected period of time or the server may not have validated the client's address. A PTO enables a connection to recover from loss of tail packets or acknowledgments.

As with loss detection, the probe timeout is per packet number space. That is, a PTO value is computed per packet number space.

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

The PTO algorithm used in QUIC implements the reliability functions of Tail Loss Probe [RACK], RTO [RFC5681], and F-RTO algorithms for TCP [RFC5682]. The timeout computation is based on TCP's retransmission timeout period [RFC6298].

### 6.2.1. Computing PTO

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



$$PTO = smoothed\_rtt + \max(4 * rttvar, kGranularity) + max\_ack\_delay$$

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

When the PTO is armed for Initial or Handshake packet number spaces, the `max_ack_delay` in the PTO period computation is set to 0, since the peer is expected to not delay these packets intentionally; see 13.2.1 of [QUIC-TRANSPORT].

The PTO period MUST be at least `kGranularity`, to avoid the timer expiring immediately.

When ack-eliciting packets in multiple packet number spaces are in flight, the timer MUST be set to the earlier value of the Initial and Handshake packet number spaces.

An endpoint MUST NOT set its PTO timer for the application data packet number space until the handshake is confirmed. Doing so prevents the endpoint from retransmitting information in packets when either the peer does not yet have the keys to process them or the endpoint does not yet have the keys to process their acknowledgments. For example, this can happen when a client sends 0-RTT packets to the server; it does so without knowing whether the server will be able to decrypt them. Similarly, this can happen when a server sends 1-RTT packets before confirming that the client has verified the server's certificate and can therefore read these 1-RTT packets.

A sender SHOULD restart its PTO timer every time an ack-eliciting packet is sent or acknowledged, or when Initial or Handshake keys are discarded (Section 4.9 of [QUIC-TLS]). This ensures the PTO is always set based on the latest estimate of the round-trip time and for the correct packet across packet number spaces.

When a PTO timer expires, the PTO backoff MUST be increased, resulting in the PTO period being set to twice its current value. The PTO backoff factor is reset when an acknowledgment is received, except in the following case. A server might take longer to respond to packets during the handshake than otherwise. To protect such a server from repeated client probes, the PTO backoff is not reset at a client that is not yet certain that the server has finished validating the client's address. That is, a client does not reset the PTO backoff factor on receiving acknowledgments in Initial packets.

This exponential reduction in the sender's rate is important because consecutive PTOs might be caused by loss of packets or acknowledgments due to severe congestion. Even when there are acknowledging packets in-flight in multiple packet number spaces, the exponential increase in probe timeout occurs across all spaces to prevent excess load on the network. For example, a timeout in the Initial packet number space doubles the length of the timeout in the Handshake packet number space.

The total length of time over which consecutive PTOs expire is limited by the idle timeout.

The PTO timer **MUST NOT** be set if a timer is set for time threshold loss detection; see Section 6.1.2. A timer that is set for time threshold loss detection will expire earlier than the PTO timer in most cases and is less likely to spuriously retransmit data.

#### 6.2.2. Handshakes and New Paths

Resumed connections over the same network **MAY** use the previous connection's final smoothed RTT value as the resumed connection's initial RTT. When no previous RTT is available, the initial RTT **SHOULD** be set to 333ms. This results in handshakes starting with a PTO of 1 second, as recommended for TCP's initial retransmission timeout; see Section 2 of [RFC6298].

A connection **MAY** use the delay between sending a `PATH_CHALLENGE` and receiving a `PATH_RESPONSE` to set the initial RTT (see `kInitialRtt` in Appendix A.2) for a new path, but the delay **SHOULD NOT** be considered an RTT sample.

Initial packets and Handshake packets could be never acknowledged, but they are removed from bytes in flight when the Initial and Handshake keys are discarded, as described below in Section 6.4. When Initial or Handshake keys are discarded, the PTO and loss detection timers **MUST** be reset, because discarding keys indicates forward progress and the loss detection timer might have been set for a now discarded packet number space.

#### 6.2.2.1. Before Address Validation

Until the server has validated the client's address on the path, the amount of data it can send is limited to three times the amount of data received, as specified in Section 8.1 of [QUIC-TRANSPORT]. If no additional data can be sent, the server's PTO timer MUST NOT be armed until datagrams have been received from the client, because packets sent on PTO count against the anti-amplification limit. Note that the server could fail to validate the client's address even if 0-RTT is accepted.

Since the server could be blocked until more datagrams are received from the client, it is the client's responsibility to send packets to unblock the server until it is certain that the server has finished its address validation (see Section 8 of [QUIC-TRANSPORT]). That is, the client MUST set the probe timer if the client has not received an acknowledgment for any of its Handshake packets and the handshake is not confirmed (see Section 4.1.2 of [QUIC-TLS]), even if there are no packets in flight. When the PTO fires, the client MUST send a Handshake packet if it has Handshake keys, otherwise it MUST send an Initial packet in a UDP datagram with a payload of at least 1200 bytes.

#### 6.2.3. Speeding Up Handshake Completion

When a server receives an Initial packet containing duplicate CRYPTO data, it can assume the client did not receive all of the server's CRYPTO data sent in Initial packets, or the client's estimated RTT is too small. When a client receives Handshake or 1-RTT packets prior to obtaining Handshake keys, it may assume some or all of the server's Initial packets were lost.

To speed up handshake completion under these conditions, an endpoint MAY, for a limited number of times per connection, send a packet containing unacknowledged CRYPTO data earlier than the PTO expiry, subject to the address validation limits in Section 8.1 of [QUIC-TRANSPORT]. Doing so at most once for each connection is adequate to quickly recover from a single packet loss. An endpoint that always retransmits packets in response to receiving packets that it cannot process risks creating an infinite exchange of packets.

Endpoints can also use coalesced packets (see Section 12.2 of [QUIC-TRANSPORT]) to ensure that each datagram elicits at least one acknowledgment. For example, a client can coalesce an Initial packet containing PING and PADDING frames with a 0-RTT data packet and a server can coalesce an Initial packet containing a PING frame with one or more packets in its first flight.

#### 6.2.4. Sending Probe Packets

When a PTO timer expires, a sender **MUST** send at least one ack-eliciting packet in the packet number space as a probe. An endpoint **MAY** send up to two full-sized datagrams containing ack-eliciting packets, to avoid an expensive consecutive PTO expiration due to a single lost datagram, or transmit data from multiple packet number spaces. All probe packets sent on a PTO **MUST** be ack-eliciting.

In addition to sending data in the packet number space for which the timer expired, the sender **SHOULD** send ack-eliciting packets from other packet number spaces with in-flight data, coalescing packets if possible. This is particularly valuable when the server has both Initial and Handshake data in-flight or the client has both Handshake and Application Data in-flight, because the peer might only have receive keys for one of the two packet number spaces.

If the sender wants to elicit a faster acknowledgment on PTO, it can skip a packet number to eliminate the acknowledgment delay.

An endpoint **SHOULD** include new data in packets that are sent on PTO expiration. Previously sent data **MAY** be sent if no new data can be sent. Implementations **MAY** use alternative strategies for determining the content of probe packets, including sending new or retransmitted data based on the application's priorities.

It is possible 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. 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.

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. Handling Retry Packets

A Retry packet causes a client to send another Initial packet, effectively restarting the connection process. A Retry packet indicates that the Initial was received, but not processed. A Retry packet cannot be treated as an acknowledgment, because it does not indicate that a packet was processed or specify the packet number.

Clients that receive a Retry packet reset congestion control and loss recovery state, including resetting any pending timers. Other connection state, in particular cryptographic handshake messages, is retained; see Section 17.2.5 of [QUIC-TRANSPORT].

The client MAY 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 in place of its default for the initial RTT estimate.

### 6.4. Discarding Keys and Packet State

When Initial and Handshake 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 acknowledgments cannot be processed. 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. However, Initial and Handshake secrets are discarded as soon as handshake and 1-RTT keys are proven to be available to both client and server; see Section 4.9.1 of [QUIC-TLS].

## 7. Congestion Control

This document specifies a sender-side congestion controller for QUIC similar to TCP NewReno ([RFC6582]).

The signals QUIC provides for congestion control are generic and are designed to support different sender-side algorithms. A sender can unilaterally choose a different algorithm to use, such as Cubic ([RFC8312]).

If a sender uses a different controller than that specified in this document, the chosen controller MUST conform to the congestion control guidelines specified in Section 3.1 of [RFC8085].

Similar to TCP, packets containing only ACK frames do not count towards bytes in flight and are not congestion controlled. Unlike TCP, QUIC can detect the loss of these packets and MAY use that information to adjust the congestion controller or the rate of ACK-only packets being sent, but this document does not describe a mechanism for doing so.

The algorithm in this document specifies and uses the controller's congestion window in bytes.

An endpoint MUST NOT send a packet if it would cause `bytes_in_flight` (see Appendix B.2) to be larger than the congestion window, unless the packet is sent on a PTO timer expiration (see Section 6.2) or when entering recovery (see Section 7.3.2).

### 7.1. Explicit Congestion Notification

If a path has been validated to support ECN ([RFC3168], [RFC8311]), QUIC treats a Congestion Experienced (CE) codepoint in the IP header as a signal of congestion. This document specifies an endpoint's response when the peer-reported ECN-CE count increases; see Section 13.4.2 of [QUIC-TRANSPORT].

## 7.2. Initial and Minimum Congestion Window

QUIC begins every connection in slow start with the congestion window set to an initial value. Endpoints SHOULD use an initial congestion window of 10 times the maximum datagram size (`max_datagram_size`), while limiting the window to the larger of 14720 bytes or twice the maximum datagram size. This follows the analysis and recommendations in [RFC6928], increasing the byte limit to account for the smaller 8-byte overhead of UDP compared to the 20-byte overhead for TCP.

If the maximum datagram size changes during the connection, the initial congestion window SHOULD be recalculated with the new size. If the maximum datagram size is decreased in order to complete the handshake, the congestion window SHOULD be set to the new initial congestion window.

Prior to validating the client's address, the server can be further limited by the anti-amplification limit as specified in Section 8.1 of [QUIC-TRANSPORT]. Though the anti-amplification limit can prevent the congestion window from being fully utilized and therefore slow down the increase in congestion window, it does not directly affect the congestion window.

The minimum congestion window is the smallest value the congestion window can decrease to as a response to loss, increase in the peer-reported ECN-CE count, or persistent congestion. The RECOMMENDED value is  $2 * \text{max\_datagram\_size}$ .

## 7.3. Congestion Control States

The NewReno congestion controller described in this document has three distinct states, as shown in Figure 1.

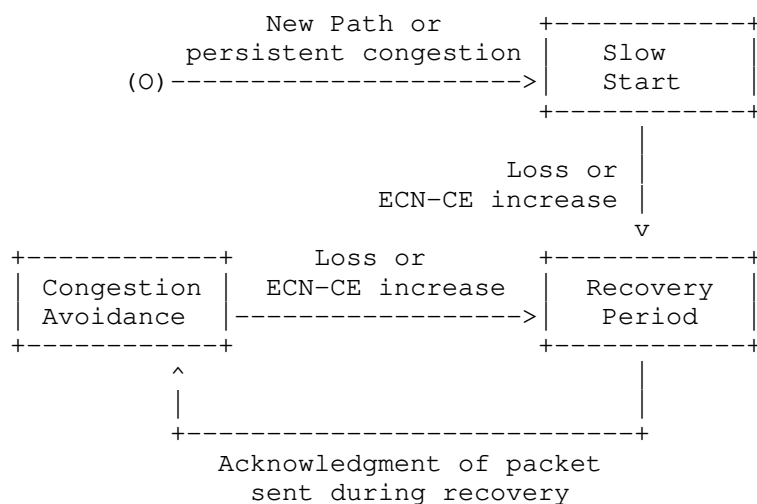


Figure 1: Congestion Control States and Transitions

These states and the transitions between them are described in subsequent sections.

#### 7.3.1. Slow Start

A NewReno sender is in slow start any time the congestion window is below the slow start threshold. A sender begins in slow start because the slow start threshold is initialized to an infinite value.

While a sender is in slow start, the congestion window increases by the number of bytes acknowledged when each acknowledgment is processed. This results in exponential growth of the congestion window.

The sender MUST exit slow start and enter a recovery period when a packet is lost or when the ECN-CE count reported by its peer increases.

A sender re-enters slow start any time the congestion window is less than the slow start threshold, which only occurs after persistent congestion is declared.

#### 7.3.2. Recovery

A NewReno sender enters a recovery period when it detects the loss of a packet or the ECN-CE count reported by its peer increases. A sender that is already in a recovery period stays in it and does not re-enter it.



On entering a recovery period, a sender MUST set the slow start threshold to half the value of the congestion window when loss is detected. The congestion window MUST be set to the reduced value of the slow start threshold before exiting the recovery period.

Implementations MAY reduce the congestion window immediately upon entering a recovery period or use other mechanisms, such as Proportional Rate Reduction ([PRR]), to reduce the congestion window more gradually. If the congestion window is reduced immediately, a single packet can be sent prior to reduction. This speeds up loss recovery if the data in the lost packet is retransmitted and is similar to TCP as described in Section 5 of [RFC6675].

The recovery period aims to limit congestion window reduction to once per round trip. Therefore during a recovery period, the congestion window does not change in response to new losses or increases in the ECN-CE count.

A recovery period ends and the sender enters congestion avoidance when a packet sent during the recovery period is acknowledged. This is slightly different from TCP's definition of recovery, which ends when the lost segment that started recovery is acknowledged ([RFC5681]).

#### 7.3.3. Congestion Avoidance

A NewReno sender is in congestion avoidance any time the congestion window is at or above the slow start threshold and not in a recovery period.

A sender in congestion avoidance uses an Additive Increase Multiplicative Decrease (AIMD) approach that MUST limit the increase to the congestion window to at most one maximum datagram size for each congestion window that is acknowledged.

The sender exits congestion avoidance and enters a recovery period when a packet is lost or when the ECN-CE count reported by its peer increases.

#### 7.4. Ignoring Loss of Undecryptable Packets

During the handshake, some packet protection keys might not be available when a packet arrives and the receiver can choose to drop the packet. 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 have arrived before the peer had packet protection keys to process those packets.

Endpoints MUST NOT ignore the loss of packets that were sent after the earliest acknowledged packet in a given packet number space.

#### 7.5. 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 acknowledgment is received that establishes loss or delivery of packets.

#### 7.6. Persistent Congestion

When a sender establishes loss of all packets sent over a long enough duration, the network is considered to be experiencing persistent congestion.

##### 7.6.1. Duration

The persistent congestion duration is computed as follows:

$$(\text{smoothed\_rtt} + \max(4 * \text{rttvar}, k\text{Granularity}) + \text{max\_ack\_delay}) * k\text{PersistentCongestionThreshold}$$

Unlike the PTO computation in Section 6.2, this duration includes the `max_ack_delay` irrespective of the packet number spaces in which losses are established.

This duration allows a sender to send as many packets before establishing persistent congestion, including some in response to PTO expiration, as TCP does with Tail Loss Probes ([RACK]) and a Retransmission Timeout ([RFC5681]).

Larger values of `kPersistentCongestionThreshold` cause the sender to become less responsive to persistent congestion in the network, which can result in aggressive sending into a congested network. Too small a value can result in a sender declaring persistent congestion unnecessarily, resulting in reduced throughput for the sender.

The RECOMMENDED value for `kPersistentCongestionThreshold` is 3, which results in behavior that is approximately equivalent to a TCP sender declaring an RTO after two TLPs.

This design does not use consecutive PTO events to establish persistent congestion, since application patterns impact PTO expirations. For example, a sender that sends small amounts of data

with silence periods between them restarts the PTO timer every time it sends, potentially preventing the PTO timer from expiring for a long period of time, even when no acknowledgments are being received. The use of a duration enables a sender to establish persistent congestion without depending on PTO expiration.

#### 7.6.2. Establishing Persistent Congestion

A sender establishes persistent congestion after the receipt of an acknowledgment if two packets that are ack-eliciting are declared lost, and:

- \* across all packet number spaces, none of the packets sent between the send times of these two packets are acknowledged;
- \* the duration between the send times of these two packets exceeds the persistent congestion duration (Section 7.6.1); and
- \* a prior RTT sample existed when these two packets were sent.

These two packets MUST be ack-eliciting, since a receiver is required to acknowledge only ack-eliciting packets within its maximum ack delay; see Section 13.2 of [QUIC-TRANSPORT].

The persistent congestion period SHOULD NOT start until there is at least one RTT sample. Before the first RTT sample, a sender arms its PTO timer based on the initial RTT (Section 6.2.2), which could be substantially larger than the actual RTT. Requiring a prior RTT sample prevents a sender from establishing persistent congestion with potentially too few probes.

Since network congestion is not affected by packet number spaces, persistent congestion SHOULD consider packets sent across packet number spaces. A sender that does not have state for all packet number spaces or an implementation that cannot compare send times across packet number spaces MAY use state for just the packet number space that was acknowledged. This might result in erroneously declaring persistent congestion, but it will not lead to a failure to detect persistent congestion.

When persistent congestion is declared, the sender's congestion window MUST be reduced to the minimum congestion window (`kMinimumWindow`), similar to a TCP sender's response on an RTO ([RFC5681]).

## 7.6.3. Example

The following example illustrates how a sender might establish persistent congestion. Assume:

$\text{smoothed\_rtt} + \max(4 * \text{rttvar}, \text{kGranularity}) + \text{max\_ack\_delay} = 2$   
 $\text{kPersistentCongestionThreshold} = 3$

Consider the following sequence of events:

Time	Action
t=0	Send packet #1 (app data)
t=1	Send packet #2 (app data)
t=1.2	Recv acknowledgment of #1
t=2	Send packet #3 (app data)
t=3	Send packet #4 (app data)
t=4	Send packet #5 (app data)
t=5	Send packet #6 (app data)
t=6	Send packet #7 (app data)
t=8	Send packet #8 (PTO 1)
t=12	Send packet #9 (PTO 2)
t=12.2	Recv acknowledgment of #9

Table 1

Packets 2 through 8 are declared lost when the acknowledgment for packet 9 is received at  $t = 12.2$ .

The congestion period is calculated as the time between the oldest and newest lost packets:  $8 - 1 = 7$ . The persistent congestion duration is:  $2 * 3 = 6$ . Because the threshold was reached and because none of the packets between the oldest and the newest lost packets were acknowledged, the network is considered to have experienced persistent congestion.

While this example shows PTO expiration, they are not required for persistent congestion to be established.

### 7.7. Pacing

A sender SHOULD pace sending of all in-flight packets based on input from the congestion controller.

Sending multiple packets into the network without any delay between them creates a packet burst that might cause short-term congestion and losses. Senders MUST either use pacing or limit such bursts. Senders SHOULD limit bursts to the initial congestion window; see Section 7.2. A sender with knowledge that the network path to the receiver can absorb larger bursts MAY use a higher limit.

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.

Endpoints can implement pacing as they choose. A perfectly paced sender spreads packets exactly evenly over time. For a window-based congestion controller, such as the one in this document, that rate can be computed by averaging the congestion window over the round-trip time. Expressed as a rate in units of bytes per time, where `congestion_window` is in bytes:

$$\text{rate} = N * \text{congestion\_window} / \text{smoothed\_rtt}$$

Or, expressed as an inter-packet interval in units of time:

$$\text{interval} = ( \text{smoothed\_rtt} * \text{packet\_size} / \text{congestion\_window} ) / N$$

Using a value for "N" that is small, but at least 1 (for example, 1.25) ensures that variations in round-trip time do not result in under-utilization of the congestion window.

Practical considerations, such as packetization, scheduling delays, and computational efficiency, can cause a sender to deviate from this rate over time periods that are much shorter than a round-trip time.

One possible implementation strategy for pacing uses a leaky bucket algorithm, where the capacity of the "bucket" is limited to the maximum burst size and the rate the "bucket" fills is determined by the above function.

#### 7.8. Under-utilizing the Congestion Window

When bytes in flight is smaller than the congestion window and sending is not pacing limited, the congestion window is under-utilized. When this occurs, the congestion window SHOULD NOT be increased in either slow start or congestion avoidance. This can happen due to insufficient application data or flow control limits.

A sender that paces packets (see Section 7.7) 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.

A sender MAY implement alternative mechanisms to update its congestion window after periods of under-utilization, such as those proposed for TCP in [RFC7661].

### 8. Security Considerations

#### 8.1. Loss and Congestion Signals

Loss detection and congestion control fundamentally involve consumption of signals, such as delay, loss, and ECN markings, from unauthenticated entities. An attacker can cause endpoints to reduce their sending rate by manipulating these signals; by dropping packets, by altering path delay strategically, or by changing ECN codepoints.

#### 8.2. Traffic Analysis

Packets that carry only ACK frames can be heuristically identified by observing packet size. Acknowledgment patterns may expose information about link characteristics or application behavior. To reduce leaked information, endpoints can bundle acknowledgments with other frames, or they can use PADDING frames at a potential cost to performance.

#### 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 can detect suppression of reports by marking occasional packets that it sends with an ECN-CE marking. If a packet sent with an ECN-CE marking is not reported as having been CE marked when the packet is acknowledged, then the sender can disable ECN for that path by not setting ECT codepoints in subsequent packets sent on that path [RFC3168].

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. Congestion controllers respond to reports of ECN-CE by reducing their rate, but the response may vary. Markings can be treated as equivalent to loss ([RFC3168]), but other responses can be specified, such as ([RFC8511]) or ([RFC8311]).

## 9. IANA Considerations

This document has no IANA actions.

## 10. References

### 10.1. Normative References

[QUIC-TLS] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-tls-34, 15 January 2021, <<https://tools.ietf.org/html/draft-ietf-quic-tls-34>>.

[QUIC-TRANSPORT] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-34, 15 January 2021, <<https://tools.ietf.org/html/draft-ietf-quic-transport-34>>.

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

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

- [RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <<https://www.rfc-editor.org/info/rfc8085>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

## 10.2. Informative References

- [FACK] Mathis, M. and J. Mahdavi, "Forward Acknowledgement: Refining TCP Congestion Control", ACM SIGCOMM , August 1996.
- [PRR] Mathis, M., Dukkupati, N., and Y. Cheng, "Proportional Rate Reduction for TCP", RFC 6937, DOI 10.17487/RFC6937, May 2013, <<https://www.rfc-editor.org/info/rfc6937>>.
- [RACK] Cheng, Y., Cardwell, N., Dukkupati, N., and P. Jha, "The RACK-TLP loss detection algorithm for TCP", Work in Progress, Internet-Draft, draft-ietf-tcpm-rack-15, 22 December 2020, <<http://www.ietf.org/internet-drafts/draft-ietf-tcpm-rack-15.txt>>.
- [RETRANSMISSION] Karn, P. and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols", ACM SIGCOMM CCR , January 1995.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, DOI 10.17487/RFC2018, October 1996, <<https://www.rfc-editor.org/info/rfc2018>>.
- [RFC3465] Allman, M., "TCP Congestion Control with Appropriate Byte Counting (ABC)", RFC 3465, DOI 10.17487/RFC3465, February 2003, <<https://www.rfc-editor.org/info/rfc3465>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/info/rfc5681>>.
- [RFC5682] Sarolahti, P., Kojo, M., Yamamoto, K., and M. Hata, "Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP", RFC 5682, DOI 10.17487/RFC5682, September 2009, <<https://www.rfc-editor.org/info/rfc5682>>.



- [RFC5827] Allman, M., Avrachenkov, K., Ayesta, U., Blanton, J., and P. Hurtig, "Early Retransmit for TCP and Stream Control Transmission Protocol (SCTP)", RFC 5827, DOI 10.17487/RFC5827, May 2010, <<https://www.rfc-editor.org/info/rfc5827>>.
- [RFC6297] Welzl, M. and D. Ros, "A Survey of Lower-than-Best-Effort Transport Protocols", RFC 6297, DOI 10.17487/RFC6297, June 2011, <<https://www.rfc-editor.org/info/rfc6297>>.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <<https://www.rfc-editor.org/info/rfc6298>>.
- [RFC6582] Henderson, T., Floyd, S., Gurtov, A., and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 6582, DOI 10.17487/RFC6582, April 2012, <<https://www.rfc-editor.org/info/rfc6582>>.
- [RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, DOI 10.17487/RFC6675, August 2012, <<https://www.rfc-editor.org/info/rfc6675>>.
- [RFC6928] Chu, J., Dukkkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <<https://www.rfc-editor.org/info/rfc6928>>.
- [RFC7661] Fairhurst, G., Sathiaselalan, A., and R. Secchi, "Updating TCP to Support Rate-Limited Traffic", RFC 7661, DOI 10.17487/RFC7661, October 2015, <<https://www.rfc-editor.org/info/rfc7661>>.
- [RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <<https://www.rfc-editor.org/info/rfc8311>>.
- [RFC8312] Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", RFC 8312, DOI 10.17487/RFC8312, February 2018, <<https://www.rfc-editor.org/info/rfc8312>>.

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

## Appendix A. Loss Recovery Pseudocode

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

The pseudocode segments in this section are licensed as Code Components; see the copyright notice.

### 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, the endpoint can still maintain state for it for an amount of time to allow for packet reordering; see Section 13.3 of [QUIC-TRANSPORT]. This enables a sender to detect 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 acknowledgment will be received, though the peer could delay sending the ACK frame containing it by up to the `max_ack_delay`.

**in\_flight:** A boolean that indicates whether the packet counts towards bytes in flight.

**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.

**kPacketThreshold:** Maximum reordering in packets before packet threshold loss detection considers a packet lost. The value recommended in Section 6.1.1 is 3.

**kTimeThreshold:** Maximum reordering in time before time threshold loss detection considers a packet lost. Specified as an RTT multiplier. The value recommended in Section 6.1.2 is 9/8.

**kGranularity:** Timer granularity. This is a system-dependent value, and Section 6.1.2 recommends a value of 1ms.

**kInitialRtt:** The RTT used before an RTT sample is taken. The value recommended in Section 6.2.2 is 333ms.

**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.

**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 Section 5.3.

**rttvar:** The RTT variation, computed as described in Section 5.3.

**min\_rtt:** The minimum RTT seen over a period of time, ignoring acknowledgment delay, as described in Section 5.2.

**first\_rtt\_sample:** The time that the first RTT sample was obtained.

**max\_ack\_delay:** The maximum amount of time by which the receiver

intends to delay acknowledgments for packets in the Application Data packet number space, as defined by the eponymous transport parameter (Section 18.2 of [QUIC-TRANSPORT]). Note that the actual `ack_delay` in a received ACK frame may be larger due to late timers, reordering, or loss.

`loss_detection_timer`: Multi-modal timer used for loss detection.

`pto_count`: The number of times a PTO has been sent without receiving an ack.

`time_of_last_ack_eliciting_packet[kPacketNumberSpace]`: The time the most recent ack-eliciting packet was sent.

`largest_acked_packet[kPacketNumberSpace]`: The largest packet number acknowledged in the packet number space so far.

`loss_time[kPacketNumberSpace]`: The time at which the next packet in that packet number space can 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.

#### A.4. Initialization

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

```
loss_detection_timer.reset()
pto_count = 0
latest_rtt = 0
smoothed_rtt = kInitialRtt
rttvar = kInitialRtt / 2
min_rtt = 0
first_rtt_sample = 0
for pn_space in [ Initial, Handshake, ApplicationData ]:
    largest_acked_packet[pn_space] = infinite
    time_of_last_ack_eliciting_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, 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
    sent_packets[pn_space][packet_number].sent_bytes = sent_bytes
    if (in_flight):
        if (ack_eliciting):
            time_of_last_ack_eliciting_packet[pn_space] = now()
        OnPacketSentCC(sent_bytes)
        SetLossDetectionTimer()
```

#### A.6. On Receiving a Datagram

When a server is blocked by anti-amplification limits, receiving a datagram unblocks it, even if none of the packets in the datagram are successfully processed. In such a case, the PTO timer will need to be re-armed.

Pseudocode for OnDatagramReceived follows:

```
OnDatagramReceived(datagram):
    // If this datagram unblocks the server, arm the
    // PTO timer to avoid deadlock.
    if (server was at anti-amplification limit):
        SetLossDetectionTimer()
```

#### A.7. On Receiving an Acknowledgment

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

Pseudocode for OnAckReceived and UpdateRtt follow:

```
IncludesAckEliciting(packets):
    for packet in packets:
        if (packet.ack_eliciting):
            return true
    return false

OnAckReceived(ack, pn_space):
    if (largest_acked_packet[pn_space] == infinite):
        largest_acked_packet[pn_space] = ack.largest_acked
```

```
else:
    largest_acked_packet[pn_space] =
        max(largest_acked_packet[pn_space], ack.largest_acked)

// DetectAndRemoveAkedPackets finds packets that are newly
// acknowledged and removes them from sent_packets.
newly_acked_packets =
    DetectAndRemoveAkedPackets(ack, pn_space)
// Nothing to do if there are no newly acked packets.
if (newly_acked_packets.empty()):
    return

// Update the RTT if the largest acknowledged is newly acked
// and at least one ack-eliciting was newly acked.
if (newly_acked_packets.largest().packet_number ==
    ack.largest_acked &&
    IncludesAckEliciting(newly_acked_packets)):
    latest_rtt =
        now() - newly_acked_packets.largest().time_sent
    UpdateRtt(ack.ack_delay)

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

lost_packets = DetectAndRemoveLostPackets(pn_space)
if (!lost_packets.empty()):
    OnPacketsLost(lost_packets)
OnPacketsAked(newly_acked_packets)

// Reset pto_count unless the client is unsure if
// the server has validated the client's address.
if (PeerCompletedAddressValidation()):
    pto_count = 0
SetLossDetectionTimer()

UpdateRtt(ack_delay):
    if (first_rtt_sample == 0):
        min_rtt = latest_rtt
        smoothed_rtt = latest_rtt
        rttvar = latest_rtt / 2
        first_rtt_sample = now()
        return

// min_rtt ignores acknowledgment delay.
min_rtt = min(min_rtt, latest_rtt)
// Limit ack_delay by max_ack_delay after handshake
```

```

// confirmation.
if (handshake confirmed):
    ack_delay = min(ack_delay, max_ack_delay)

// Adjust for acknowledgment 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.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 fire immediately.

Pseudocode for `SetLossDetectionTimer` follows (where the " $^$ " operator represents exponentiation):

```

GetLossTimeAndSpace():
    time = loss_time[Initial]
    space = Initial
    for pn_space in [ Handshake, ApplicationData ]:
        if (time == 0 || loss_time[pn_space] < time):
            time = loss_time[pn_space];
            space = pn_space
    return time, space

GetPtoTimeAndSpace():
    duration = (smoothed_rtt + max(4 * rttvar, kGranularity))
               * (2 ^ pto_count)
    // Arm PTO from now when there are no inflight packets.
    if (no in-flight packets):
        assert(!PeerCompletedAddressValidation())
        if (has handshake keys):
            return (now() + duration), Handshake
        else:
            return (now() + duration), Initial
    pto_timeout = infinite
    pto_space = Initial

```

```
for space in [ Initial, Handshake, ApplicationData ]:
    if (no in-flight packets in space):
        continue;
    if (space == ApplicationData):
        // Skip Application Data until handshake confirmed.
        if (handshake is not confirmed):
            return pto_timeout, pto_space
        // Include max_ack_delay and backoff for Application Data.
        duration += max_ack_delay * (2 ^ pto_count)

    t = time_of_last_ack_eliciting_packet[space] + duration
    if (t < pto_timeout):
        pto_timeout = t
        pto_space = space
    return pto_timeout, pto_space

PeerCompletedAddressValidation():
    // Assume clients validate the server's address implicitly.
    if (endpoint is server):
        return true
    // Servers complete address validation when a
    // protected packet is received.
    return has received Handshake ACK ||
        handshake confirmed

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

    if (server is at anti-amplification limit):
        // The server's timer is not set if nothing can be sent.
        loss_detection_timer.cancel()
        return

    if (no ack-eliciting packets in flight &&
        PeerCompletedAddressValidation()):
        // There is nothing to detect lost, so no timer is set.
        // However, the client needs to arm the timer if the
        // server might be blocked by the anti-amplification limit.
        loss_detection_timer.cancel()
        return

    timeout, _ = GetPtoTimeAndSpace()
    loss_detection_timer.update(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:

```
OnLossDetectionTimeout():
    earliest_loss_time, pn_space = GetLossTimeAndSpace()
    if (earliest_loss_time != 0):
        // Time threshold loss Detection
        lost_packets = DetectAndRemoveLostPackets(pn_space)
        assert(!lost_packets.empty())
        OnPacketsLost(lost_packets)
        SetLossDetectionTimer()
        return

    if (bytes_in_flight > 0):
        // PTO. Send new data if available, else retransmit old data.
        // If neither is available, send a single PING frame.
        _, pn_space = GetPtoTimeAndSpace()
        SendOneOrTwoAckElicitingPackets(pn_space)
    else:
        assert(!PeerCompletedAddressValidation())
        // 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):
            SendOneAckElicitingHandshakePacket()
        else:
            SendOneAckElicitingPaddedInitialPacket()

    pto_count++
    SetLossDetectionTimer()
```

## A.10. Detecting Lost Packets

DetectAndRemoveLostPackets is called every time an ACK is received or the time threshold loss detection timer expires. This function operates on the sent\_packets for that packet number space and returns a list of packets newly detected as lost.

Pseudocode for DetectAndRemoveLostPackets follows:

```

DetectAndRemoveLostPackets(pn_space):
    assert(largest_acked_packet[pn_space] != infinite)
    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

    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.
        // Note: The use of kPacketThreshold here assumes that there
        // were no sender-induced gaps in the packet number space.
        if (unacked.time_sent <= lost_send_time ||
            largest_acked_packet[pn_space] >=
                unacked.packet_number + kPacketThreshold):
            sent_packets[pn_space].remove(unacked.packet_number)
            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)

    return lost_packets

```

#### A.11. Upon Dropping Initial or Handshake Keys

When Initial or Handshake keys are discarded, packets from the space are discarded and loss detection state is updated.

Pseudocode for OnPacketNumberSpaceDiscarded follows:

```

OnPacketNumberSpaceDiscarded(pn_space):
    assert(pn_space != ApplicationData)
    RemoveFromBytesInFlight(sent_packets[pn_space])
    sent_packets[pn_space].clear()
    // Reset the loss detection and PTO timer
    time_of_last_ack_eliciting_packet[pn_space] = 0
    loss_time[pn_space] = 0
    pto_count = 0
    SetLossDetectionTimer()

```

## Appendix B. Congestion Control Pseudocode

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

The pseudocode segments in this section are licensed as Code Components; see the copyright notice.

### B.1. Constants of interest

Constants used in congestion control are based on a combination of RFCs, papers, and common practice.

`kInitialWindow`: Default limit on the initial bytes in flight as described in Section 7.2.

`kMinimumWindow`: Minimum congestion window in bytes as described in Section 7.2.

`kLossReductionFactor`: Scaling factor applied to reduce the congestion window when a new loss event is detected. Section 7 recommends a value is 0.5.

`kPersistentCongestionThreshold`: Period of time for persistent congestion to be established, specified as a PTO multiplier. Section 7.6 recommends a value of 3.

### B.2. Variables of interest

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

`max_datagram_size`: The sender's current maximum payload size. Does not include UDP or IP overhead. The max datagram size is used for congestion window computations. An endpoint sets the value of this variable based on its Path Maximum Transmission Unit (PMTU; see Section 14.2 of [QUIC-TRANSPORT]), with a minimum value of 1200 bytes.

`ecn_ce_counters[kPacketNumberSpace]`: The highest value reported for the ECN-CE counter in the packet number space by the peer in an ACK frame. This value 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 acknowledged 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 allowed to be in flight.

`congestion_recovery_start_time`: The time the current recovery period started due to the detection of loss or ECN. 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.

The congestion control pseudocode also accesses some of the variables from the loss recovery pseudocode.

### 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
for pn_space in [ Initial, Handshake, ApplicationData ]:
    ecn_ce_counters[pn_space] = 0
```

### B.4. On Packet Sent

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

```
OnPacketSentCC(sent_bytes):
    bytes_in_flight += sent_bytes
```

### B.5. On Packet Acknowledgment

Invoked from loss detection's `OnAckReceived` and is supplied with the newly `acked_packets` from `sent_packets`.

In congestion avoidance, implementers that use an integer representation for `congestion_window` should be careful with division, and can use the alternative approach suggested in Section 2.1 of [RFC3465].

```
InCongestionRecovery(sent_time):
    return sent_time <= congestion_recovery_start_time

OnPacketsAacked(acked_packets):
    for acked_packet in acked_packets:
        OnPacketAacked(acked_packet)

OnPacketAacked(acked_packet):
    if (!acked_packet.in_flight):
        return;
    // Remove from bytes_in_flight.
    bytes_in_flight -= acked_packet.sent_bytes
    // Do not increase congestion_window if application
    // limited or flow control limited.
    if (IsAppOrFlowControlLimited())
        return
    // Do not increase congestion window in recovery period.
    if (InCongestionRecovery(acked_packet.time_sent)):
        return
    if (congestion_window < ssthresh):
        // Slow start.
        congestion_window += acked_packet.sent_bytes
    else:
        // Congestion avoidance.
        congestion_window +=
            max_datagram_size * acked_packet.sent_bytes
        / congestion_window
```

#### B.6. On New Congestion Event

Invoked from ProcessECN and OnPacketsLost when a new congestion event is detected. If not already in recovery, this starts a recovery period and reduces the slow start threshold and congestion window immediately.

```
OnCongestionEvent(sent_time):
    // No reaction if already in a recovery period.
    if (InCongestionRecovery(sent_time)):
        return

    // Enter recovery period.
    congestion_recovery_start_time = now()
    ssthresh = congestion_window * kLossReductionFactor
    congestion_window = max(ssthresh, kMinimumWindow)
    // A packet can be sent to speed up loss recovery.
    MaybeSendOnePacket()
```

### B.7. Process ECN Information

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

```
ProcessECN(ack, pn_space):
    // If the ECN-CE counter reported by the peer has increased,
    // this could be a new congestion event.
    if (ack.ce_counter > ecn_ce_counters[pn_space]):
        ecn_ce_counters[pn_space] = ack.ce_counter
        sent_time = sent_packets[ack.largest_acked].time_sent
        OnCongestionEvent(sent_time)
```

### B.8. On Packets Lost

Invoked when DetectAndRemoveLostPackets deems packets lost.

```
OnPacketsLost(lost_packets):
    sent_time_of_last_loss = 0
    // Remove lost packets from bytes_in_flight.
    for lost_packet in lost_packets:
        if lost_packet.in_flight:
            bytes_in_flight -= lost_packet.sent_bytes
            sent_time_of_last_loss =
                max(sent_time_of_last_loss, lost_packet.time_sent)
    // Congestion event if in-flight packets were lost
    if (sent_time_of_last_loss != 0):
        OnCongestionEvent(sent_time_of_last_loss)

    // Reset the congestion window if the loss of these
    // packets indicates persistent congestion.
    // Only consider packets sent after getting an RTT sample.
    if (first_rtt_sample == 0):
        return
    pc_lost = []
    for lost in lost_packets:
        if lost.time_sent > first_rtt_sample:
            pc_lost.insert(lost)
    if (InPersistentCongestion(pc_lost)):
        congestion_window = kMinimumWindow
        congestion_recovery_start_time = 0
```

### B.9. Removing Discarded Packets From Bytes In Flight

When Initial or Handshake keys are discarded, packets sent in that space no longer count toward bytes in flight.

Pseudocode for RemoveFromBytesInFlight follows:

```
RemoveFromBytesInFlight(discarded_packets):  
    // Remove any unacknowledged packets from flight.  
    foreach packet in discarded_packets:  
        if packet.in_flight  
            bytes_in_flight -= size
```

#### 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-32

- \* Clarifications to definition of persistent congestion (#4413, #4414, #4421, #4429, #4437)

##### C.2. Since draft-ietf-quic-recovery-31

- \* Limit the number of Initial packets sent in response to unauthenticated packets (#4183, #4188)

##### C.3. Since draft-ietf-quic-recovery-30

Editorial changes only.

##### C.4. Since draft-ietf-quic-recovery-29

- \* Allow caching of packets that can't be decrypted, by allowing the reported acknowledgment delay to exceed max\_ack\_delay prior to confirming the handshake (#3821, #3980, #4035, #3874)
- \* Persistent congestion cannot include packets sent before the first RTT sample for the path (#3875, #3889)
- \* Recommend reset of min\_rtt in persistent congestion (#3927, #3975)
- \* Persistent congestion is independent of packet number space (#3939, #3961)
- \* Only limit bursts to the initial window without information about the path (#3892, #3936)
- \* Add normative requirements for increasing and reducing the congestion window (#3944, #3978, #3997, #3998)

## C.5. Since draft-ietf-quic-recovery-28

- \* Refactored pseudocode to correct PTO calculation (#3564, #3674, #3681)

## C.6. Since draft-ietf-quic-recovery-27

- \* Added recommendations for speeding up handshake under some loss conditions (#3078, #3080)
- \* PTO count is reset when handshake progress is made (#3272, #3415)
- \* PTO count is not reset by a client when the server might be awaiting address validation (#3546, #3551)
- \* Recommend repairing losses immediately after entering the recovery period (#3335, #3443)
- \* Clarified what loss conditions can be ignored during the handshake (#3456, #3450)
- \* Allow, but don't recommend, using RTT from previous connection to seed RTT (#3464, #3496)
- \* Recommend use of adaptive loss detection thresholds (#3571, #3572)

## C.7. Since draft-ietf-quic-recovery-26

No changes.

## C.8. Since draft-ietf-quic-recovery-25

No significant changes.

## C.9. Since draft-ietf-quic-recovery-24

- \* Require congestion control of some sort (#3247, #3244, #3248)
- \* Set a minimum reordering threshold (#3256, #3240)
- \* PTO is specific to a packet number space (#3067, #3074, #3066)

## C.10. Since draft-ietf-quic-recovery-23

- \* Define under-utilizing the congestion window (#2630, #2686, #2675)
- \* PTO MUST send data if possible (#3056, #3057)



- \* Connection Close is not ack-eliciting (#3097, #3098)
- \* MUST limit bursts to the initial congestion window (#3160)
- \* Define the current max\_datagram\_size for congestion control (#3041, #3167)

C.11. Since draft-ietf-quic-recovery-22

- \* PTO should always send an ack-eliciting packet (#2895)
- \* Unify the Handshake Timer with the PTO timer (#2648, #2658, #2886)
- \* Move ACK generation text to transport draft (#1860, #2916)

C.12. Since draft-ietf-quic-recovery-21

- \* No changes

C.13. Since draft-ietf-quic-recovery-20

- \* Path validation can be used as initial RTT value (#2644, #2687)
- \* max\_ack\_delay transport parameter defaults to 0 (#2638, #2646)
- \* ACK delay only measures intentional delays induced by the implementation (#2596, #2786)

C.14. Since draft-ietf-quic-recovery-19

- \* Change kPersistentThreshold from an exponent to a multiplier (#2557)
- \* 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)

- \* Clarify that the time threshold loss alarm takes precedence over the crypto handshake timer (#2590, #2620)
- \* Change initial RTT to 500ms to align with RFC6298 (#2184)

C.15. 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.16. 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.17. 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 keys 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.18. Since draft-ietf-quic-recovery-14

- \* Used max\_ack\_delay from transport params (#1796, #1782)
- \* Merge ACK and ACK\_ECN (#1783)

## C.19. Since draft-ietf-quic-recovery-13

- \* 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.20. 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.21. Since draft-ietf-quic-recovery-11

No significant changes.

C.22. 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.23. Since draft-ietf-quic-recovery-09

No significant changes.

C.24. Since draft-ietf-quic-recovery-08

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

C.25. 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.26. Since draft-ietf-quic-recovery-06

No significant changes.

C.27. Since draft-ietf-quic-recovery-05

- \* Add more congestion control text (#776)

C.28. Since draft-ietf-quic-recovery-04

No significant changes.

C.29. Since draft-ietf-quic-recovery-03

No significant changes.

C.30. Since draft-ietf-quic-recovery-02

- \* Integrate F-RTO (#544, #409)

- \* Add congestion control (#545, #395)

- \* Require connection abort if a skipped packet was acknowledged (#415)

- \* Simplify RTO calculations (#142, #417)

C.31. Since draft-ietf-quic-recovery-01

- \* Overview added to loss detection

- \* Changes initial default RTT to 100ms

- \* Added time-based loss detection and fixes early retransmit

- \* Clarified loss recovery for handshake packets

- \* Fixed references and made TCP references informative

C.32. Since draft-ietf-quic-recovery-00

- \* Improved description of constants and ACK behavior

C.33. Since draft-iyengar-quic-loss-recovery-01

- \* Adopted as base for draft-ietf-quic-recovery

- \* Updated authors/editors list

- \* Added table of contents

#### Appendix D. Contributors

The IETF QUIC Working Group received an enormous amount of support from many people. The following people provided substantive contributions to this document:

- \* Alessandro Ghedini
- \* Benjamin Saunders
- \* Gorrry Fairhurst
- \* (Kazu Yamamoto)
- \* (Kazuho Oku)
- \* Lars Eggert
- \* Magnus Westerlund
- \* Marten Seemann
- \* Martin Duke
- \* Martin Thomson
- \* Mirja Kühlewind
- \* Nick Banks
- \* Praveen Balasubramanian

#### Acknowledgments

#### Authors' Addresses

Jana Iyengar (editor)  
Fastly

Email: [jri.ietf@gmail.com](mailto:jri.ietf@gmail.com)

Ian Swett (editor)  
Google

Email: [ianswett@google.com](mailto:ianswett@google.com)

TCP Maintenance & Minor Extensions (tcpm)  
Internet-Draft  
Updates: 3168, 3449 (if approved)  
Intended status: Standards Track  
Expires: September 23, 2022

B. Briscoe  
Independent  
M. Kuehlewind  
Ericsson  
R. Scheffenegger  
NetApp  
March 22, 2022

More Accurate ECN Feedback in TCP  
draft-ietf-tcpm-accurate-ecn-18

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 was originally 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 updates the original ECN specification to specify a 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 previously assigned to the ECN-Nonce. It also overloads the two existing ECN flags in the TCP header. The resulting extra space is exploited to feed back the IP-ECN field received during the 3-way handshake as well. Supplementary feedback information can optionally be provided in a new TCP option, which is never used on the TCP SYN. The document also specifies the treatment of this updated TCP wire protocol by middleboxes, updating BCP 69 with respect to ACK filtering.

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 23, 2022.

#### Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include 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. Document Roadmap . . . . .	5
1.2. Goals . . . . .	5
1.3. Terminology . . . . .	6
1.4. Recap of Existing ECN feedback in IP/TCP . . . . .	6
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 . . . . .	9
2.4. Feedback Metrics . . . . .	10
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. Backward Compatibility . . . . .	13
3.1.3. Forward Compatibility . . . . .	15
3.1.4. Retransmission of the SYN . . . . .	15
3.1.5. Implications of AccECN Mode . . . . .	16
3.2. AccECN Feedback . . . . .	18
3.2.1. Initialization of Feedback Counters . . . . .	19
3.2.2. The ACE Field . . . . .	19
3.2.2.1. ACE Field on the ACK of the SYN/ACK . . . . .	20
3.2.2.2. Encoding and Decoding Feedback in the ACE Field . . . . .	21
3.2.2.3. Testing for Mangling of the IP/ECN Field . . . . .	23
3.2.2.4. Testing for Zeroing of the ACE Field . . . . .	25
3.2.2.5. Safety against Ambiguity of the ACE Field . . . . .	26



3.2.3.	The AccECN Option . . . . .	28
3.2.3.1.	Encoding and Decoding Feedback in the AccECN Option Fields . . . . .	30
3.2.3.2.	Path Traversal of the AccECN Option . . . . .	31
3.2.3.3.	Usage of the AccECN TCP Option . . . . .	35
3.3.	AccECN Compliance Requirements for TCP Proxies, Offload Engines and other Middleboxes . . . . .	37
3.3.1.	Requirements for TCP Proxies . . . . .	37
3.3.2.	Requirements for Transparent Middleboxes and TCP Normalizers . . . . .	37
3.3.3.	Requirements for TCP ACK Filtering . . . . .	38
3.3.4.	Requirements for TCP Segmentation Offload . . . . .	39
4.	Updates to RFC 3168 . . . . .	40
5.	Interaction with TCP Variants . . . . .	41
5.1.	Compatibility with SYN Cookies . . . . .	41
5.2.	Compatibility with TCP Experiments and Common TCP Options . . . . .	42
5.3.	Compatibility with Feedback Integrity Mechanisms . . . . .	42
6.	Protocol Properties . . . . .	43
7.	IANA Considerations . . . . .	45
8.	Security Considerations . . . . .	47
9.	Acknowledgements . . . . .	48
10.	Comments Solicited . . . . .	48
11.	References . . . . .	48
11.1.	Normative References . . . . .	48
11.2.	Informative References . . . . .	49
Appendix A.	Example Algorithms . . . . .	52
A.1.	Example Algorithm to Encode/Decode the AccECN Option . . . . .	52
A.2.	Example Algorithm for Safety Against Long Sequences of ACK Loss . . . . .	53
A.2.1.	Safety Algorithm without the AccECN Option . . . . .	53
A.2.2.	Safety Algorithm with the AccECN Option . . . . .	55
A.3.	Example Algorithm to Estimate Marked Bytes from Marked Packets . . . . .	57
A.4.	Example Algorithm to Count Not-ECT Bytes . . . . .	58
Appendix B.	Rationale for Usage of TCP Header Flags . . . . .	58
B.1.	Three TCP Header Flags in the SYN-SYN/ACK Handshake . . . . .	58
B.2.	Four Codepoints in the SYN/ACK . . . . .	59
B.3.	Space for Future Evolution . . . . .	60
Authors' Addresses	. . . . .	61

## 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 feed back this information to the sender. In RFC 3168, ECN was specified for TCP in such a way that only one feedback signal could 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 update to the ECN feedback scheme of RFC 3168 that provides more accurate information and could be used by these and potentially other future TCP extensions. A fuller treatment of the motivation for this specification is given in the associated requirements document [RFC7560].

This document specifies a standards track 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. This document updates RFC 3168 with respect to negotiation and use of the feedback scheme for TCP. All aspects of RFC 3168 other than the TCP feedback scheme, in particular the definition of ECN at the IP layer, remain unchanged by this specification. Section 4 gives a more detailed specification of exactly which aspects of RFC 3168 this document updates.

AccECN 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 can coexist alongside 'classic' [RFC3168] 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. This document uses the term Classic ECN when it needs to distinguish the RFC 3168 ECN TCP feedback scheme from the AccECN TCP feedback scheme.

AccECN feedback overloads the two existing ECN flags in the TCP header and allocates the currently reserved flag (previously called NS) in the TCP header, to be used as one three-bit counter 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 a change to the TCP wire protocol; it covers 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, but ultimately the motivation for accurate ECN feedback. Like Classic ECN feedback, AccECN can be used by standard Reno congestion control [RFC5681] to respond to the existence of at least one congestion

notification within a round trip. Or, unlike Reno, AccECN can be used to respond to the extent of congestion notification over a round trip, as for example DCTCP does in controlled environments [RFC8257]. For congestion response, this specification refers to RFC 3168, or ECN experiments such as those referred to in [RFC8311], namely: a TCP-based Low Latency Low Loss Scalable (L4S) congestion control [I-D.ietf-tsvwg-l4s-arch]; or Alternative Backoff with ECN (ABE) [RFC8511].

It is RECOMMENDED that the AccECN protocol is implemented alongside SACK [RFC2018] and the experimental ECN++ protocol [I-D.ietf-tcpm-generalized-ecn], which allows the ECN capability to be used on TCP control packets. 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 section outlines the goals of AccECN (Section 1.2). Then terminology is defined (Section 1.3) and a recap of existing prerequisite technology is given (Section 1.4).

Section 2 gives an informative overview of the AccECN protocol. Then Section 3 gives the normative protocol specification, and Section 4 clarifies which aspects of RFC 3168 are updated by this specification. Section 5 assesses the interaction of AccECN with commonly used variants of TCP, whether standardized or not. Section 6 summarizes the features and properties of AccECN.

Section 7 summarizes the protocol fields and numbers that IANA will need to assign and Section 8 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 and Appendix B explains why AccECN uses flags in the main TCP header and quantifies the space left for future use.

### 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 recognizes that a perfect scheme that fully satisfies all the requirements is unlikely and trade-offs between requirements are likely. Section 6 presents the properties of AccECN against these requirements and discusses the trade-offs made.

The requirements document recognizes 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. 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 (ACK=1).

**Pure ACK:** A TCP acknowledgement without a data payload.

**Acceptable packet / segment:** A packet or segment that passes the acceptability tests in [RFC0793] and [RFC5961].

**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.

### 1.4. 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.

IP-ECN codepoint	Codepoint name	Description
0b00	Not-ECT	Not ECN-Capable Transport
0b01	ECT(1)	ECN-Capable Transport (1)
0b10	ECT(0)	ECN-Capable Transport (0)
0b11	CE	Congestion Experienced

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
Header Length				Reserved			N S	C W R	E C E	U R G	A C K	P S H	R S T	S Y N	F I 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:

- o an essential part that re-uses ECN TCP header bits for the Data Receiver to feed back the number of packets arriving with CE in the IP-ECN field. This provides more accuracy than classic ECN feedback, but limited resilience against ACK loss;
- o 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 in the IP-ECN field (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:

- o 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. 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 initialized at the start of the half-connection. Three count the number of arriving payload bytes respectively marked CE, ECT(1) and ECT(0) in the IP-ECN field. 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 3 later). The 24 LSBs of each byte counter 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 ought to 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 not allowed 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 assuming the most likely wrap, but with a degree of conservatism.

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 options are reaching the sender and 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 L4S, 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 provided 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 feedback about CE markings in the IP-ECN field of both data and control packets. According to [RFC3168] the Data Sender is meant to set the IP-ECN field of 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 Section 3.2.2.3, Section 3.2.2.4 and Section 3.2.3.2 of the present document and para 2 of Section 20.2 of [RFC3168]).

The initial SYN is the most critical control packet, so AccECN provides feedback on its IP-ECN field. Although RFC 3168 prohibits an ECN-capable SYN, providing feedback of ECN marking on the SYN supports future scenarios in which SYNs might be ECN-enabled (without prejudging whether they ought to be). For instance, [RFC8311] updates this aspect of RFC 3168 to allow experimentation with ECN-capable TCP control packets.

Even if the TCP client (or server) has set the SYN (or SYN/ACK) to not-ECT in compliance with RFC 3168, feedback on the state of the IP-ECN field when it arrives at the receiver could still be useful, because middleboxes have been known to overwrite the IP-ECN field as if it is still part of the old Type of Service (ToS) field [Mandalaril8]. For example, if a TCP client has set the SYN to Not-ECT, but receives feedback that the IP-ECN field on the SYN arrived with a different codepoint, it can detect such middlebox interference. Previously, neither end knew what IP-ECN field the other had sent. So, if a TCP server received ECT or CE on a SYN, it could not know whether it was invalid (or valid) because only the TCP client knew whether it originally marked the SYN as Not-ECT (or ECT). Therefore, prior to AccECN, the server's only safe course of action in this example 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 7) as shown below.

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

Figure 2: The (post-AccECN) definition of the TCP header flags during the TCP handshake

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 AE, CWR and ECE TCP flags on the SYN/ACK to the combination in the top block of 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).

When the TCP server returns any of the 4 combinations in the top block of Table 2, it confirms 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.

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.

Once in AccECN mode, a TCP client or server has the rights and obligations to participate in the ECN protocol defined in Section 3.1.5.

The procedure for the client to follow if a SYN/ACK does not arrive before its retransmission timer expires is given in Section 3.1.4.

### 3.1.2. Backward Compatibility

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, as above. 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.

A	B	SYN A->B			SYN/ACK B->A			Feedback Mode
		AE	CWR	ECE	AE	CWR	ECE	
AccECN	AccECN	1	1	1	0	1	0	AccECN (no ECT on SYN)
AccECN	AccECN	1	1	1	0	1	1	AccECN (ECT1 on SYN)
AccECN	AccECN	1	1	1	1	0	0	AccECN (ECT0 on SYN)
AccECN	AccECN	1	1	1	1	1	0	AccECN (CE on SYN)
AccECN	Nonce	1	1	1	1	0	1	(Reserved)
AccECN	ECN	1	1	1	0	0	1	classic ECN
AccECN	No ECN	1	1	1	0	0	0	Not ECN
Nonce	AccECN	0	1	1	0	0	1	classic ECN
ECN	AccECN	0	1	1	0	0	1	classic ECN
No ECN	AccECN	0	0	0	0	0	0	Not ECN
AccECN	Broken	1	1	1	1	1	1	Not ECN

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 in Section 3.1 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. If it has set itself into classic ECN feedback mode it MUST then comply with [RFC3168].

The server response called 'Nonce' in the table is now historic. For an AccECN implementation, there is no need to recognize or support ECN Nonce feedback [RFC3540], which has been reclassified as historic [RFC8311]. AccECN is compatible with alternative ECN feedback integrity approaches (see Section 5.3).

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.

When an AccECN-enabled TCP server (B) receives a SYN with AE,CWR,ECE = 0,1,1 it MUST do one of the following:

- \* set both its half connections into the classic ECN feedback mode and return a SYN/ACK with AE, CWR, ECE = 0,0,1 as shown. Then it MUST comply with [RFC3168].
- \* set both its half-connections into No ECN mode and return a SYN/ACK with AE,CWR,ECE = 0,0,0, then continue with ECN disabled. This latter case is unlikely to be desirable, but it is allowed as a possibility, e.g. for minimal TCP implementations.

When an AccECN-enabled TCP server (B) receives a SYN with AE,CWR,ECE = 0,0,0 it MUST set both its half connections into the Not ECN feedback mode, return a SYN/ACK with AE,CWR,ECE = 0,0,0 as shown and continue with ECN disabled.

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 and continue with ECN disabled.

The following additional rules do not fit the structure of the table, but they complement it:

**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.

**In-window SYN during TIME-WAIT:** Many TCP implementations create a new TCP connection if they receive an in-window SYN packet during TIME-WAIT state. When a TCP host enters TIME-WAIT or CLOSED state, it ought to ignore any previous state about the negotiation of AccECN for that connection and renegotiate the feedback mode according to Table 2.

### 3.1.3. 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 behaviour described in the present specification applies whether or not the three remaining reserved TCP header flags are zero.

### 3.1.4. 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 with the three TCP-ECN flags cleared (AE=CWR=ECE=0). A retransmitted SYN MUST use the same ISN as the original SYN.

Retrying once before fall-back 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). However, other fall-back strategies will need to follow all the rules in Section 3.1.5, which concern behaviour when SYNs or SYN/ACKs negotiating different types of feedback have been sent within the same connection.

Further it might make sense to also remove any other new or 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.

Whichever fall-back strategy is used, 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 needs to 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 3.2.3.2.

### 3.1.5. Implications of AccECN Mode

Section 3.1.1 describes the only ways that a host can enter AccECN mode, whether as a client or as a server.

As a Data Sender, a host in AccECN mode has the rights and obligations concerning the use of ECN defined below, which build on those in [RFC3168] as updated by [RFC8311]:

- o Using ECT:

- \* It can set an ECT codepoint in the IP header of packets to indicate to the network that the transport is capable and willing to participate in ECN for this packet.
- \* It does not have to set ECT on any packet (for instance if it has reason to believe such a packet would be blocked).

- o Switching feedback negotiation (e.g. fall-back):

- \* It SHOULD NOT set ECT on any packet if it has received at least one valid SYN or Acceptable SYN/ACK with AE=CWR=ECE=0. A

"valid SYN" has the same port numbers and the same ISN as the SYN that caused the server to enter AccECN mode.

- \* It MUST NOT send an ECN-setup SYN [RFC3168] within the same connection as it has sent a SYN requesting AccECN feedback.
- \* It MUST NOT send an ECN-setup SYN/ACK [RFC3168] within the same connection as it has sent a SYN/ACK agreeing to use AccECN feedback.

The above rules are necessary because, if one peer were to negotiate the feedback mode in two different types of handshake, it would not be possible for the other peer to know for certain which handshake packet(s) the other end had eventually received or in which order it received them. So, in the absence of these rules, the two peers could end up using different feedback modes without knowing it.

o Congestion response:

- \* It is still obliged to respond appropriately to AccECN feedback that indicates there were ECN marks on packets it had previously sent, as defined in Section 6.1 of [RFC3168] and updated by Sections 2.1 and 4.1 of [RFC8311].

In general, it is obliged to respond to congestion feedback even when it is solely sending non-ECN-capable packets (for rationale, some examples and some exceptions see Section 3.2.2.3, Section 3.2.2.4).

- \* The commitment to respond appropriately to incoming indications of congestion remains even if it sends a SYN packet with AE=CWR=ECE=0, in a later transmission within the same TCP connection.
- \* Unlike an RFC 3168 data sender, it MUST NOT set CWR to indicate it has received and responded to indications of congestion (for the avoidance of doubt, this does not preclude it from setting the bits of the ACE counter field, which includes an overloaded use of the same bit).

As a Data Receiver:

- o a host in AccECN mode MUST feed back the information in the IP-ECN field of incoming packets using Accurate ECN feedback, as specified in Section 3.2 below.

- o if it receives an ECN-setup SYN or ECN-setup SYN/ACK [RFC3168] during the same connection as it receives a SYN requesting AccECN feedback or a SYN/ACK agreeing to use AccECN feedback, it MUST reset the connection with a RST packet.
- o If for any reason it is not willing to provide ECN feedback on a particular TCP connection, to indicate this unwillingness it SHOULD clear the AE, CWR and ECE flags in all SYN and/or SYN/ACK packets that it sends.
- o it MUST NOT use reception of packets with ECT set in the IP-ECN field as an implicit signal that the peer is ECN-capable. Reason: ECT at the IP layer does not explicitly confirm the peer has the correct ECN feedback logic, as the packets could have been mangled at the IP layer.

### 3.2. AccECN Feedback

Each Data Receiver of each half connection maintains four counters, r.cep, r.ceb, r.e0b and r.elb:

- o The Data Receiver MUST increment the CE packet counter (r.cep), for every Acceptable packet that it receives with the CE code point in the IP ECN field, including CE marked control packets but excluding CE on SYN packets (SYN=1; ACK=0).
- o A Data Receiver that supports sending of the AccECN TCP Option MUST increment the r.ceb, r.e0b or r.elb byte counters by the number of TCP payload octets in Acceptable packets marked respectively with the CE, ECT(0) and ECT(1) codepoint in their IP-ECN field, including any payload octets on control packets, but not including any payload octets on SYN packets (SYN=1; ACK=0).

Each Data Sender of each half connection maintains four counters, s.cep, s.ceb, s.e0b and s.elb intended to track the equivalent counters at the Data Receiver.

A Data Receiver feeds back the CE packet counter using the Accurate ECN (ACE) field, as explained in Section 3.2.2. And it optionally feeds back all the byte counters using the AccECN TCP Option, as specified in Section 3.2.3.

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

When a host first enters AccECN mode, in its role as a Data Receiver it initializes its counters to  $r.cep = 5$ ,  $r.e0b = r.elb = 1$  and  $r.ceb = 0$ ,

Non-zero initial values are used to support a stateless handshake (see Section 5.1) and to be distinct from cases where the fields are incorrectly zeroed (e.g. by middleboxes - see Section 3.2.3.2.4).

When a host enters AccECN mode, in its role as a Data Sender it initializes its counters to  $s.cep = 5$ ,  $s.e0b = s.elb = 1$  and  $s.ceb = 0$ .

### 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 3.

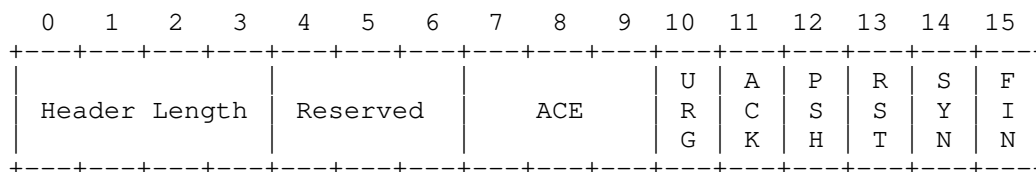


Figure 3: Definition of the ACE field within bytes 13 and 14 of the TCP Header (when AccECN has been negotiated and SYN=0).

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.

With one exception (Section 3.2.2.1), a host with both of its half-connections in AccECN mode MUST interpret the AE, CWR and ECE flags as the 3-bit ACE counter on a segment with the SYN flag cleared (SYN=0). On such a packet, a Data Receiver MUST encode the three least significant bits of its  $r.cep$  counter into the ACE field that it feeds back to the Data Sender. 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).

### 3.2.2.1. ACE Field on the ACK of the SYN/ACK

A TCP client (A) in AccECN mode MUST feed back which of the 4 possible values of the IP-ECN field was on the SYN/ACK by writing it into the ACE field of a pure ACK with no SACK blocks using the binary encoding in Table 3 (which is the same as that used on the SYN/ACK in Table 2). This shall be called the handshake encoding of the ACE field, and it is the only exception to the rule that the ACE field carries the 3 least significant bits of the r.cep counter on packets with SYN=0.

Normally, a TCP client acknowledges a SYN/ACK with an ACK that satisfies the above conditions anyway (SYN=0, no data, no SACK blocks). If an AccECN TCP client intends to acknowledge the SYN/ACK with a packet that does not satisfy these conditions (e.g. it has data to include on the ACK), it SHOULD first send a pure ACK that does satisfy these conditions (see Section 5.2), so that it can feed back which of the four values of the IP-ECN field arrived on the SYN/ACK. A valid exception to this "SHOULD" would be where the implementation will only be used in an environment where mangling of the ECN field is unlikely.

IP-ECN codepoint on SYN/ACK	ACE on pure ACK of SYN/ACK	r.cep of client in AccECN mode
Not-ECT	0b010	5
ECT(1)	0b011	5
ECT(0)	0b100	5
CE	0b110	6

Table 3: The encoding of the ACE field in the ACK of the SYN-ACK to reflect the SYN-ACK's IP-ECN field

When an AccECN server in SYN-RCVD state receives a pure ACK with SYN=0 and no SACK blocks, instead of treating the ACE field as a counter, it MUST infer the meaning of each possible value of the ACE field from Table 4, which also shows the value that an AccECN server MUST set s.cep to as a result.

Given this encoding of the ACE field on the ACK of a SYN/ACK is exceptional, an AccECN server using large receive offload (LRO) might prefer to disable LRO until such an ACK has transitioned it out of SYN-RCVD state.

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

Table 4: Meaning of the ACE field on the ACK of the SYN/ACK

{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 3.2.2.4).

{Note 2}: If the server is in AccECN mode, these values are Currently Unused but the AccECN server's behaviour is still defined for forward compatibility. Then the designer of a future protocol can know for certain what AccECN servers will do with these codepoints.

{Note 3}: 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. The values 0b000 or 0b001 will remind it that it did not enter AccECN mode, because AccECN does not use them (see Section 5.1 for details). If a stateless server that implements AccECN receives either of these two values in the ACK, its action is implementation-dependent and outside the scope of this spec, It will certainly not take the action in the third column because, after it receives either of these values, it is not in AccECN mode. I.e., it will not disable ECN (at least not just because ACE is 0b000) and it will not set s.cep.

#### 3.2.2.2. Encoding and Decoding Feedback in the ACE Field

Whenever the Data Receiver sends an ACK with SYN=0 (with or without data), unless the handshake encoding in Section 3.2.2.1 applies, the Data Receiver MUST encode the least significant 3 bits of its r.cep counter into the ACE field (see Appendix A.2).

Whenever the Data Sender receives an ACK with SYN=0 (with or without data), it first checks whether it has already been superseded by another ACK in which case it ignores the ECN feedback. If the ACK has not been superseded, and if the special handshake encoding in Section 3.2.2.1 does not apply, the Data Sender decodes the ACE field as follows (see Appendix A.2 for examples).

- o It takes the least significant 3 bits of its local s.cep counter and subtracts them from the incoming ACE counter to work out the minimum positive increment it could apply to s.cep (assuming the ACE field only wrapped at most once).
- o It then follows the safety procedures in Section 3.2.2.5.2 to calculate or estimate how many packets the ACK could have acknowledged under the prevailing conditions to determine whether the ACE field might have wrapped more than once.

The encode/decode procedures during the three-way handshake are exceptions to the general rules given so far, so they are spelled out step by step below for clarity:

- o If a TCP server in AccECN mode receives a CE mark in the IP-ECN field of a SYN (SYN=1, ACK=0), it MUST NOT increment r.cep (it remains at its initial value of 5).

Reason: It would be redundant for the server to include CE-marked SYNs in its r.cep counter, because it already reliably delivers feedback of any CE marking using the encoding in Table 2 in the SYN/ACK. This also ensures that, when the server starts using the ACE field, it has not unnecessarily consumed more than one initial value, given they can be used to negotiate variants of the AccECN protocol (see Appendix B.3).

- o If a TCP client in AccECN mode receives CE feedback in the TCP flags of a SYN/ACK, it MUST NOT increment s.cep (it remains at its initial value of 5), so that it stays in step with r.cep on the server. Nonetheless, the TCP client still triggers the congestion control actions necessary to respond to the CE feedback.
- o If a TCP client in AccECN mode receives a CE mark in the IP-ECN field of a SYN/ACK, it MUST increment r.cep, but no more than once no matter how many CE-marked SYN/ACKs it receives (i.e. incremented from 5 to 6, but no further).

Reason: Incrementing r.cep ensures the client will eventually deliver any CE marking to the server reliably when it starts using the ACE field. Even though the client also feeds back any CE marking on the ACK of the SYN/ACK using the encoding in Table 3,

this ACK is not delivered reliably, so it can be considered as a timely notification that is redundant but unreliable. The client does not increment `r.cep` more than once, because the server can only increment `s.cep` once (see next bullet). Also, this limits the unnecessarily consumed initial values of the ACE field to two.

- o If a TCP server in AccECN mode and in SYN-RCVD state receives CE feedback in the TCP flags of a pure ACK with no SACK blocks, it MUST increment `s.cep` (from 5 to 6). The TCP server then triggers the congestion control actions necessary to respond to the CE feedback.

Reasoning: The TCP server can only increment `s.cep` once, because the first ACK it receives will cause it to transition out of SYN-RCVD state. The server's congestion response would be no different even if it could receive feedback of more than one CE-marked SYN/ACK.

Once the TCP server transitions to ESTABLISHED state, it might later receive other pure ACK(s) with the handshake encoding in the ACE field. A server MAY implement a test for such a case, but it is not required. Therefore, once in the ESTABLISHED state, it will be sufficient for the server to consider the ACE field to be encoded as the normal ACE counter on all packets with SYN=0.

Reasoning: Such ACKs will be quite unusual, e.g. a SYN/ACK (or ACK of the SYN/ACK) that is delayed for longer than the server's retransmission timeout; or packet duplication by the network. And the impact of any error in the feedback on such ACKs will only be temporary.

### 3.2.2.3. 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 invalid transition (defined below) of the IP/ECN field, for the remainder of the half-connection the client is advised to send non-ECN-capable packets, but it still ought to respond to any feedback of CE markings (explained below). However, the client MUST remain in the AccECN feedback mode and it MUST continue to feed back any ECN markings on arriving packets (in its role as Data Receiver).

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 invalid

transition of the IP/ECN field, for the remainder of the half-connection the server is advised to send non-ECN-capable packets, but it still ought to respond to any feedback of CE markings (explained below). However, the server **MUST** remain in the AccECN feedback mode and it **MUST** continue to feed back any ECN markings on arriving packets (in its role as Data Receiver).

If a Data Sender in AccECN mode starts sending non-ECN-capable packets because it has detected mangling, it is still advised to respond to CE feedback. Reason: any CE-marking arriving at the Data Receiver could be due to something early in the path mangling the non-ECN-capable IP/ECN field into an ECN-capable codepoint and then, later in the path, a network bottleneck might be applying CE-markings to indicate genuine congestion. This argument applies whether the handshake packet originally sent by the client or server was non-ECN-capable or ECN-capable because, in either case, an unsafe transition could imply that future non-ECN-capable packets might get mangled.

The above advice on switching to sending non-ECN-capable packets but still responding to CE-markings unless they become continuous is not stated normatively (in capitals), because the best strategy might depend on experience of the most likely types of mangling, which can only be known at the time of deployment.

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 is advised to continue to send ECN-capable packets without having tested for mangling of the IP/ECN field on the SYN/ACK.

Invalid transitions of the IP/ECN field are defined in section 18 of [RFC3168] and repeated here for convenience:

- o the not-ECT codepoint changes;
- o either ECT codepoint transitions to not-ECT;
- o 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 [Mandalaril18].

Once a Data Sender has entered AccECN mode it is advised to check whether it is receiving continuous CE marking. Specifying exactly how to do this is beyond the scope of the present specification, but the sender might check whether the feedback for every packet it sends for the first three or four rounds indicates CE-marking. If continuous CE-marking is detected, for the remainder of the half-connection, the Data Sender ought to send non-ECN-capable packets and it is advised not to respond to any feedback of CE markings. The Data Sender might occasionally test whether it can resume sending ECN-capable packets. As always, once a host has entered AccECN mode, it MUST remain in the same feedback mode and it MUST continue to feed back any ECN markings on arriving packets.

All the fall-back behaviours in this section are necessary in case mangling of the IP/ECN field is asymmetric, which is currently common over some mobile networks [Mandalaril18]. 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.

#### 3.2.2.4. 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 counter ought to be non-zero.

If AccECN has been successfully negotiated, the Data Sender SHOULD check the value of the ACE counter in the first packet (with or without data) that arrives with `SYN=0`. If the value of this ACE field is zero (0b000), for the remainder of the half-connection the Data Sender ought to send non-ECN-capable packets and it is advised not to respond to any feedback of CE markings. Reason: the symptoms imply either potential mangling of the ECN fields in both the IP and TCP headers, or a broken remote TCP implementation. This advice is not stated normatively (in capitals), because the best strategy might depend on experience of the most likely types of mangling, which can only be known at the time of deployment.

If reordering occurs, "the first packet ... that arrives" will not necessarily be the same as the first packet in sequence order. The test has been specified loosely like this to simplify implementation, and because it would not have been any more precise to have specified the first packet in sequence order, which would not necessarily be the first ACE counter that the Data Receiver fed back anyway, given it might have been a retransmission. Usually, the server checks the ACK of the SYN/ACK from the client, while the client checks the first data segment from the server.

The possibility of re-ordering means that there is a small chance that the ACE field on the first packet to arrive is genuinely zero (without middlebox interference). This would cause a host to unnecessarily disable ECN for a half connection. Therefore, in environments where there is no evidence of the ACE field being zeroed, implementations can skip this test.

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.

#### 3.2.2.5. 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 or thinned out, the 3-bit counter in the ACE field might have cycled between two ACKs arriving at the Data Sender. The following safety procedures minimize this ambiguity.

##### 3.2.2.5.1. Data Receiver Safety Procedures

The following rules define when a Data Receiver in AccECN mode emits an ACK:

**Change-Triggered ACKs:** An AccECN Data Receiver SHOULD emit an ACK whenever a data packet marked CE arrives after the previous packet was not CE.

Even though this rule is stated as a "SHOULD", it is important for a transition to trigger an ACK if at all possible, The only valid exception to this rule is given below these bullets.

For the avoidance of doubt, this rule is deliberately worded to apply solely when `_data_` packets arrive, but the comparison with the previous packet includes any packet, not just data packets.

**Increment-Triggered ACKs:** An AccECN Data Receiver MUST emit an ACK if 'n' CE marks have arrived since the previous ACK. If there is newly delivered data to acknowledge, 'n' SHOULD be 2. If there is no newly delivered data to acknowledge, 'n' SHOULD be 3 and MUST be no less than 3. In either case, 'n' MUST be no greater than 7.

The above rules for when to send an ACK are designed to be complemented by those in Section 3.2.3.3, which concern whether the AccECN TCP Option ought to be included on ACKs.



If the arrivals of a number of data packets are all processed as one event, e.g. using large receive offload (LRO) or generic receive offload (GRO), both the above rules SHOULD be interpreted as requiring multiple ACKs to be emitted back-to-back (for each transition and for each repetition by 'n' CE marks). If this is problematic for high performance, either rule can be interpreted as requiring just a single ACK at the end of the whole receive event.

Even if a number of data packets do not arrive as one event, the 'Change-Triggered ACKs' rule could sometimes cause the ACK rate to be problematic for high performance (although high performance protocols such as DCTCP already successfully use change-triggered ACKs). The rationale for change-triggered ACKs is so that the Data Sender can rely on them to detect queue growth as soon as possible, particularly at the start of a flow. The 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 CE marks are infrequent, as is the case for most AQMs at the time of writing, or there are multiple marks in a row, the additional load will be low. However, marking patterns with numerous non-contiguous CE marks could increase the load significantly. One possible compromise would be for the receiver to heuristically detect whether the sender is in slow-start, then to implement change-triggered ACKs while the sender is in slow-start, and offload otherwise.

With ECN-capable pure ACKs [I-D.ietf-tcpm-generalized-ecn], the 'Increment-Triggered ACKs' rule could cause ECN-marked pure ACKs to trigger further ACKs. Although TCP normally only ACKs newly delivered data, in this case the ACKs of ACKs would feed back new congestion state. The minimum of 3 for 'n' in this case ensures that, even if there is pathological congestion in both directions, any resulting ping-pong of ACKs will be rapidly damped.

These ACKs of ACKs could be misidentified as duplicate ACKs in certain circumstances described below. Therefore, a host in AccECN mode that is sending ECN-capable pure ACKs SHOULD add one of the following additional checks when it tests whether an incoming pure ACK is a duplicate:

- o If SACK has been negotiated for the connection, but there is no SACK option on the incoming pure ACK, it is not a duplicate;
- o If timestamps are in use, and the incoming pure ACK echoes a timestamp older than the oldest unacknowledged data, it is not a duplicate.

In the unlikely event that neither SACK nor timestamps are in use, or if the implementation has opted not to include either of the above

two checks, it SHOULD NOT send ECN-capable pure ACKs. If it does, it could lead to false detection of duplicate ACKs, causing spurious retransmission(s) with a resulting unnecessary reduction in congestion window; but only in certain circumstances. Specifically, if TCP peer A has been sending data, then receiving, then within one round trip it starts sending again, and the ECN-capable pure ACKs it sent in the previous round encounter heavy enough congestion to trigger peer B to invoke the above 'n'-CE-mark rule. Also note that falsely considering these ACKs as duplicates would incorrectly imply that data left the network.

#### 3.2.2.5.2. Data Sender Safety Procedures

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, it SHOULD deem whether it cycled by taking the safest likely case under the prevailing conditions. 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 neutralize the effect of any action it took based on a conservative assumption that it later found to be incorrect.

The Data Sender can estimate how many packets (of any marking) an ACK acknowledges. If the ACE counter on an ACK seems to imply that the minimum number of newly CE-marked packets is greater than the number of newly acknowledged packets, the Data Sender SHOULD believe the ACE counter, unless it can be sure that it is counting all control packets correctly.

#### 3.2.3. The AccECN Option

The AccECN Option is defined as shown in Figure 4. The initial 'E' of each field name stands for 'Echo'.

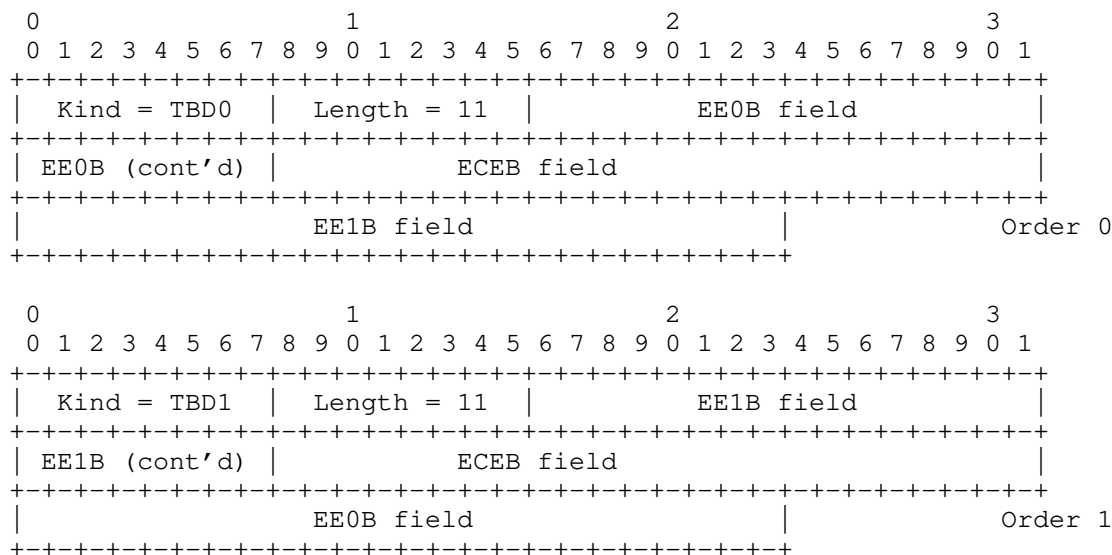


Figure 4: The AccECN TCP Option

Figure 4 shows two option field orders; order 0 and order 1. They both consists of three 24-bit fields. Order 0 provides the 24 least significant bits of the r.e0b, r.ceb and r.elb counters, respectively. Order 1 provides the same fields, but in the opposite order. On each packet, the Data Receiver can use whichever order is more efficient.

When a Data Receiver sends an AccECN Option, it MUST set the Kind field to TBD0 if using Order 0, or to TBD1 if using Order 1. These two new TCP Option Kinds are registered in Section 7 and called respectively AccECN0 and AccECN1.

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.4.

Whenever a Data Receiver sends an AccECN Option, the rules in Section 3.2.3.3 allow it to omit unchanged fields from the tail of the option, to help cope with option space limitations, 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	Type 0	Type 1
11	EE0B, ECEB, EE1B	EE1B, ECEB, EE0B
8	EE0B, ECEB	EE1B, ECEB
5	EE0B	EE1B
2	(empty)	(empty)

Fields included in AccECN TCP Options of each length and type

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.

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. For forward compatibility, 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, treating it as padding.

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. Support for the AccECN Option is particularly valuable over paths that introduce a high degree of ACK filtering, where the 3-bit ACE counter alone might sometimes be insufficient, when it is ambiguous whether it has wrapped. 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 Section 3.2.3.2.

#### 3.2.3.1. Encoding and Decoding Feedback in the AccECN Option Fields

Whenever the Data Receiver includes any of the counter fields (ECEB, EE0B, EE1B) in an AccECN Option, it MUST encode the 24 least significant bits of the current value of the associated counter into the field (respectively r.ceb, r.e0b, r.e1b).

Whenever the Data Sender receives ACK carrying an AccECN Option, it first checks whether the ACK has already been superseded by another ACK in which case it ignores the ECN feedback. If the ACK has not been superseded, the Data Sender normally decodes the fields in the AccECN Option as follows. For each field, it takes the least significant 24 bits of its associated local counter (s.ceb, s.e0b or s.e1b) and subtracts them from the counter in the associated field of the incoming AccECN Option (respectively ECEB, EE0B, EE1B), to work out the minimum positive increment it could apply to s.ceb, s.e0b or s.e1b (assuming the field in the option only wrapped at most once).

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, as specified in Section 3.2, any data on the SYN (SYN=1, ACK=0) is not included in any of the byte counters held locally for each ECN marking nor in the AccECN Option on the wire.

#### 3.2.3.2. Path Traversal of the AccECN Option

##### 3.2.3.2.1. Testing the AccECN Option during the Handshake

The TCP client MUST NOT include the AccECN TCP Option on the SYN. If there is somehow an AccECN Option on a SYN, it MUST be ignored when forwarded or received. (A fall-back strategy for the loss of the SYN, possibly due to middlebox interference, is specified in Section 3.1.4.)

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 on 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 omit the AccECN Option in any of the above three cases due to insufficient option space or 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.

### 3.2.3.2.2. 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 same AE, CWR and ECE TCP flags as 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. retrying the AccECN Option for a second time before fall-back - most appropriate during high levels of congestion). However, other fall-back strategies will need to follow all the rules in Section 3.1.5, which concern behaviour when SYNs or SYN/ACKs negotiating different types of feedback have been sent within the same connection.

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. [RFC9040] further discusses caching of TCP parameters and status information.

If a host falls back to not sending the AccECN Option, it will continue to process any incoming AccECN Options as normal.

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 is advised to 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 might 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 it ought to 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.

#### 3.2.3.2.3. Testing for Absence of the AccECN Option

If the TCP client has successfully negotiated AccECN but does not receive an AccECN Option on the SYN/ACK (e.g. because it has been stripped by a middlebox or not sent by the server), the client 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 3.2.2.5. 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 3.2.3.2.2).

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.

#### 3.2.3.2.4. Test for Zeroing of the AccECN Option

For a related test for invalid initialization of the ACE field, see Section 3.2.2.4

Section 3.2.1 required the Data Receiver to initialize the `r.e0b` and `r.e1b` counters to a non-zero value. Therefore, in either direction the initial value of the `EE0B` field or `EE1B` field in the AccECN Option (if one exists) ought to be non-zero. If AccECN has been negotiated:

- o the TCP server MAY check that the initial value of the `EE0B` field or the `EE1B` field is non-zero in the first segment that acknowledges sequence space that at least covers the ISN plus 1. If it runs a test and either initial value is zero, the server

will switch into a mode that ignores the AccECN Option for this half connection.

- o the TCP client MAY check the initial value of the EE0B field or the EE1B field is non-zero on the SYN/ACK. If it runs a test and either initial value 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 3.2.2.5.

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.

#### 3.2.3.2.5. Consistency between AccECN Feedback Fields

When the AccECN Option is available it ought to provide more unambiguous feedback. However, it supplements but does not replace the ACE field. An endpoint using AccECN feedback MUST always reconcile the information provided in the ACE field with that in any AccECN Option, so that the state of the ACE-related packet counter can be relied on if future feedback does not carry the AccECN Option.

If the AccECN option is present, the s.cep counter might increase more than expected from the increase of the s.ceb counter (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 5.3).

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), and if there is no explanation other than an invalid protocol transition due to some form of feedback mangling, 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.

### 3.2.3.3. Usage of the AccECN TCP Option

If a Data Receiver in AccECN mode intends to use the AccECN TCP Option to provide feedback, the rules below determine when it includes an AccECN TCP Option, and which fields to include, given other options might be competing for limited option space:

**Importance of Congestion Control:** AccECN is for congestion control, which SHOULD generally be considered important relative to other TCP options.

If SACK has been negotiated, and the smallest recommended AccECN Option would leave insufficient space for two SACK blocks on a particular ACK, the Data Receiver MUST give precedence to the SACK option (total 18 octets), because loss feedback is more critical.

**Recommended Simple Scheme:** The Data Receiver SHOULD include an AccECN TCP Option on every scheduled ACK if any byte counter has incremented since the last ACK. Whenever possible, it SHOULD include a field for every byte counter that has changed at some time during the connection (see examples later).

A scheduled ACK means an ACK that the Data Receiver would send by its regular delayed ACK rules. Recall that Section 1.3 defines an 'ACK' as either with data payload or without. But the above rule is worded so that, in the common case when most of the data is from a server to a client, the server only includes an AccECN TCP Option while it is acknowledging data from the client.

When available TCP option space is limited on particular packets, the recommended scheme will need to include compromises. To guide the implementer the rules below are ranked in order of importance, but the final decision has to be implementation-dependent, because tradeoffs will alter as new TCP options are defined and new use-cases arise.

**Necessary Option Length:** The Data Receiver MUST only include an AccECN TCP Option on a packet if it includes all the counter(s) that have incremented since the previous AccECN Option. It MUST only truncate unchanged fields from the right-hand tail of the option to preserve the order of the remaining fields (see Section 3.2.3);

**Change-Triggered AccECN TCP Options:** If an arriving packet increments a different byte counter to that incremented by the

previous packet, the Data Receiver SHOULD feed it back in an AccECN Option on the next scheduled ACK.

For the avoidance of doubt, this rule does not concern the arrival of control packets with no payload, because they cannot alter any byte counters.

Continual Repetition: Otherwise, if arriving packets continue to increment the same byte counter:

- \* the Data Receiver SHOULD include a counter that has continued to increment on the next scheduled ACK following a change-triggered AccECN TCP Option;
- \* while the same counter continues to increment, it SHOULD include the counter every  $n$  ACKs as consistently as possible, where  $n$  can be chosen by the implementer;
- \* It SHOULD always include an AccECN Option if the `r.ceb` counter is incrementing and it MAY include an AccECN Option if `r.ec0b` or `r.ec1b` is incrementing
- \* It SHOULD, include each counter at least once for every  $2^{22}$  bytes incremented to prevent overflow during continual repetition.

The above rules complement those in Section 3.2.2.5, which determine when to generate an ACK irrespective of whether an AccECN TCP Option is to be included.

The recommended scheme is intended as a simple way to ensure that all the relevant byte counters will be carried on any ACK that reaches the Data Sender, no matter how many pure ACKs are filtered or coalesced along the network path, and without consuming the space available for payload data with counter field(s) that have never changed.

As an example of the recommended scheme, if `ECT(0)` is the only codepoint that has ever arrived in the IP-ECN field, the Data Receiver will feed back an AccECN0 TCP Option with only the `EE0B` field on every packet. However, as soon as even one CE-marked packet arrives, on every packet that acknowledges new data it will start to include an option with two fields, `EE0B` and `ECEB`. As a second example, if the first packet to arrive happens to be CE-marked, the Data Receiver will have to arbitrarily choose whether to precede the `ECEB` field with an `EE0B` field or an `EE1B` field. If it chooses, say, `EEB0` but it turns out never to receive `ECT(0)`, it can start sending

EE1B and ECEB instead - it does not have to include the EE0B field if the r.e0b counter has never changed during the connection.

With the recommended scheme, if the data sending direction switches during a connection, there can be cases where the AccECN TCP Option that is meant to feed back the counter values at the end of a volley in one direction never reaches the other peer, due to packet loss. ACE feedback ought to be sufficient to fill this gap, given accurate feedback becomes moot after data transmission has paused.

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 any of the rules in this section.

### 3.3. AccECN Compliance Requirements for TCP Proxies, Offload Engines and other Middleboxes

#### 3.3.1. Requirements for TCP Proxies

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.

#### 3.3.2. Requirements for Transparent Middleboxes and TCP Normalizers

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 'normalize' the TCP wire protocol by checking that all values in header fields comply with a rather narrow interpretation of the TCP specifications that is also not always up to date.

A middlebox that is not normalizing the TCP protocol and does not itself act as a back-to-back pair of TCP endpoints (i.e. a middlebox that intends to be transparent or invisible at the transport layer) ought to forward the AccECN TCP Option unaltered, whether or not the length value matches one of those specified in Section 3.2.3, and whether or not the initial values of the byte-counter fields match those in Section 3.2.1. This is because blocking apparently invalid values prevents the standardized set of values being extended in future (given outdated normalizers would block updated hosts from using the extended AccECN standard).

A TCP normalizer is likely to block or alter an AccECN TCP Option if the length value or the initial values of its byte-counter fields do not match one of those specified in Section 3.2.3 or Section 3.2.1. However, to comply with the present AccECN specification, a middlebox MUST NOT change the ACE field; or those fields of the AccECN Option that are currently specified in Section 3.2.3; or any AccECN field covered by integrity protection (e.g. [RFC5925]).

### 3.3.3. Requirements for TCP ACK Filtering

A node that implements ACK filtering (aka. thinning or coalescing) SHOULD determine if an ACK is part of a connection using AccECN and SHOULD then preserve the correct operation of AccECN feedback. The following notes might help with each part of this requirement:

- o To determine whether a pure TCP ACK is part of an AccECN connection without resorting to connection tracking and per-flow state, a useful heuristic would be to check for a non-zero ECN field at the IP layer (because the ECN++ experiment only allows TCP pure ACKs to be ECN-capable if AccECN has been negotiated [I-D.ietf-tcpm-generalized-ecn]). This heuristic is simple and stateless. However, it might omit some AccECN ACKs, because it is only recommended but not obligatory to use ECN++ with AccECN - only deployment experience will tell. Also, TCP ACKs might be ECN-capable owing to some scheme other than AccECN, e.g. [RFC5690] or some future standards action. Again, only deployment experience will tell.
- o The main concern with preserving correct AccECN operation involves leaving enough ACKs for the Data Sender to work out whether the 3-bit ACE field has wrapped. ACE field wrap might be of less concern if packets also carry the AccECN TCP Option.

Note that the present specification of AccECN in TCP does not presume to rely on any of the above ACK filtering behaviour in the network (hence the use of 'SHOULD' rather than 'MUST' above), because it has to be robust against pre-existing network nodes that do not distinguish AccECN ACKs, and robust against ACK loss during overload more generally.

Section 5.2.1 of BCP 69 [RFC3449] gives best current practice on pure TCP ACK filtering. It gives no advice on ACKs carrying ECN feedback, other than that filtering ought to preserve the correct operation of ECN feedback, because at the time it said that "SACK and ECN remain areas of ongoing research". This section updates that best current practice for a TCP connection that supports AccECN feedback.

#### 3.3.4. Requirements for TCP Segmentation Offload

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').

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.

Current offload hardware ejects a segment from the coalescing process whenever the TCP ECN flags change. Thus Classic ECN causes offload to be inefficient. In data centres it has been fortunate for this 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 (but less likely the more short flows are in the mix). The ACE counter approach has been designed so that coalescing can continue over arbitrary patterns of marking and only needs to stop when the counter wraps. Nonetheless, until the particular offload hardware in use implements this more efficient approach, it is likely to be more efficient for AccECN connections to implement this counter-style logic using software segmentation offload.

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 ACE field has been designed as a counter so that it is straightforward for offload hardware to pass on the highest counter, and to push a segment from its cache before the counter wraps. The purpose of working towards standardized TCP ECN feedback is to reduce the risk for hardware developers, who would otherwise have to guess which scheme is likely to become dominant.

The above process has been designed to enable a continuing incremental deployment path - to more highly dynamic congestion control. Once offload hardware supports AccECN, it will be able to coalesce efficiently for any sequence of marks, instead of relying for efficiency on the long marking sequences from step marking. In the next stage, marking can evolve from a step to a ramp function. That in turn will allow host congestion control algorithms to respond faster to dynamics, while being backwards compatible with existing host algorithms.

#### 4. Updates to RFC 3168

Normative statements in the following sections of RFC3168 are updated by the present AccECN specification:

- o The whole of "6.1.1 TCP Initialization" of [RFC3168] is updated by Section 3.1 of the present specification.
- o In "6.1.2. The TCP Sender" of [RFC3168], all mentions of a congestion response to an ECN-Echo (ECE) ACK packet are updated by Section 3.2 of the present specification to mean an increment to the sender's count of CE-marked packets, s.cep. And the requirements to set the CWR flag no longer apply, as specified in Section 3.1.5 of the present specification. Otherwise, the remaining requirements in "6.1.2. The TCP Sender" still stand.

It will be noted that RFC 8311 already updates, or potentially updates, a number of the requirements in "6.1.2. The TCP Sender". Section 6.1.2 of RFC 3168 extended standard TCP congestion control [RFC5681] to cover ECN marking as well as packet drop. Whereas, RFC 8311 enables experimentation with alternative responses to ECN marking, if specified for instance by an experimental RFC on the IETF document stream. RFC 8311 also strengthened the statement that "ECT(0) SHOULD be used" to a "MUST" (see [RFC8311] for the details).

- o The whole of "6.1.3. The TCP Receiver" of [RFC3168] is updated by Section 3.2 of the present specification, with the exception of the last paragraph (about congestion response to drop and ECN in the same round trip), which still stands. Incidentally, this last paragraph is in the wrong section, because it relates to TCP sender behaviour.
- o The following text within "6.1.5. Retransmitted TCP packets":

"the TCP data receiver SHOULD ignore the ECN field on arriving data packets that are outside of the receiver's current window."

is updated by more stringent acceptability tests for any packet (not just data packets) in the present specification. Specifically, in the normative specification of AccECN (Section 3) only 'Acceptable' packets contribute to the ECN counters at the AccECN receiver and Section 1.3 defines an Acceptable packet as one that passes the acceptability tests in both [RFC0793] and [RFC5961].

- o Sections 5.2, 6.1.1, 6.1.4, 6.1.5 and 6.1.6 of [RFC3168] prohibit use of ECN on TCP control packets and retransmissions. The present specification does not update that aspect of RFC 3168, but it does say what feedback an AccECN Data Receiver ought to provide if it receives an ECN-capable control packet or retransmission. This ensures AccECN is forward compatible with any future scheme that allows ECN on these packets, as provided for in section 4.3 of [RFC8311] and as proposed in [I-D.ietf-tcpm-generalized-ecn].

## 5. Interaction with TCP Variants

This section is informative, not normative.

### 5.1. 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:

- o the TCP client has to have requested AccECN support on the SYN
- o it (the server) has to 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.

## 5.2. Compatibility with TCP Experiments and Common TCP Options

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 3.2.3.3 provides guidance on how important it is to send an AccECN Option relative to other options, and which fields are more important to include.

Implementers of TFO need to take careful note of the recommendation in Section 3.2.2.1. That section recommends that, if the client has successfully negotiated AccECN, when acknowledging the SYN/ACK, even if it has data to send, it sends a pure ACK immediately before the data. Then it can reflect the IP-ECN field of the SYN/ACK on this pure ACK, which allows the server to detect ECN mangling. Note that, as specified in Section 3.2, any data on the SYN (SYN=1, ACK=0) is not included in any of the byte counters held locally for each ECN marking, nor in the AccECN Option on the wire.

## 5.3. 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:

- o 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 standardization 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 therefore ought to only be done sparingly.
- o 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 neutralize any advantage that any of these three parties would otherwise gain.

ConEx is an experimental change to the Data Sender that would be 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.

- o The standards track 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 ECN Nonce [RFC3540] was proposed to ensure integrity of congestion feedback. With minor changes AccECN could be optimized 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 generalized 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.

## 6. 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. Within the constraints of the change-triggered ACK rules, the receiver can control how frequently it sends the AccECN TCP Option and therefore to some extent 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. And if data or ACKs are reordered, stale congestion information can be identified and ignored.

**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 options are needed 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. Following ACK reordering, the Data Sender can reconstruct the order in which feedback was sent, but not until all the missing feedback has arrived.

**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 3WSH 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. Then, the designers of security devices can understand which currently unused values might appear in future. So, even if they choose to treat such values as anomalous while they are not widely used, any blocking will at least be under policy control not hard-coded. Then, if previously unused values start to appear on the Internet (or in standards), such policies could be quickly reversed.

## 7. IANA Considerations

This document reassigns bit 7 of the TCP header flags to the AccECN protocol. 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:

Bit	Name	Reference
7	AE (Accurate ECN)	RFC XXXX

#### TCP header flag reassignment

[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 two new TCP options for AccECN, assigned values of TBD0 and TBD1 (decimal) from the TCP option space. These values are defined as:

Kind	Length	Meaning	Reference
TBD0	N	Accurate ECN Order 0 (AccECN0)	RFC XXXX
TBD1	N	Accurate ECN Order 1 (AccECN1)	RFC XXXX

#### New TCP Option assignments

[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 implementations using experimental option 254 per [RFC6994] with the single magic number 0xACCE (16 bits), as allocated in the IANA "TCP Experimental Option Experiment Identifiers (TCP ExIDs)" registry, SHOULD migrate to use these new option kinds (TBD0 & TBD1).

[TO BE REMOVED: The description of the 0xACCE value in the TCP ExIDs registry should be changed to "AccECN (current and new implementations SHOULD use option kinds TBD0 and TBD1)" at the following location: <https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml#tcp-exids> ]

## 8. 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 3.2.2.5). 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 present, the AccECN Data Sender is then required to switch to more conservative assumptions about wrap of congestion indication counters (see Section 3.2.2.5 and Appendix A.2).

Section 5.1 describes how a TCP server can negotiate AccECN and use the SYN cookie method for mitigating SYN flooding attacks.

There is concern that ECN feedback 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 5.3 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. Assuring that Data Senders respond appropriately to ECN feedback is possible, but the scope of the present document is confined to the feedback protocol, and excludes the response to this feedback.

In Section 3.2.3 a Data Sender is allowed to ignore an unrecognized TCP AccECN Option length and read as many whole 3-octet fields from it as possible up to a maximum of 3, treating the remainder as padding. This opens up a potential covert channel of up to 29B (40 - (2+3\*3))B. However, it is really an overt channel (not hidden) and it is no different to the use of unknown TCP options with unknown option lengths in general. Therefore, where this is of concern, it can already be adequately mitigated by regular TCP normalizer technology (see Section 3.3.2).

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. A covert channel can be used to compromise privacy. However, as explained above, undefined TCP options in general open up such channels and common techniques are available to close them off.

There is a potential concern that a Data Receiver could deliberately omit the AccECN Option pretending that it had been stripped by a middlebox. No known way can yet be contrived for a receiver to take advantage of this behaviour, which seems to always degrade its own performance. However, the concern is mentioned here for completeness.

## 9. Acknowledgements

We want to thank Koen De Schepper, Praveen Balasubramanian, Michael Welzl, Gorrry Fairhurst, David Black, Spencer Dawkins, Michael Scharf, Michael Tuexen, Yuchung Cheng, Kenjiro Cho, Olivier Tilmans, Ilpo Jaervinen, Neal Cardwell, Yoshifumi Nishida, Martin Duke and Jonathan Morton 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 Comcast Innovation Fund, 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), and the Research Council of Norway through the TimeIn 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.

## 10. 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.

## 11. References

### 11.1. Normative References

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

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/info/rfc5681>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

#### 11.2. Informative References

- [I-D.ietf-tcpm-generalized-ecn]  
Bagnulo, M. and B. Briscoe, "ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control Packets", draft-ietf-tcpm-generalized-ecn-09 (work in progress), January 2022.
- [I-D.ietf-tsvwg-l4s-arch]  
Briscoe, B., Schepper, K. D., Bagnulo, M., and G. White, "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture", draft-ietf-tsvwg-l4s-arch-17 (work in progress), March 2022.
- [Mandalari18]  
Mandalari, A., Lutu, A., Briscoe, B., Bagnulo, M., and Oe. Alay, "Measuring ECN++: Good News for ++, Bad News for ECN over Mobile", IEEE Communications Magazine , March 2018.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, DOI 10.17487/RFC2018, October 1996, <<https://www.rfc-editor.org/info/rfc2018>>.
- [RFC3449] Balakrishnan, H., Padmanabhan, V., Fairhurst, G., and M. Sooriyabandara, "TCP Performance Implications of Network Path Asymmetry", BCP 69, RFC 3449, DOI 10.17487/RFC3449, December 2002, <<https://www.rfc-editor.org/info/rfc3449>>.

- [RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, DOI 10.17487/RFC3540, June 2003, <<https://www.rfc-editor.org/info/rfc3540>>.
- [RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <<https://www.rfc-editor.org/info/rfc4987>>.
- [RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", RFC 5562, DOI 10.17487/RFC5562, June 2009, <<https://www.rfc-editor.org/info/rfc5562>>.
- [RFC5690] Floyd, S., Arcia, A., Ros, D., and J. Iyengar, "Adding Acknowledgement Congestion Control to TCP", RFC 5690, DOI 10.17487/RFC5690, February 2010, <<https://www.rfc-editor.org/info/rfc5690>>.
- [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <<https://www.rfc-editor.org/info/rfc5925>>.
- [RFC5961] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", RFC 5961, DOI 10.17487/RFC5961, August 2010, <<https://www.rfc-editor.org/info/rfc5961>>.
- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <<https://www.rfc-editor.org/info/rfc6824>>.
- [RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, DOI 10.17487/RFC6994, August 2013, <<https://www.rfc-editor.org/info/rfc6994>>.
- [RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <<https://www.rfc-editor.org/info/rfc7413>>.
- [RFC7560] Kuehlewind, M., Ed., Scheffenegger, R., and B. Briscoe, "Problem Statement and Requirements for Increased Accuracy in Explicit Congestion Notification (ECN) Feedback", RFC 7560, DOI 10.17487/RFC7560, August 2015, <<https://www.rfc-editor.org/info/rfc7560>>.



- [RFC7713] Mathis, M. and B. Briscoe, "Congestion Exposure (ConEx) Concepts, Abstract Mechanism, and Requirements", RFC 7713, DOI 10.17487/RFC7713, December 2015, <<https://www.rfc-editor.org/info/rfc7713>>.
- [RFC8257] Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <<https://www.rfc-editor.org/info/rfc8257>>.
- [RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <<https://www.rfc-editor.org/info/rfc8311>>.
- [RFC8511] Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst, "TCP Alternative Backoff with ECN (ABE)", RFC 8511, DOI 10.17487/RFC8511, December 2018, <<https://www.rfc-editor.org/info/rfc8511>>.
- [RFC9040] Touch, J., Welzl, M., and S. Islam, "TCP Control Block Interdependence", RFC 9040, DOI 10.17487/RFC9040, July 2021, <<https://www.rfc-editor.org/info/rfc9040>>.

## 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

$$\text{ECEB} = \text{r.ceb} \% \text{DIVOPT}$$

where `'%'` is the remainder operator.

On the arrival of an AccECN Option, the Data Sender first makes sure the ACK has not been superseded in order to avoid winding the `s.ceb` counter backwards. It uses the TCP acknowledgement number and any SACK options to calculate `newlyAackedB`, the amount of new data that the ACK acknowledges in bytes (`newlyAackedB` can be zero but not negative). If `newlyAackedB` is zero, either the ACK has been superseded or CE-marked packet(s) without data could have arrived. To break the tie for the latter case, the Data Sender could use timestamps (if present) to work out `newlyAackedT`, the amount of new time that the ACK acknowledges. If the Data Sender determines that the ACK has been superseded it ignores the AccECN Option. Otherwise, the Data Sender calculates the minimum non-negative difference `d.ceb` between the ECEB field and its local `s.ceb` counter, using modulo arithmetic as follows:

```
if ((newlyAcedB > 0) || (newlyAcedT > 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 % DIVOPT = 1  
d.ceb = (1461 + 2^24 - 1) % 2^24  
       = 1460  
s.ceb = 33,554,433 + 1460  
       = 33,555,893
```

In practice an implementation might use heuristics to guess the feedback in missing ACKs, then when it subsequently receives feedback it might find that it needs to correct its earlier heuristics as part of the decoding process. The above decoding process does not include any such heuristics.

#### 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.cep into the ACE field, and how the Data Sender in AccECN mode could decode the ACE field into its s.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 3.2.2.5 and Section 3.2.3.2); 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:

```
DIVACE = 2^3
```

Every time an Acceptable CE marked packet arrives (Section 3.2.2.2), the Data Receiver increments its local value of r.cep by 1. It repeats the same value of ACE in every subsequent ACK until the next CE marking arrives, where

ACE = r.cep % 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 ensures the ACK has not been superseded using the TCP acknowledgement number, any SACK options and timestamps (if available) to calculate newlyAckedB, as in Appendix A.1. If the ACK has not been superseded, the Data Sender calculates the minimum difference d.cep between the ACE field and its local s.cep counter, using modulo arithmetic as follows:

```
if ((newlyAckedB > 0) || (newlyAckedT > 0))  
    d.cep = (ACE + DIVACE - (s.cep % DIVACE)) % DIVACE
```

Section 3.2.2.5 expects the Data Sender to assume that the ACE field cycled if it is the safest likely case under prevailing conditions. The 3-bit ACE field in an arriving ACK could have cycled and become ambiguous to the Data Sender if a sequence 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 missing ACKs were 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 for implementation-dependent interpretation. A Data Sender might take account of the prevailing size of data segments and the prevailing CE marking rate just before the sequence of missing ACKs. 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

```
dSafer.cep = newlyAckedPkt - ((newlyAckedPkt - d.cep) % 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  $10 - ((10-2) \% 8) = 10$ ).

Note that checks would need to be added to the above pseudocode for  $(d.cep > newlyAkedPkt)$ , which could occur if `newlyAkedPkt` had been wrongly estimated using an inappropriate packet size.

ACKs that acknowledge a large stretch of packets might be common in data centres to achieve a high packet rate or might be due to ACK thinning by a middlebox. In these cases, cycling of the ACE field would often appear to have been possible, so the above algorithm would be over-conservative, leading to a false high marking rate and poor performance. Therefore it would be reasonable to only use `dSafer.cep` rather than `d.cep` if the moving average of `newlyAkedPkt` was well below 8.

Implementers could build in more heuristics to estimate prevailing average segment size and prevailing ECN marking. For instance, `newlyAkedPkt` in the above formula could be replaced with `newlyAkedPktHeur = newlyAkedPkt*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 `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.cep += dSafer.cep
```

If missing acknowledgement numbers arrive later (due to reordering), Section 3.2.2.5 says "the Data Sender MAY attempt to neutralize 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.

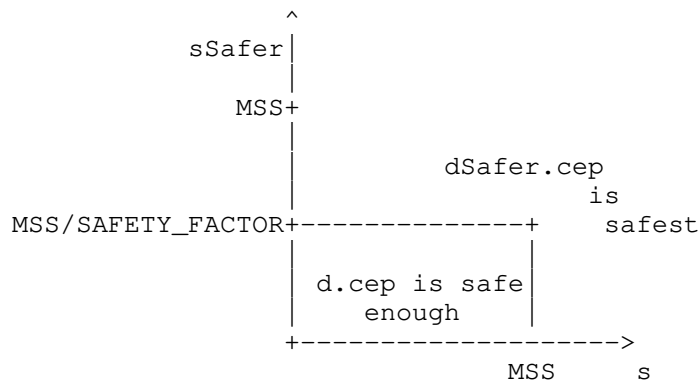
#### A.2.2. Safety Algorithm with the AccECN Option

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. If for some reason

it needs CE-marked packets, if `dSafer.cep` is different from `d.cep`, it can determine whether `d.cep` is likely to be a safe enough estimate by checking whether the average marked segment size ( $s = d.ceb/d.cep$ ) is less than the MSS (where `d.ceb` is the amount of newly CE-marked bytes - see Appendix A.1). Specifically, it could use the following algorithm:

```
SAFETY_FACTOR = 2
if (dSafer.cep > d.cep) {
    if (d.ceb <= MSS * d.cep) { % Same as (s <= MSS), but no DBZ
        sSafer = d.ceb/dSafer.cep
        if (sSafer < MSS/SAFETY_FACTOR)
            dSafer.cep = d.cep % d.cep is a safe enough estimate
    } % else
        % No need for else; dSafer.cep is already correct,
        % because d.cep must have been too small
}
```

The chart below shows when the above algorithm will consider `d.cep` can replace `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=1460 [B]`:

- o if `d.cep=0`, `dSafer.cep=8` and `d.ceb=1460`, then  $s=\text{infinity}$  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`.

- o if  $d_{cep}=2$ ,  $d_{safer_{cep}}=10$  and  $d_{ceb}=1460$ , then  $s=730$  and  $s_{safer}=146$ .  
Therefore  $d_{cep}$  is safe enough, because the average size of 10 data segments is unlikely to have been as small as  $MSS/10$ .
- o if  $d_{cep}=7$ ,  $d_{safer_{cep}}=15$  and  $d_{ceb}=10200$ , then  $s=1457$  and  $s_{safer}=680$ .  
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. Some possible ways to calculate  $s_{ave}$  are outlined below. The precise details will depend on why an estimate of marked bytes is needed.

The implementation 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 is reset once per RTT. Either way, it would estimate  $s_{ave}$  as:

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

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:

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

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 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.

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 retransmissions 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 ought to 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 approach to negotiate the highest version TCP ECN feedback scheme that both ends support, as justified below. It follows from the original TCP ECN capability negotiation [RFC3168], in which the client set the 2 least significant of the original 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.

ECN originally used header flags rather than a TCP option because it was considered more efficient to use a header flag for 1 bit of feedback per ACK, and this bit could be overloaded to indicate support for ECN during the handshake. During the development of ECN, 1 bit crept up to 2, in order to deliver the feedback reliably and to work round some broken hosts that reflected the reserved flags during the handshake.

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 had 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 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. [Mandalaril8], 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 3.2.2.4 ensures that 3 unused codepoints on the ACK of the SYN/ACK, 6 unused values on the first SYN=0 data packet from the client and 7 unused values on the first SYN=0 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 3.2.3.2.4) 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 3.1.3 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  
Independent  
UK

EMail: [ietf@bobbriscoe.net](mailto:ietf@bobbriscoe.net)  
URI: <http://bobbriscoe.net/>

Mirja Kuehlewind  
Ericsson  
Germany

EMail: [ietf@kuehlewind.net](mailto:ietf@kuehlewind.net)

Richard Scheffenegger  
NetApp  
Vienna  
Austria

EMail: [Richard.Scheffenegger@netapp.com](mailto:Richard.Scheffenegger@netapp.com)

Network Working Group  
Internet-Draft  
Obsoletes: 5562 (if approved)  
Intended status: Experimental  
Expires: 4 August 2022

M. Bagnulo  
UC3M  
B. Briscoe  
Independent  
31 January 2022

ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control  
Packets  
draft-ietf-tcpm-generalized-ecn-09

## 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 4 August 2022.

## Copyright Notice

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

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

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

## Table of Contents

1. 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 . . . . .	8
3.2. Sender Behaviour . . . . .	8
3.2.1. SYN (Send) . . . . .	10
3.2.2. SYN-ACK (Send) . . . . .	13
3.2.3. Pure ACK (Send) . . . . .	14
3.2.4. Window Probe (Send) . . . . .	16
3.2.5. FIN (Send) . . . . .	16
3.2.6. RST (Send) . . . . .	17
3.2.7. Retransmissions (Send) . . . . .	17
3.2.8. General Fall-back for any Control Packet or Retransmission . . . . .	18
3.3. Receiver Behaviour . . . . .	18
3.3.1. Receiver Behaviour for Any TCP Control Packet or Retransmission . . . . .	18
3.3.2. SYN (Receive) . . . . .	19
3.3.3. Pure ACK (Receive) . . . . .	20
3.3.4. FIN (Receive) . . . . .	20
3.3.5. RST (Receive) . . . . .	20
3.3.6. Retransmissions (Receive) . . . . .	21
4. Rationale . . . . .	21
4.1. The Reliability Argument . . . . .	21
4.2. SYNs . . . . .	22
4.2.1. Argument 1a: Unrecognized CE on the SYN . . . . .	22
4.2.2. Argument 1b: ECT Considered Invalid on the SYN . . . . .	23
4.2.3. Caching Strategies for ECT on SYNs . . . . .	25
4.2.4. Argument 2: DoS Attacks . . . . .	27
4.3. SYN-ACKs . . . . .	28
4.3.1. Possibility of Unrecognized CE on the SYN-ACK . . . . .	28

4.3.2.	Response to Congestion on a SYN-ACK . . . . .	29
4.3.3.	Fall-Back if ECT SYN-ACK Fails . . . . .	30
4.4.	Pure ACKs . . . . .	30
4.4.1.	Mechanisms to Respond to CE-Marked Pure ACKs . . . . .	32
4.4.2.	Summary: Enabling ECN on Pure ACKs . . . . .	35
4.5.	Window Probes . . . . .	36
4.6.	FINs . . . . .	37
4.7.	RSTs . . . . .	37
4.8.	Retransmitted Packets. . . . .	38
4.9.	General Fall-back for any Control Packet . . . . .	39
5.	Interaction with popular variants or derivatives of TCP . . . . .	40
5.1.	IW10 . . . . .	41
5.2.	TFO . . . . .	42
5.3.	L4S . . . . .	42
5.4.	Other transport protocols . . . . .	43
6.	Security Considerations . . . . .	43
7.	IANA Considerations . . . . .	43
8.	Acknowledgments . . . . .	44
9.	References . . . . .	44
9.1.	Normative References . . . . .	44
9.2.	Informative References . . . . .	45
	Authors' Addresses . . . . .	48

## 1. Introduction

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

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

ECN++ uses a sender-only deployment model. It works whether the two ends of the TCP connection use classic ECN feedback [RFC3168] or Accurate ECN feedback (AccECN [I-D.ietf-tcpm-accurate-ecn]), the two ECN feedback mechanisms for TCP being standardized at the time of writing.

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

ECN++ is designed for compatibility with a number of latency improvements to TCP such as TCP Fast Open (TFO [RFC7413]), initial window of 10 SMSS (IW10 [RFC6928]) and Low latency Low Loss Scalable Transport (L4S [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.

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

Finally, there are ongoing experimental efforts to promote the adoption of a slightly modified variant of DCTCP (and similar congestion controls) over the Internet to achieve low latency, low loss and scalable throughput (L4S) for all communications [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 be classified into the L4S queue that is expected to have lower latency, while the packets controlling and retransmitting these data packets would still get stuck behind the queue induced by non-L4S-enabled TCP traffic.

## 1.2. Experiment Goals

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

- \* How SYNs, Window probes, pure ACKs, FINs, RSTs and retransmissions that carry the ECT(0), ECT(1) or CE codepoints are processed by the TCP endpoints and the network (including routers, firewalls and other middleboxes). In particular we would like to learn if these packets are frequently blocked or if these packets are usually forwarded and processed.
- \* The scale of deployment of the different flavours of ECN, including [RFC3168], [RFC5562], [RFC3540] and [I-D.ietf-tcpm-accurate-ecn].



- \* How much the performance of TCP communications is improved by allowing ECN marking of each packet type.
- \* To identify any issues (including security issues) raised by enabling ECN marking of these packets.
- \* To conduct the specific experiments identified in the text by the strings "EXPERIMENTATION NEEDED" or "MEASUREMENTS NEEDED".

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

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

### 1.3. Document Structure

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

## 2. Terminology

The 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 [RFC8174].

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.

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

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

Window probe and retransmission are abbreviated to W Probe and Re-XMT.  
 \* For a SYN, "negotiated" means "requested".

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

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

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

### 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

If the SYN sent by a TCP initiator does not attempt to negotiate Accurate ECN feedback, or does not use an equivalent safety mechanism, it MUST still comply with RFC 3168, which says that a TCP initiator "MUST NOT set ECT on a SYN".

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

See Section 4.2 for discussion and rationale.

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

### 3.2.1.2. Caching where to use ECT on SYNs

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

Until AccECN servers become widely deployed, a TCP initiator that sets ECT on a SYN (which typically implies the same SYN also requests AccECN, as above) SHOULD also maintain a cache entry per server to record servers that it is not worth sending an ECT SYN to, e.g. because they do not support AccECN and therefore have no logic for congestion markings on the SYN. Mobile hosts MAY maintain a cache entry per access network to record 'non-ECT SYN' entries against proxies (see Section 4.2.3). This cache can be implemented as part of the shared state across multiple TCP connections, following [RFC2140].

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

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

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

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

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

### 3.2.1.3. SYN Congestion Response

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

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

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

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

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

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

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

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

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

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

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

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

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

### 3.2.2. SYN-ACK (Send)

#### 3.2.2.1. Setting ECT on the SYN-ACK

For the ECN++ experiment, the TCP implementation will set ECT on SYN-ACKs. 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 4.3.2.



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

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

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

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

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

The server MAY cache failed connection attempts, e.g. per client access network. 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.

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

#### 3.2.7. Retransmissions (Send)

For the ECN++ experiment, the TCP sender will set ECT on retransmitted segments. 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 the present RFC 8311 experiment to be useful, TCP receivers will need to satisfy the following requirements:

- \* Any TCP implementation SHOULD accept receipt of any valid TCP control packet or retransmission irrespective of its IP/ECN field. If any existing implementation does not, it SHOULD be updated to do so.
- \* A TCP implementation taking part in the experiments proposed here MUST accept receipt of any valid TCP control packet or retransmission irrespective of its IP/ECN field.

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

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

### 3.3.2. SYN (Receive)

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

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

- \* Any TCP server implementation SHOULD accept receipt of a valid SYN that requests ECN support for the connection, irrespective of the IP/ECN field of the SYN. If any existing implementation does not, it SHOULD be updated to do so.
- \* A TCP implementation taking part in the ECN++ experiment MUST accept receipt of a valid SYN, irrespective of its IP/ECN field.
- \* If the SYN is CE-marked and the server has no logic to feed back a CE mark on a SYN-ACK (e.g. it does not support AccECN), it has to ignore the CE-mark (the client detects this case and behaves conservatively in mitigation - see Section 3.2.1.3).

Rationale: At the time of the writing, some implementations of TCP servers (see Section 4.2.2.2) assume that, if a host receives a SYN with a non-zero IP/ECN field, it must be due to network mangling, and they disable ECN for the rest of the connection. Section 4.2.2.2 cites a measurement study run in 2017 that found no occurrence of this type of network mangling. However, a year earlier, when ECN was

enabled on connections from Apple clients, there was a case of a whole network that re-marked the ECN field of every packet to CE (it was rapidly fixed).

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

### 3.3.3. Pure ACK (Receive)

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

- \* Any TCP implementation SHOULD accept receipt of a pure ACK with a non-zero 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]). 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 into feeding back spoofed congestion indications to the sender, which in turn would be fooled into continually reducing its congestion window.

## 4. Rationale

This section is informative, not normative. It presents counter-arguments against the justifications in the RFC series for disabling ECN on TCP control segments and retransmissions. It also gives rationale for why ECT is safe on control segments that have not, so far, been mentioned in the RFC series. 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. SYNs

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

"First, when the TCP SYN packet is sent, there are no guarantees that the other TCP endpoint (node B in Figure 2) is ECN-Capable, or that it would be able to understand and react if the ECN CE codepoint was set by a congested router.

Second, the ECN-Capable codepoint in TCP SYN packets could be misused by malicious clients to "improve" the well-known TCP SYN attack. By setting an ECN-Capable codepoint in TCP SYN packets, a malicious host might be able to inject a large number of TCP SYN packets through a potentially congested ECN-enabled router, congesting it even further."

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

##### 4.2.1. Argument 1a: Unrecognized CE on the SYN

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

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

1. An AccECN server uses the 3 '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 to set ECT on a SYN, it only finds out whether the server would have supported it afterwards. The trick the client uses for ECN++ is to set ECT on the SYN optimistically then, if the SYN-ACK reveals that the server wouldn't have understood CE on the SYN, the client responds conservatively as if the SYN was marked with CE.

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

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

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

Given, 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 SYNs (see the last paragraph of Section 3.2.1.2).

#### 4.2.2.2. ECT on SYN Considered Invalid by Servers

A study conducted in Nov 2017 [Kuehlewindl8] 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 negotiation." Of course, some of the 84% might be due to similar code in other OSs.

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

Ironically, networks rarely seem to alter the IP/ECN field on a SYN from zero to non-zero anyway. In a study conducted in Jan-May 2017 over millions of paths from vantage points in a few dozen mobile and fixed networks [Mandalaril8], no such transition was observed. With such a small or non-existent incidence of this sort of network mangling, it would be preferable to report any residual problem paths so that they can be fixed.

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

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

#### 4.2.3. Caching Strategies for ECT on SYNs

Given the server handling of ECN on SYNs outlined in Section 4.2.2.2 above, an initiator might combine AccECN with three candidate 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 'over-strict' ECN test.

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

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

a. No cache.

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

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

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

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

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

- \* The "pessimistic ECT and cache successes" strategy (S1) suffers from exposing the initial SYN to the prevailing loss level, even if the server supports ECT on SYNs, but only on the first connection to each AccECN server. If AccECN becomes widely deployed on servers, SYNs to those AccECN servers that are less frequently used by the client and therefore don't fit in the cache will not benefit from ECN protection at all.
- \* The "optimistic ECT without a cache" strategy (S2A) is the simplest. It would satisfy the goal of an implementer who is solely interested in low latency using AccECN and ECN++ and is not concerned about fall-back to Classic ECN.

- \* The "optimistic ECT and cache failures" strategy (S2B) exploits ECT on SYNs from the very first attempt. But if the server turns out to be 'over-strict' it will disable ECN for the connection, but only for the first connection if it's one of the client's more popular servers that fits in the cache. If the server turns out not to support AccECN, the initiator has to conservatively limit its initial window, but again only for the first connection if it's one of the client's more popular servers (and anyway this rarely makes any difference when most client requests fit in a single packet).

Note that, if AccECN deployment grows, 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](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, the more data packets that the client sends in its IW, the more likely at least one will be CE marked, leading it to exit slow-start early. On the other hand, it is highly unlikely that the SYN-ACK itself pushed the AQM into congestion, so it will be safe to introduce another single segment immediately (1 RTT after the SYN-ACK). Therefore, starting to probe for capacity with a slow start from an initial window of 1 segment seems appropriate to the circumstances. This is the approach adopted in Section 3.2.2.

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

#### 4.3.3. Fall-Back if ECT SYN-ACK Fails

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."

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 feed back 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: As a general rule, TCP does not ACK a pure ACK. However,

even if the receiver of a CE-mark on a pure ACK does not feed it back immediately, it could still include it within subsequent feedback, for instance when it later sends a 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 takes into account all types of packets, including pure ACKs. CE-marked pure ACKs will solely increment the packet counter; not any byte counter, because by definition they contain no bytes of data.

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 (assuming the congestion control supports a weighted response, e.g. [RFC8257]). For instance, one could calculate the fraction of CE-marked bytes (headers and data) over each round trip (say) as follows:

$$\frac{(\text{CE-marked header bytes} + \text{CE-marked data bytes})}{(\text{all header bytes} + \text{all data bytes})}$$

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.

This specification uses a safe approach. Allowing hosts to set ECT on Pure ACKs without a feasible response mechanism could result in risk. It would certainly improve the flow's own performance, but it would slightly increase potential harm to others. Moreover, it would set an undesirable precedent for setting ECT on packets with no mechanism to respond to any resulting congestion signals. 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.

#### 4.8. Retransmitted Packets.

RFC 3168 says the sender "MUST NOT" set ECT on retransmitted packets. The rationale for this consumes nearly 2 pages of RFC 3168, so the reader is referred to section 6.1.5 of RFC 3168, rather than quoting it all here. There are essentially three arguments, namely: reliability; DoS attacks; and 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 [Mandalaril8]. 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 another 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 will typically only set ECT on the SYN if it requests AccECN support. If, however, the SYN-ACK tells the initiator that the responder does not support AccECN, Section 3.2.1.1 advises the initiator to conservatively reduce its initial window, preferably to 1 SMSS because, if the SYN was CE-marked, the SYN-ACK has no way to feed that back.

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

- \* as long as the initiator is caching failures to negotiate AccECN, subsequent attempts to access the same server will not use ECT on the SYN anyway, so there will no longer be any need to conservatively reduce IW;
- \* currently, at least for web sessions, it is extremely rare for a TCP initiator (client) to have more than one data segment to send at the start of a TCP connection (see Fig 3 in [Manzoor17]) - IW10 is primarily exploited by TCP servers.

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

- \* Feedback that the SYN-ACK was CE-marked is an explicit indication that the queue has been building, not just uncertainty due to absence of feedback;
- \* Given it is now likely that a queue already exists, the more data packets that the server sends in its IW, the more likely at least one will be CE marked, leading it to exit slow-start early.

Experimentation will be needed to determine the best strategy. It should be noted that experience from recent congestion avoidance experiments where the window is reduced by less than half is not necessarily applicable to a flow start scenario. Reducing cwnd by less is one thing. Reducing an increase in cwnd by less is another.

## 5.2. TFO

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

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

- \* The handling of ECN marking on a SYN is no different whether or not it carries data.
- \* In response to any CE-marking on the SYN-ACK, the responder adopts the normal response to congestion, as discussed in Section 7.2 of [RFC7413].

## 5.3. L4S

A Low Latency Low Loss Scalable throughput (L4S) variant of TCP such as TCP Prague [PragueLinux] is mandated to negotiate AccECN feedback, and strongly recommended to use ECN++ [I-D.ietf-tsvwg-ecn-l4s-id].

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

The normative specification for ECT on a SYN in Section 3.2.1 recommends the "optimistic ECT and cache failures" strategy (S2B defined in Section 4.2.3) for the general Internet. However, if a user's Internet access bottleneck supported L4S ECN but not Classic ECN, the "optimistic ECT without a cache" strategy (S2A) would make most sense, because there would be little point trying to avoid the 'over-strict' test and negotiate Classic ECN, if L4S ECN but not Classic ECN was available on that user's access link (as is the case with Low Latency DOCSIS [DOCSIS3.1]).

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

#### 5.4. Other transport protocols

Experience from experiments on adding ECN support to all TCP packets ought to be directly transferable between TCP and other transport protocols, 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 [RFC9000] is another standards track transport protocol offering similar services to TCP but intended to exploit some of the benefits of running over UDP. Building on the arguments in the current draft, a QUIC sender sets ECT(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.

Section 3.2.6 on sending TCP RSTs points out that implementers need to take care to ensure that the ECN field on a RST does not depend on TCP's state machine. Otherwise the internal information revealed could be of use to potential attackers. This point applies more generally to all control packets, not just RSTs.

#### 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. Richard Scheffenegger provided useful advice gained from implementing ECN++ for FreeBSD.

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, partly by CableLabs and partly by the Comcast Innovation Fund. The views expressed here are solely those of the authors.

## 9. References

### 9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC5961] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", RFC 5961, DOI 10.17487/RFC5961, August 2010, <<https://www.rfc-editor.org/info/rfc5961>>.
- [I-D.ietf-tcpm-accurate-ecn] Briscoe, B., Kuehlewind, M., and R. Scheffenegger, "More Accurate ECN Feedback in TCP", Work in Progress, Internet-Draft, draft-ietf-tcpm-accurate-ecn-15, 12 July 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-accurate-ecn-15>>.

- [RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <<https://www.rfc-editor.org/info/rfc8311>>.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.

## 9.2. Informative References

- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.
- [RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, DOI 10.17487/RFC3540, June 2003, <<https://www.rfc-editor.org/info/rfc3540>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.
- [RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <<https://www.rfc-editor.org/info/rfc4987>>.
- [RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", RFC 5562, DOI 10.17487/RFC5562, June 2009, <<https://www.rfc-editor.org/info/rfc5562>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/info/rfc5681>>.
- [RFC5690] Floyd, S., Arcia, A., Ros, D., and J. Iyengar, "Adding Acknowledgement Congestion Control to TCP", RFC 5690, DOI 10.17487/RFC5690, February 2010, <<https://www.rfc-editor.org/info/rfc5690>>.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <<https://www.rfc-editor.org/info/rfc6298>>.



- [RFC6928] Chu, J., Dukkkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <<https://www.rfc-editor.org/info/rfc6928>>.
- [RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <<https://www.rfc-editor.org/info/rfc7413>>.
- [RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <<https://www.rfc-editor.org/info/rfc7567>>.
- [RFC7661] Fairhurst, G., Sathiaselalan, A., and R. Secchi, "Updating TCP to Support Rate-Limited Traffic", RFC 7661, DOI 10.17487/RFC7661, October 2015, <<https://www.rfc-editor.org/info/rfc7661>>.
- [RFC8257] Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <<https://www.rfc-editor.org/info/rfc8257>>.
- [RFC2140] Touch, J., "TCP Control Block Interdependence", RFC 2140, DOI 10.17487/RFC2140, April 1997, <<https://www.rfc-editor.org/info/rfc2140>>.
- [I-D.ietf-tsvwg-ecn-l4s-id]  
Schepper, K. D. and B. Briscoe, "Explicit Congestion Notification (ECN) Protocol for Very Low Queuing Delay (L4S)", Work in Progress, Internet-Draft, draft-ietf-tsvwg-ecn-l4s-id-23, 24 December 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-ecn-l4s-id-23>>.
- [I-D.ietf-tsvwg-l4s-arch]  
Briscoe, B., Schepper, K. D., Bagnulo, M., and G. White, "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture", Work in Progress, Internet-Draft, draft-ietf-tsvwg-l4s-arch-15, 24 December 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-tsvwg-l4s-arch-15>>.

- [I-D.stewart-tsvwg-sctp-ecn]  
Stewart, R. R., Tuexen, M., and X. Dong, "ECN for Stream Control Transmission Protocol (SCTP)", Work in Progress, Internet-Draft, draft-stewart-tsvwg-sctp-ecn-05, 15 January 2014, <<https://datatracker.ietf.org/doc/html/draft-stewart-tsvwg-sctp-ecn-05>>.
- [RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <<https://www.rfc-editor.org/info/rfc9000>>.
- [judd-nsdi]  
Judd, G.J., "Attaining the promise and avoiding the pitfalls of TCP in the Datacenter", USENIX Symposium on Networked Systems Design and Implementation (NSDI'15) pp.145-157, May 2015, <<https://www.usenix.org/node/188966>>.
- [ecn-pam] Trammell, B., Kühlewind, M., Boppart, D., Learmonth, I., Fairhurst, G., and R. Scheffenegger, "Enabling Internet-Wide Deployment of Explicit Congestion Notification", Int'l Conf. on Passive and Active Network Measurement (PAM'15) pp.193-205, 2015, <[https://link.springer.com/chapter/10.1007/978-3-319-15509-8\\_15](https://link.springer.com/chapter/10.1007/978-3-319-15509-8_15)>.
- [ECN-PLUS] Kuzmanovic, A., "The Power of Explicit Congestion Notification", ACM SIGCOMM 35(4):61--72, 2005, <<http://dl.acm.org/citation.cfm?id=1080100>>.
- [Mandalari18]  
Mandalari, A., Lutu, A., Briscoe, B., Bagnulo, M., and Ö. Alay, "Measuring ECN++: Good News for ++, Bad News for ECN over Mobile", IEEE Communications Magazine, March 2018, <<https://ieeexplore.ieee.org/document/8316790>>.
- [Manzoor17]  
Manzoor, J., Drago, I., and R. Sadre, "How HTTP/2 is changing Web traffic and how to detect it", In Proc: Network Traffic Measurement and Analysis Conference (TMA) 2017 pp.1-9, June 2017, <<https://ieeexplore.ieee.org/document/8002899>>.

[Kuehlewind18]

Kühlewind, M., Walter, M., Learmonth, I., and B. Trammell, "Tracing Internet Path Transparency", In Proc: Network Traffic Measurement and Analysis Conference (TMA) 2018 , June 2018, <[http://tma.ifip.org/2018/wp-content/uploads/sites/3/2018/06/tma2018\\_paper12.pdf](http://tma.ifip.org/2018/wp-content/uploads/sites/3/2018/06/tma2018_paper12.pdf)>.

[strict-ecn]

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

[relax-strict-ecn]

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

[ecn-overload]

Steen, H., "Destruction Testing: Ultra-Low Delay using Dual Queue Coupled Active Queue Management", Masters Thesis, Uni Oslo , May 2017, <<https://www.duo.uio.no/bitstream/handle/10852/57424/thesis-henrste.pdf?sequence=1>>.

[PragueLinux]

Briscoe, B., De Schepper, K., Albisser, O., Misund, J., Tilmans, O., Kühlewind, M., and A.S. Ahmed, "Implementing the 'TCP Prague' Requirements for Low Latency Low Loss Scalable Throughput (L4S)", Proc. Linux Netdev 0x13 , March 2019, <<https://www.netdevconf.org/0x13/session.html?talk-tcp-prague-l4s>>.

[DOCSIS3.1]

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

#### Authors' Addresses

Marcelo Bagnulo  
Universidad Carlos III de Madrid  
Av. Universidad 30  
28911 Leganes Madrid  
Spain

Phone: 34 91 6249500

Email: [marcelo@it.uc3m.es](mailto:marcelo@it.uc3m.es)  
URI: <http://www.it.uc3m.es>

Bob Briscoe  
Independent  
United Kingdom

Email: [ietf@bobbriscoe.net](mailto:ietf@bobbriscoe.net)  
URI: <http://bobbriscoe.net/>

TCPM  
Internet-Draft  
Intended status: Standards Track  
Expires: January 11, 2021

M. Scharf  
Hochschule Esslingen  
V. Murgai

M. Jethanandani  
Kloud Services  
July 10, 2020

YANG Model for Transmission Control Protocol (TCP) Configuration  
draft-scharf-tcpm-yang-tcp-06

Abstract

This document specifies a YANG model for TCP on devices that are configured by network management protocols. The YANG model defines a container for all TCP connections and groupings of some of the parameters that can be imported and used in TCP implementations or by other models that need to configure TCP parameters. The model includes definitions from YANG Groupings for TCP Client and TCP Servers (I-D.ietf-netconf-tcp-client-server). The model is NMDA (RFC 8342) compliant.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 11, 2021.

Copyright Notice

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

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

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

## Table of Contents

1. Introduction . . . . .	2
2. Requirements Language . . . . .	3
2.1. Note to RFC Editor . . . . .	3
3. Model Overview . . . . .	4
3.1. Modeling Scope . . . . .	4
3.2. Model Design . . . . .	5
3.3. Tree Diagram . . . . .	6
4. TCP YANG Model . . . . .	6
5. IANA Considerations . . . . .	13
5.1. The IETF XML Registry . . . . .	13
5.2. The YANG Module Names Registry . . . . .	13
6. Security Considerations . . . . .	13
7. References . . . . .	13
7.1. Normative References . . . . .	13
7.2. Informative References . . . . .	15
Appendix A. Acknowledgements . . . . .	16
Appendix B. Changes compared to previous versions . . . . .	16
Appendix C. Examples . . . . .	17
C.1. Keepalive Configuration . . . . .	17
Authors' Addresses . . . . .	18

## 1. Introduction

The Transmission Control Protocol (TCP) [RFC0793] is used by many applications in the Internet, including control and management protocols. 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 [RFC7950] 1.1 model for configuring TCP on network elements that support YANG data models, and is Network Management Datastore Architecture (NMDA) [RFC8342] compliant. This module defines a container for TCP connection, and includes definitions from YANG Groupings for TCP Clients and TCP Servers [I-D.ietf-netconf-tcp-client-server]. The model has a narrow scope and focuses on fundamental TCP functions and basic statistics. The model can be augmented or updated to address more advanced or implementation-specific TCP features in the future.

Many protocol stacks on Internet hosts use other methods to configure TCP, such as operating system configuration or policies. Many TCP/IP stacks cannot be configured by network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. Moreover, many existing TCP/IP stacks do not use YANG data models. Such TCP implementations often have other means to configure the parameters listed in this document, which are outside the scope of this document.

This specification is orthogonal to the Management Information Base (MIB) for the Transmission Control Protocol (TCP) [RFC4022]. The basic statistics defined in this document follow the model of the TCP MIB. An TCP Extended Statistics MIB [RFC4898] is also available, but this document does not cover such extended statistics. It is possible also to translate a MIB into a YANG model, for instance using Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules [RFC6643]. However, this approach is not used in this document, as such a translated model would not be up-to-date.

There are other existing TCP-related YANG models, which are othogonal to this specification. Examples are:

- o TCP header attributes are modeled in other models, such as YANG Data Model for Network Access Control Lists (ACLs) [RFC8519] and Distributed Denial-of-Service Open Thread Signaling (DOTS) Data Channel Specification [I-D.ietf-dots-data-channel].
- o TCP-related configuration of a NAT (e.g., NAT44, NAT64, Destination NAT, ...) is defined in A YANG Module for Network Address Translation (NAT) and Network Prefix Translation (NPT) [RFC8512] and A YANG Data Model for Dual-Stack Lite (DS-Lite) [RFC8513].

## 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.

### 2.1. Note to RFC Editor

This document uses several placeholder values throughout the document. Please replace them as follows and remove this note before publication.

RFC XXXX, where XXXX is the number assigned to this document at the time of publication.

2020-07-10 with the actual date of the publication of this document.

### 3. Model Overview

#### 3.1. Modeling Scope

TCP is implemented on many different system architectures. As a result, there are many different and often implementation-specific ways to configure parameters of the TCP protocol engine. In addition, in many TCP/IP stacks configuration exists for different scopes:

- o Global configuration: Many TCP implementations have configuration parameters that affect all TCP connections. Typical examples include enabling or disabling optional protocol features.
- o Interface configuration: It can be useful to use different TCP parameters on different interfaces, e.g., different device ports or IP interfaces. In that case, TCP parameters can be part of the interface configuration. Typical examples are the Maximum Segment Size (MSS) or configuration related to hardware offloading.
- o Connection parameters: Many implementations have means to influence the behavior of each TCP connection, e.g., on the programming interface used by applications. A typical example are socket options in the socket API, such as disabling the Nagle algorithm by `TCP_NODELAY`. If an application uses such an interface, it is possible that the configuration of the application or application protocol includes TCP-related parameters. An example is the BGP YANG Model for Service Provider Networks [I-D.ietf-idr-bgp-model].
- o Policies: Setting of TCP parameters can also be part of system policies, templates, or profiles. An example would be the preferences defined in An Abstract Application Layer Interface to Transport Services [I-D.ietf-taps-interface].

As a result, there is no ground truth for setting certain TCP parameters, and traditionally different TCP implementations have used different modeling approaches. For instance, one implementation may define a given configuration parameter globally, while another one uses per-interface settings, and both approaches work well for the corresponding use cases. Also, different systems may use different default values. In addition, TCP can be implemented in different



ways and design choices by the protocol engine often affect configuration options.

Nonetheless, a number of TCP stack parameters require configuration by YANG models. This document therefore defines a minimal YANG model with fundamental parameters directly following from TCP standards.

An important use case is the TCP configuration on network elements such as routers, which often use YANG data models. The model therefore specifies TCP parameters that are important on such TCP stacks. A typical example is the support of TCP-AO [RFC5925]. TCP-AO is increasingly supported on routers to secure routing protocols such as BGP. In that case, TCP-AO configuration is required on routers.

Given an installed base, the model also allows enabling of the legacy TCP MD5 [RFC2385] signature option. As the TCP MD5 signature option is obsoleted by TCP-AO, it is strongly RECOMMENDED to use TCP-AO.

Similar to the TCP MIB [RFC4022], this document also specifies basic statistics and a TCP connection table.

- o Statistics: Counters for the number of active/passive opens, sent and received segments, errors, and possibly other detailed debugging information
- o TCP connection table: Access to status information for all TCP connections

This allows implementations of TCP MIB [RFC4022] to migrate to the YANG model defined in this memo.

### 3.2. Model Design

The YANG model defined in this document includes definitions from the YANG Groupings for TCP Clients and TCP Servers [I-D.ietf-netconf-tcp-client-server]. Similar to that model, this specification defines YANG groupings. This allows reuse of these groupings in different YANG data models. It is intended that these groupings will be used either standalone or for TCP-based protocols as part of a stack of protocol-specific configuration models. An example could be the BGP YANG Model for Service Provider Networks [I-D.ietf-idr-bgp-model].

### 3.3. Tree Diagram

This section provides a abridged tree diagram for the YANG module defined in this document. Annotations used in the diagram are defined in YANG Tree Diagrams [RFC8340].

```
module: ietf-tcp
  +--rw tcp!
    +--rw connections
    |   ...
    +--rw server {server}?
    |   ...
    +--rw client {client}?
    |   ...
    +--ro statistics {statistics}?
    |   ...
```

### 4. TCP YANG Model

<CODE BEGINS> file "ietf-tcp@2020-07-10.yang"

```
module ietf-tcp {
  yang-version "1.1";
  namespace "urn:ietf:params:xml:ns:yang:ietf-tcp";
  prefix "tcp";

  import ietf-yang-types {
    prefix "yang";
    reference
      "RFC 6991: Common YANG Data Types.";
  }
  import ietf-tcp-client {
    prefix "tcpc";
  }
  import ietf-tcp-server {
    prefix "tcps";
  }
  import ietf-tcp-common {
    prefix "tcpcmn";
  }
  import ietf-inet-types {
    prefix "inet";
  }

  organization
    "IETF TCPM Working Group";

  contact
```

```
"WG Web:    <http://tools.ietf.org/wg/tcpm>
WG List:    <tcpm@ietf.org>

Authors: Michael Scharf (michael.scharf at hs-esslingen dot de)
        Vishal Murgai (vmurgai at gmail dot com)
        Mahesh Jethanandani (mjethanandani at gmail dot com)";

description
  "This module focuses on fundamental and standard TCP functions
  that widely implemented. The model can be augmented to address
  more advanced or implementation specific TCP features.";

revision "2020-07-10" {
  description
    "Initial Version";
  reference
    "RFC XXX, TCP Configuration.";
}

// Features
feature server {
  description
    "TCP Server configuration supported.";
}

feature client {
  description
    "TCP Client configuration supported.";
}

feature statistics {
  description
    "This implementation supports statistics reporting.";
}

// TCP-AO Groupings

grouping ao {
  leaf enable-ao {
    type boolean;
    default "false";
    description
      "Enable support of TCP-Authentication Option (TCP-AO).";
  }

  leaf send-id {
    type uint8 {
      range "0..255";
    }
  }
}
```

```
    }
    must "../enable-ao = 'true'";
    description
        "The SendID is inserted as the KeyID of the TCP-AO option
        of outgoing segments.";
    reference
        "RFC 5925: The TCP Authentication Option.";
}

leaf recv-id {
    type uint8 {
        range "0..255";
    }
    must "../enable-ao = 'true'";
    description
        "The RecvID is matched against the TCP-AO KeyID of incoming
        segments.";
    reference
        "RFC 5925: The TCP Authentication Option.";
}

leaf include-tcp-options {
    type boolean;
    must "../enable-ao = 'true'";
    description
        "Include TCP options in HMAC calculation.";
}

leaf accept-ao-mismatch {
    type boolean;
    must "../enable-ao = 'true'";
    description
        "Accept packets with HMAC mismatch.";
}
description
    "Authentication Option (AO) for TCP.";
reference
    "RFC 5925: The TCP Authentication Option.";
}

// MD5 grouping

grouping md5 {
    description
        "Grouping for use in authenticating TCP sessions using MD5.";
    reference
        "RFC 2385: Protection of BGP Sessions via the TCP MD5
        Signature.";
```

```
    leaf enable-md5 {
      type boolean;
      default "false";
      description
        "Enable support of MD5 to authenticate a TCP session.";
    }
  }

  // TCP configuration

  container tcp {
    presence "The container for TCP configuration.";

    description
      "TCP container.";

    container connections {
      list connection {
        key "local-address remote-address local-port remote-port";

        leaf local-address {
          type inet:ip-address;
          description
            "Local address that forms the connection identifier.";
        }

        leaf remote-address {
          type inet:ip-address;
          description
            "Remote address that forms the connection identifier.";
        }

        leaf local-port {
          type inet:port-number;
          description
            "Local TCP port that forms the connection identifier.";
        }

        leaf remote-port {
          type inet:port-number;
          description
            "Remote TCP port that forms the connection identifier.";
        }
      }

      container common {
        uses tcpcmn:tcp-common-grouping;

        choice authentication {
```

```
    case ao {
        uses ao;
        description
            "Use TCP-AO to secure the connection.";
    }

    case md5 {
        uses md5;
        description
            "Use TCP-MD5 to secure the connection.";
    }
    description
        "Choice of how to secure the TCP connection.";
}
description
    "Common definitions of TCP configuration. This includes
    parameters such as how to secure the connection,
    that can be part of either the client or server.";
}
description
    "Connection related parameters.";
}
description
    "A container of all TCP connections.";
}

container server {
    if-feature server;
    uses tcps:tcp-server-grouping;
    description
        "Definitions of TCP server configuration.";
}

container client {
    if-feature client;
    uses tcpc:tcp-client-grouping;
    description
        "Definitions of TCP client configuration.";
}

container statistics {
    if-feature statistics;
    config false;

    leaf active-opens {
        type yang:counter32;
        description
            "The number of times that TCP connections have made a direct
```

```
        transition to the SYN-SENT state from the CLOSED state.";
    }

    leaf passive-opens {
        type yang:counter32;
        description
            "The number of times TCP connections have made a direct
            transition to the SYN-RCVD state from the LISTEN state.";
    }

    leaf attempt-fails {
        type yang:counter32;
        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.";
    }

    leaf establish-resets {
        type yang:counter32;
        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.";
    }

    leaf currently-established {
        type yang:gauge32;
        description
            "The number of TCP connections for which the current state
            is either ESTABLISHED or CLOSE-WAIT.";
    }

    leaf in-segments {
        type yang:counter64;
        description
            "The total number of segments received, including those
            received in error. This count includes segments received
            on currently established connections.";
    }

    leaf out-segments {
        type yang:counter64;
        description
            "The total number of segments sent, including those on
            current connections but excluding those containing only
```

```
        retransmitted octets.";
    }

    leaf retransmitted-segments {
        type yang:counter32;
        description
            "The total number of segments retransmitted; that is, the
             number of TCP segments transmitted containing one or more
             previously transmitted octets.";
    }

    leaf in-errors {
        type yang:counter32;
        description
            "The total number of segments received in error (e.g., bad
             TCP checksums).";
    }

    leaf out-resets {
        type yang:counter32;
        description
            "The number of TCP segments sent containing the RST flag.";
    }

    action reset {
        description
            "Reset statistics action command.";
        input {
            leaf reset-at {
                type yang:date-and-time;
                description
                    "Time when the reset action needs to be
                     executed.";
            }
        }
        output {
            leaf reset-finished-at {
                type yang:date-and-time;
                description
                    "Time when the reset action command completed.";
            }
        }
    }
    description
        "Statistics across all connections.";
}
}
```



<CODE ENDS>

## 5. IANA Considerations

### 5.1. The IETF XML Registry

This document registers two URIs in the "ns" subregistry of the IETF XML Registry [RFC3688]. Following the format in IETF XML Registry [RFC3688], the following registrations are requested:

URI: urn:ietf:params:xml:ns:yang:ietf-tcp  
Registrant Contact: The TCPM WG of the IETF.  
XML: N/A, the requested URI is an XML namespace.

### 5.2. The YANG Module Names Registry

This document registers a YANG modules in the YANG Module Names registry YANG - A Data Modeling Language [RFC6020]. Following the format in YANG - A Data Modeling Language [RFC6020], the following registrations are requested:

name: ietf-tcp  
namespace: urn:ietf:params:xml:ns:yang:ietf-tcp  
prefix: tcp  
reference: RFC XXXX

## 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) described in Using the NETCONF protocol over 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

- [I-D.ietf-netconf-tcp-client-server]  
Watsen, K. and M. Scharf, "YANG Groupings for TCP Clients and TCP Servers", draft-ietf-netconf-tcp-client-server-06 (work in progress), June 2020.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, DOI 10.17487/RFC2385, August 1998, <<https://www.rfc-editor.org/info/rfc2385>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <<https://www.rfc-editor.org/info/rfc5925>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.

- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.
- [RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <<https://www.rfc-editor.org/info/rfc8342>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

## 7.2. Informative References

- [I-D.ietf-dots-data-channel]  
Boucadair, M. and T. Reddy.K, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Data Channel Specification", draft-ietf-dots-data-channel-31 (work in progress), July 2019.
- [I-D.ietf-idr-bgp-model]  
Jethanandani, M., Patel, K., Hares, S., and J. Haas, "BGP YANG Model for Service Provider Networks", draft-ietf-idr-bgp-model-09 (work in progress), June 2020.
- [I-D.ietf-taps-interface]  
Trammell, B., Welzl, M., Enghardt, T., Fairhurst, G., Kuehlewind, M., Perkins, C., Tiesel, P., Wood, C., and T. Pauly, "An Abstract Application Layer Interface to Transport Services", draft-ietf-taps-interface-06 (work in progress), March 2020.
- [RFC4022] Raghunarayan, R., Ed., "Management Information Base for the Transmission Control Protocol (TCP)", RFC 4022, DOI 10.17487/RFC4022, March 2005, <<https://www.rfc-editor.org/info/rfc4022>>.

- [RFC4898] Mathis, M., Heffner, J., and R. Raghunarayan, "TCP Extended Statistics MIB", RFC 4898, DOI 10.17487/RFC4898, May 2007, <<https://www.rfc-editor.org/info/rfc4898>>.
- [RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules", RFC 6643, DOI 10.17487/RFC6643, July 2012, <<https://www.rfc-editor.org/info/rfc6643>>.
- [RFC8512] Boucadair, M., Ed., Sivakumar, S., Jacquenet, C., Vinapamula, S., and Q. Wu, "A YANG Module for Network Address Translation (NAT) and Network Prefix Translation (NPT)", RFC 8512, DOI 10.17487/RFC8512, January 2019, <<https://www.rfc-editor.org/info/rfc8512>>.
- [RFC8513] Boucadair, M., Jacquenet, C., and S. Sivakumar, "A YANG Data Model for Dual-Stack Lite (DS-Lite)", RFC 8513, DOI 10.17487/RFC8513, January 2019, <<https://www.rfc-editor.org/info/rfc8513>>.
- [RFC8519] Jethanandani, M., Agarwal, S., Huang, L., and D. Blair, "YANG Data Model for Network Access Control Lists (ACLs)", RFC 8519, DOI 10.17487/RFC8519, March 2019, <<https://www.rfc-editor.org/info/rfc8519>>.

#### Appendix A. Acknowledgements

Michael Scharf was supported by the StandICT.eu project, which is funded by the European Commission under the Horizon 2020 Programme.

The following persons have contributed to this document by reviews:  
Mohamed Boucadair

#### Appendix B. Changes compared to previous versions

Changes compared to draft-scharf-tcpm-yang-tcp-04

- o Removed congestion control
- o Removed global stack parameters

Changes compared to draft-scharf-tcpm-yang-tcp-03

- o Updated TCP-AO grouping
- o Added congestion control

Changes compared to draft-scharf-tcpm-yang-tcp-02

- o Initial proposal of a YANG model including base configuration parameters, TCP-AO configuration, and a connection list
- o Editorial bugfixes and outdated references reported by Mohamed Boucadair
- o Additional co-author Mahesh Jethanandani

Changes compared to draft-scharf-tcpm-yang-tcp-01

- o Alignment with [I-D.ietf-netconf-tcp-client-server]
- o Removing backward-compatibility to the TCP MIB
- o Additional co-author Vishal Murgai

Changes compared to draft-scharf-tcpm-yang-tcp-00

- o Editorial improvements

## Appendix C. Examples

### C.1. Keepalive Configuration

This particular example demonstrates how both a particular connection can be configured for keepalives.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  This example shows how TCP keepalive can be configured for
  a given connection. An idle connection is dropped after
  idle-time + (max-probes * probe-interval).
-->
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <tcp
    xmlns="urn:ietf:params:xml:ns:yang:ietf-tcp">
    <connections>
      <connection>
        <local-address>192.168.1.1</local-address>
        <remote-address>192.168.1.2</remote-address>
        <local-port>1025</local-port>
        <remote-port>80</remote-port>
        <common>
          <keepalives>
            <idle-time>5</idle-time>
            <max-probes>5</max-probes>
            <probe-interval>10</probe-interval>
          </keepalives>
        </common>
      </connection>
    </connections>
  <!--
    It is not clear why a server and client configuration is
    needed here even as they under a feature statement and therefore
    are required only if the feature is declared. Adding it so
    that yanglint allows this validation to run.
  -->
  <server>
    <local-address>192.168.1.1</local-address>
  </server>
  <client>
    <remote-address>192.168.1.2</remote-address>
  </client>
</tcp>
</config>
```

#### Authors' Addresses

Michael Scharf  
Hochschule Esslingen - University of Applied Sciences  
Flandernstr. 101  
Esslingen 73732  
Germany

Email: michael.scharf@hs-esslingen.de

Vishal Murgai

Email: vmurgai@gmail.com

Mahesh Jethanandani  
Kloud Services

Email: mjethanandani@gmail.com