

TCP Maintenance & Minor Extensions (tcpm)  
Internet-Draft  
Updates: 3168, 3449 (if approved)  
Intended status: Standards Track  
Expires: January 13, 2022

B. Briscoe  
Independent  
M. Kuehlewind  
Ericsson  
R. Scheffenegger  
NetApp  
July 12, 2021

More Accurate ECN Feedback in TCP  
draft-ietf-tcpm-accurate-ecn-15

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 specifies 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.

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 13, 2022.

## Copyright Notice

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

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

## Table of Contents

1. Introduction . . . . .	3
1.1. Document Roadmap . . . . .	5
1.2. Goals . . . . .	5
1.3. Terminology . . . . .	5
1.4. Recap of Existing ECN feedback in IP/TCP . . . . .	6
2. AccECN Protocol Overview and Rationale . . . . .	7
2.1. Capability Negotiation . . . . .	8
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 . . . . .	18
3.2.2. The ACE Field . . . . .	19
3.2.3. The AccECN Option . . . . .	27
3.3. AccECN Compliance Requirements for TCP Proxies, Offload Engines and other Middleboxes . . . . .	36
3.3.1. Requirements for TCP Proxies . . . . .	36
3.3.2. Requirements for Transparent Middleboxes and TCP Normalizers . . . . .	36
3.3.3. Requirements for TCP ACK Filtering . . . . .	36
3.3.4. Requirements for TCP Segmentation Offload . . . . .	37

4. Updates to RFC 3168 . . . . .	38
5. Interaction with TCP Variants . . . . .	39
5.1. Compatibility with SYN Cookies . . . . .	40
5.2. Compatibility with TCP Experiments and Common TCP Options . . . . .	40
5.3. Compatibility with Feedback Integrity Mechanisms . . . . .	41
6. Protocol Properties . . . . .	42
7. IANA Considerations . . . . .	44
8. Security Considerations . . . . .	45
9. Acknowledgements . . . . .	46
10. Comments Solicited . . . . .	47
11. References . . . . .	47
11.1. Normative References . . . . .	47
11.2. Informative References . . . . .	47
Appendix A. Example Algorithms . . . . .	50
A.1. Example Algorithm to Encode/Decode the AccECN Option . . . . .	50
A.2. Example Algorithm for Safety Against Long Sequences of ACK Loss . . . . .	51
A.2.1. Safety Algorithm without the AccECN Option . . . . .	51
A.2.2. Safety Algorithm with the AccECN Option . . . . .	53
A.3. Example Algorithm to Estimate Marked Bytes from Marked Packets . . . . .	55
A.4. Example Algorithm to Count Not-ECT Bytes . . . . .	56
Appendix B. Rationale for Usage of TCP Header Flags . . . . .	56
B.1. Three TCP Header Flags in the SYN-SYN/ACK Handshake . . . . .	56
B.2. Four Codepoints in the SYN/ACK . . . . .	57
B.3. Space for Future Evolution . . . . .	58
Authors' Addresses . . . . .	59

## 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 E	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 should be able to infer how much to increment its own counters, even if the protocol field has wrapped.

The 3-bit ACE field can wrap fairly frequently. Therefore, even if it appears to have incremented by one (say), the field might have actually cycled completely then incremented by one. The Data Receiver is 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 recommended in order to protect against the receiver using attacks similar to 'ACK-Division' to artificially inflate the congestion window, which is why [RFC5681] now recommends that TCP counts acknowledged bytes not packets.

## 2.5. Generic (Dumb) Reflector

The ACE field provides 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 [Mandalari18]. 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 and send Not-ECT for the rest of the connection. Previously, 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 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 TCP flags on the SYN/ACK to one of the 4 values shown in the top block of Table 2 to confirm that it supports AccECN. The TCP server MUST NOT set one of these 4 combination of flags on the SYN/ACK unless the preceding SYN requested support for AccECN as above.

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

Once a TCP client (A) has sent the above SYN to declare that it supports AccECN, and once it has received the above SYN/ACK segment that confirms that the TCP server supports AccECN, the TCP client MUST set both its half connections into AccECN mode.

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 should 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 may 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 should be arranged to expire so that the initiator will infrequently attempt to check whether the problem has been resolved.

The fall-back procedure if the TCP server receives no ACK to acknowledge a SYN/ACK that tried to negotiate AccECN is specified in Section 3.2.3.2.

### 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].
- \* 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 The Data Receiver 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 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 = 1 and r.ceb = r.elb = 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 = 1` and `s.ceb = s.elb = 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.

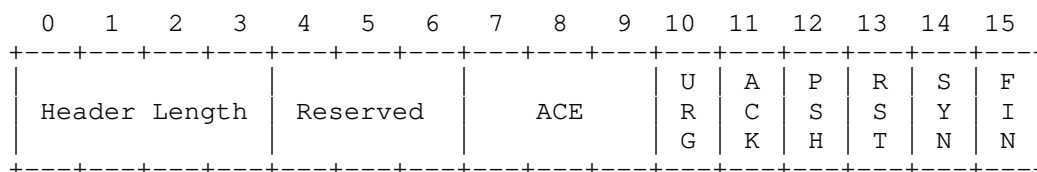


Figure 3: Definition of the ACE field within bytes 13 and 14 of the TCP Header (when AccECN has been negotiated and SYN=0).

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 3.2.2.3).

{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 on the SYN/ACK using the encoding in Table 2. 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 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), the Data Sender disables sending ECN-capable packets for the remainder of the half-connection by setting the IP/ECN field in all subsequent packets to Not-ECT.

Usually, the server checks the ACK of the SYN/ACK from the client, while the client checks the first data segment from the server. However, 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.

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.

#### 3.2.2.4. Testing for Mangling of the IP/ECN Field

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

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

The ACK of the SYN/ACK is not reliably delivered (nonetheless, the count of CE marks is still eventually delivered reliably). If this ACK does not arrive, the server can continue to send ECN-capable packets without having tested for mangling of the IP/ECN field on the SYN/ACK.

Invalid transitions of the IP/ECN field are defined in [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 [Mandalaril8].

Once a Data Sender has entered AccECN mode it SHOULD check whether all feedback received for the first three or four rounds indicated that every packet it sent was CE-marked. If so, for the remainder of the connection, the Data Sender SHOULD NOT send ECN-capable packets, but it MUST continue to feed back any ECN markings on arriving packets.

The above fall-back behaviours are necessary in case mangling of the IP/ECN field is asymmetric, which is currently common over some mobile networks [Mandalaril8]. 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.

#### 3.2.2.5. 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 new data to acknowledge, 'n' SHOULD be 2. If there is no new data

to acknowledge, 'n' SHOULD be 3 and MUST be no less than 3. In either case, 'n' MUST be no greater than 6.

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 new 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'.

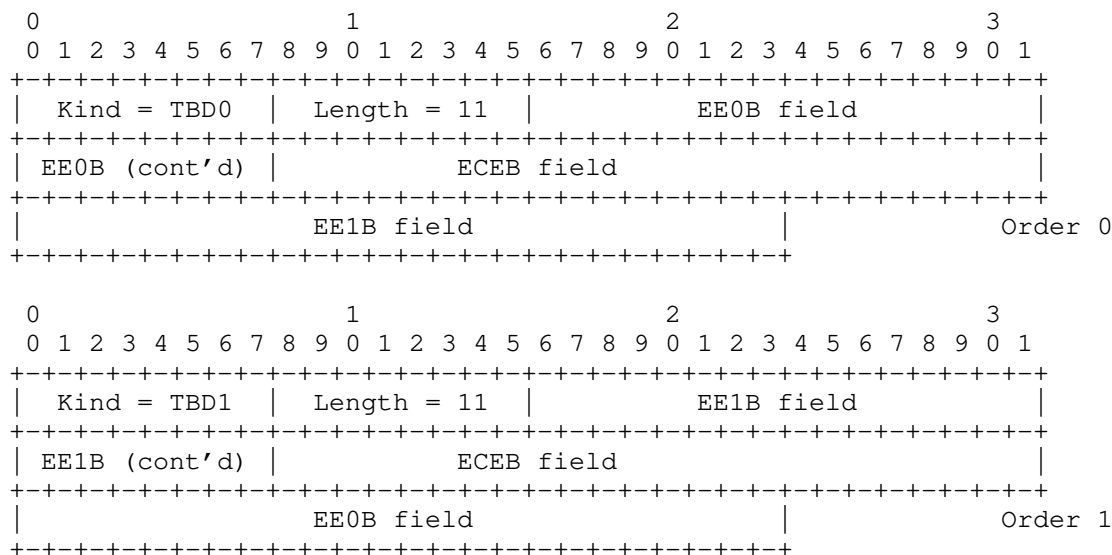


Figure 4: The AccECN TCP Option

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)

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. If sending of the AccECN Option is implemented, the fallbacks 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.elb) 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.elb (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. [I-D.ietf-tcpm-2140bis] 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 should resend the SYN/ACK, again confirming its support for AccECN, but this time without the AccECN Option. This approach rules out any interference by middleboxes that may drop packets with unknown options, even though it is more likely that the SYN/ACK would have been lost due to congestion. The TCP server MAY try to send another packet with the AccECN Option at a later point during the connection but should monitor if that packet got lost as well, in which case it SHOULD disable the sending of the AccECN Option for this half-connection.

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

#### 3.2.3.2.3. 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 3.2.2.3

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

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

While a host is in the mode that ignores the AccECN Option it MUST adopt the conservative interpretation of the ACE field discussed in Section 3.2.2.5.

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 supplements but does not replace the ACE field. An endpoint using AccECN feedback MUST always consider the information provided in the ACE field whether or not the AccECN Option is also available.

If the AccECN option is present, the s.cep counter might increase while the s.ceb counter does not (e.g. due to a CE-marked control packet). The sender's response to such a situation is out of scope, and needs to be dealt with in a specification that uses ECN-capable control packets. Theoretically, this situation could also occur if a middlebox mangled the AccECN Option but not the ACE field. However, the Data Sender has to assume that the integrity of the AccECN Option is sound, based on the above test of the well-known initial values and optionally other integrity tests (Section 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), this invalid protocol transition has to be due to some form of feedback mangling. So, the Data Sender MUST disable sending ECN-capable packets for the remainder of the half-connection by setting the IP/ECN field in all subsequent packets to Not-ECT.

#### 3.2.3.3. 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 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 that acknowledges new data. 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, `EE0B` 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) and itself also implements ECN marking will not need to filter ACKs from connections that use AccECN feedback. Therefore, such a node SHOULD detect connections that are using AccECN feedback and it

SHOULD refrain from filtering the ACKs of such connections (if it coalesced ACKs it would not be AccECN-compliant, but the requirement is stated as a "SHOULD" in order to allow leeway for pre-existing ACK filtering functions to be brought into line).

A node that implements ACK filtering and does not itself implement ECN marking does not need to treat AccECN connections any differently from other TCP connections. Nonetheless, it is RECOMMENDED that such nodes implement ECN marking and comply with the requirements of the previous paragraph. This should be a better way than ACK filtering to improve the performance of AccECN TCP connections.

The rationale for these requirements is that AccECN feedback provides sufficient information to a Data Receiver for it to be able to monitor ECN marking of the ACKs it has sent, so that it can thin the ACK stream itself. This could eventually mean that ACK filtering in the network gives no performance advantage. Then TCP will be able to maintain its own control over ACK coalescing. This will also allow the TCP Data Sender to use the timing of ACK arrivals to more reliably infer further information about the path congestion level.

Note that the specification of AccECN in TCP does not presume to rely on any of the above ACK filtering behaviour in the network, because it has to be robust against pre-existing network nodes that still filter AccECN ACKs, and robust against ACK loss during overload.

Section 5.2.1 of [RFC3449] gives best current practice on ACK filtering (aka. thinning or coalescing). 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 is said that "ECN remain areas of ongoing research". This section updates that advice 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 should 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 must have requested AccECN support on the SYN
- o it (the server) must have confirmed that it supported AccECN

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

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

### 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 should therefore 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

[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

[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). {ToDo: is this still true?}

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-2140bis] Touch, J., Welzl, M., and S. Islam, "TCP Control Block Interdependence", draft-ietf-tcpm-2140bis-11 (work in progress), April 2021.

- [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-07 (work in progress), February 2021.
- [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-08 (work in progress), November 2020.
- [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>>.
- [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>>.

## 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  $10 - ((10-2) \% 8) = 10$ ).

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

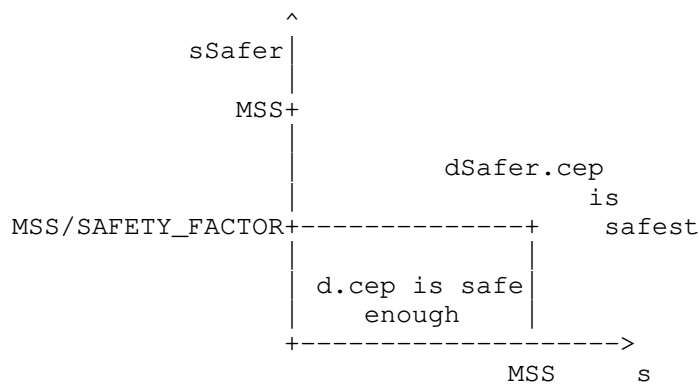
- see Appendix A.1). Specifically, it could use the following algorithm:

```

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=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, dSafer.cep=10 and d.ceb=1460, then s=730 and sSafer=146.  
Therefore d.cep is safe enough, because the average size of 10 data segments is unlikely to have been as small as MSS/10.

- 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.elb. Note that, because r.e0b is initialized to 1 and the other two counters are initialized to 0, the initial sum will be 1, which matches the initial offset of the TCP sequence number on completion of the 3WHS.

For this approach to be precise, it has to be assumed that spurious (unnecessary) retransmissions do not lead to double counting. This assumption is currently correct, given that RFC 3168 requires that the Data Sender marks retransmitted segments as Not-ECT. However, the converse is not true; necessary 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 should be no need to keep track of the details of retransmissions.

### Appendix B. Rationale for Usage of TCP Header Flags

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

AccECN uses a rather unorthodox 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 3.2.2.3 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)

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.

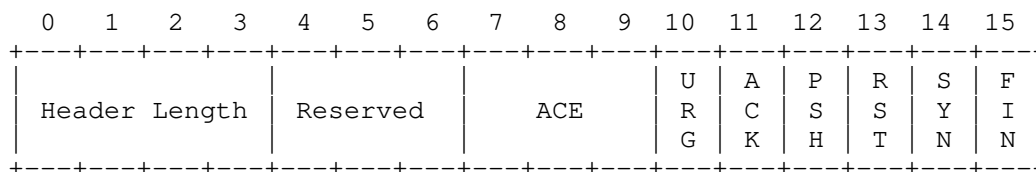


Figure 3: Definition of the ACE field within bytes 13 and 14 of the TCP Header (when AccECN has been negotiated and SYN=0).

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'.

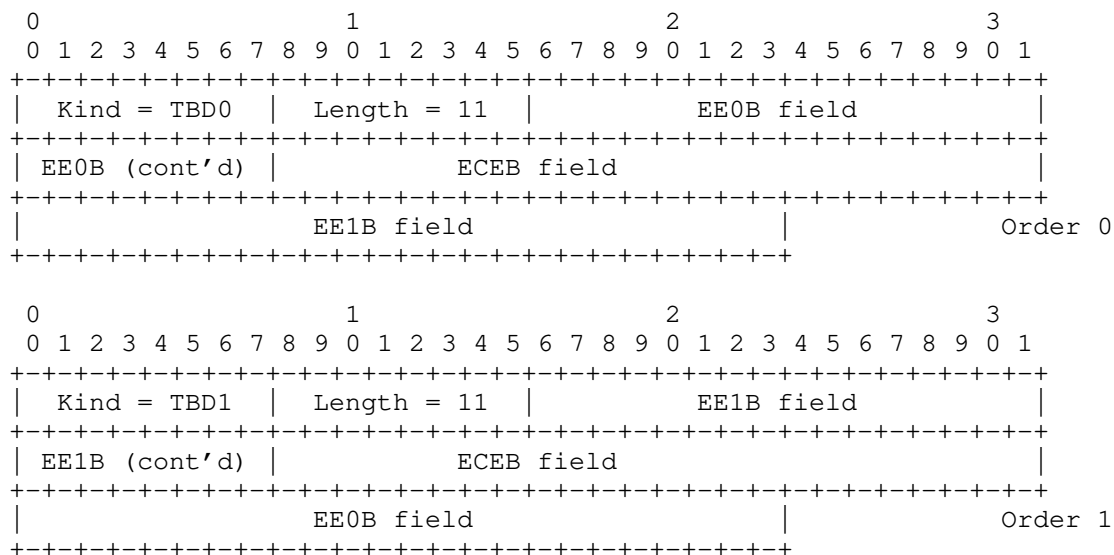


Figure 4: The AccECN TCP Option

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-1](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.

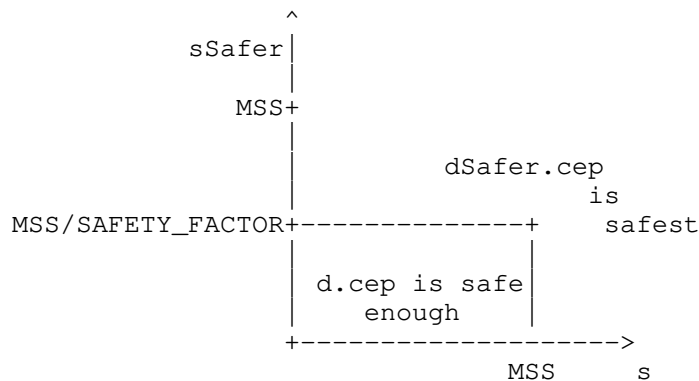
#### A.2.2. Safety Algorithm with the AccECN Option

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  
Intended status: Standards Track  
Expires: January 13, 2022

P. Balasubramanian  
Y. Huang  
M. Olson  
Microsoft  
July 12, 2021

HyStart++: Modified Slow Start for TCP  
draft-ietf-tcpm-hystartplusplus-02

Abstract

This document describes HyStart++, a simple modification to the slow start phase of TCP congestion control algorithms. Traditional slow start can cause overshooting of the ideal send rate and cause large packet loss within a round-trip time which results in poor performance. HyStart++ is composed of the delay increase variant of HyStart to prevent overshooting of the ideal sending rate, while also mitigating poor performance which can result from false positives.

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 13, 2022.

Copyright Notice

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

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

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Terminology . . . . .	3
3. Definitions . . . . .	3
4. HyStart++ Algorithm . . . . .	3
4.1. Summary . . . . .	3
4.2. Algorithm Details . . . . .	4
4.3. Tuning constants . . . . .	6
5. Deployments and Performance Evaluations . . . . .	7
6. Security Considerations . . . . .	7
7. IANA Considerations . . . . .	7
8. References . . . . .	7
8.1. Normative References . . . . .	7
8.2. Informative References . . . . .	8
Authors' Addresses . . . . .	8

## 1. Introduction

[RFC5681] describes the slow start congestion control algorithm for TCP. The slow start algorithm is used when the congestion window (cwnd) is less than the slow start threshold (ssthresh). During slow start, in absence of packet loss signals, TCP sender increases cwnd exponentially to probe the network capacity. Such a fast growth can lead to overshooting the ideal sending rate and cause significant packet loss. This is counter-productive for the TCP flow itself, and also impacts the rest of the traffic sharing the bottleneck link. TCP has several mechanisms for loss recovery, but they are only effective for moderate loss. When these techniques are unable to recover lost packets, a last-resort retransmission timeout (RTO) is used to trigger packet recovery. In most operating systems, the minimum RTO is set to a large value (200 msec or 300 msec) to prevent spurious timeouts. This results in a long idle time which drastically impairs flow completion times.

HyStart++ adds delay increase as a signal to exit slow start before any packet loss occurs. This is one of two algorithms specified in [HyStart]. After the HyStart delay algorithm finds an exit point, a Conservative Slow Start (CSS) phase is used to determine if the slow start exit was spurious. This provides protection against jitter and prevents performance problems that result from early slow start exit due to false positives. HyStart++ reduces packet loss and retransmissions, and improves goodput in lab measurements as well as real world deployments.

## 2. Terminology

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

## 3. Definitions

We repeat here some definition from [RFC5681] to aid the reader.

**SENDER MAXIMUM SEGMENT SIZE (SMSS):** The SMSS is the size of the largest segment that the sender can transmit. This value can be based on the maximum transmission unit of the network, the path MTU discovery [RFC1191, RFC4821] algorithm, RMSS (see next item), or other factors. The size does not include the TCP/IP headers and options.

**RECEIVER MAXIMUM SEGMENT SIZE (RMSS):** The RMSS is the size of the largest segment the receiver is willing to accept. This is the value specified in the MSS option sent by the receiver during connection startup. Or, if the MSS option is not used, it is 536 bytes [RFC1122]. The size does not include the TCP/IP headers and options.

**RECEIVER WINDOW (rwnd):** The most recently advertised receiver window.

**CONGESTION WINDOW (cwnd):** A TCP state variable that limits the amount of data a TCP can send. At any given time, a TCP MUST NOT send data with a sequence number higher than the sum of the highest acknowledged sequence number and the minimum of cwnd and rwnd.

## 4. HyStart++ Algorithm

### 4.1. Summary

[HyStart] specifies two algorithms (a "Delay Increase" algorithm and an "Inter-Packet Arrival" algorithm) to be run in parallel to detect that the sending rate has reached capacity. In practice, the Inter-Packet Arrival algorithm does not perform well and is not able to detect congestion early, primarily due to ACK compression. The idea of the Delay Increase algorithm is to look for RTT spikes, which suggest that the bottleneck buffer is filling up.

In HyStart++, a TCP sender uses traditional slow start and then uses the "Delay Increase" algorithm to trigger an exit from slow start. But instead of using a congestion avoidance algorithm, the sender uses a Conservative Slow Start (CSS) algorithm to determine if the exit was spurious. If the exit is determined to be spurious, slow

start is resumed. If the exit is determined to be not spurious, the sender enters congestion avoidance.

#### 4.2. Algorithm Details

We assume that Appropriate Byte Counting (as described in [RFC3465]) is in use and  $L$  is the cwnd increase limit. The choice of value of  $L$  is up to the implementation.

A round is chosen to be approximately the Round-Trip Time (RTT). Round can be approximated using sequence numbers as follows:

Define windowEnd as a sequence number initialize to SND.UNA

When windowEnd is ACKed, the current round ends and windowEnd is set to SND.NXT

At the start of each round during normal slow start and CSS:

lastRoundMinRTT = currentRoundMinRTT

currentRoundMinRTT = infinity

rttSampleCount = 0

For each arriving ACK in slow start, where  $N$  is the number of previously unacknowledged bytes acknowledged in the arriving ACK:

Update the cwnd

$cwnd = cwnd + \min(N, L * SMSS)$

Keep track of minimum observed RTT

$currentRoundMinRTT = \min(currentRoundMinRTT, currRTT)$

where currRTT is the RTT sampled from the incoming ACK

rttSampleCount += 1

For rounds where cwnd is at or higher than LOW\_CWND and  $N\_RTT\_SAMPLE$  RTT samples have been obtained, check if delay increase triggers slow start exit

if ( $cwnd \geq (LOW\_CWND * SMSS)$  AND  $rttSampleCount \geq N\_RTT\_SAMPLE$ )

```
RttThresh = clamp(MIN_RTT_THRESH, lastRoundMinRTT / 8,  
MAX_RTT_THRESH)  
  
if (currentRoundMinRTT >= (lastRoundMinRTT + RttThresh))  
  
    cssBaselineMinRtt = currentRoundMinRTT  
  
    exit slow start and enter CSS
```

CSS lasts CSS\_ROUNDS rounds. If the transition into CSS happens in the middle of a round, that partial round counts towards the limit.

For each arriving ACK in CSS, where N is the number of previously unacknowledged bytes acknowledged in the arriving ACK:

Update the cwnd

```
cwnd = cwnd + (min (N, L * SMSS) / CSS_GROWTH_DIVISOR)
```

Keep track of minimum observed RTT

```
currentRoundMinRTT = min(currentRoundMinRTT, currRTT)
```

where currRTT is the sampled RTT from the incoming ACK

```
rttSampleCount += 1
```

For CSS rounds where N\_RTT\_SAMPLE RTT samples have been obtained, check if current round's minRTT drops below baseline indicating that HyStart exit was spurious.

```
if (currentRoundMinRTT < cssBaselineMinRtt)  
  
    cssBaselineMinRtt = infinity  
  
    resume slow start including HyStart++
```

If CSS\_ROUNDS rounds are complete, enter congestion avoidance.

```
ssthresh = cwnd
```

If congestion is observed anytime during slow start or CSS, enter congestion avoidance.

```
ssthresh = cwnd
```

#### 4.3. Tuning constants

It is RECOMMENDED that a HyStart++ implementation use the following constants:

LOW\_CWND = 16

MIN\_RTT\_THRESH = 4 msec

MAX\_RTT\_THRESH = 16 msec

N\_RTT\_SAMPLE = 8

CSS\_GROWTH\_DIVISOR = 4

CSS\_ROUNDS = 5

These constants have been determined with lab measurements and real world deployments. An implementation MAY tune them for different network characteristics.

Using smaller values of LOW\_CWND will cause the algorithm to kick in before the last round RTT can be measured, particularly if the implementation uses an initial cwnd of 10 MSS. Higher values will delay the detection of delay increase and reduce the ability of HyStart++ to prevent overshoot problems.

The delay increase sensitivity is determined by MIN\_RTT\_THRESH and MAX\_RTT\_THRESH. Smaller values of MIN\_RTT\_THRESH may cause spurious exits from slow start. Larger values of MAX\_RTT\_THRESH may result in slow start not exiting until loss is encountered for connections on large RTT paths.

A TCP implementation is required to take at least one RTT sample each round. Using lower values of N\_RTT\_SAMPLE will lower the accuracy of the measured RTT for the round; higher values will improve accuracy at the cost of more processing.

The minimum value of CSS\_GROWTH\_DIVISOR SHOULD be at least 2. Otherwise the cwnd growth could again become too aggressive and cause ideal send rate overshoot. Values larger than 4 will cause the algorithm to be less aggressive and maybe less performant.

Smaller values of CSS\_ROUNDS may miss detecting jitter and larger values may limit performance.

An implementation SHOULD use HyStart++ only for the initial slow start (when ssthresh is at its initial value of arbitrarily high per

[RFC5681]) and fall back to using traditional slow start for the remainder of the connection lifetime. This is acceptable because subsequent slow starts will use the discovered ssthresh value to exit slow start and avoid the overshoot problem. An implementation MAY use HyStart++ to grow the restart window ([RFC5681]) after a long idle period.

## 5. Deployments and Performance Evaluations

As of the time of writing, HyStart++ has been default enabled for all TCP connections in Windows for two years. The original Hystart has been default-enabled for all TCP connections in Linux TCP for a decade.

In lab measurements with Windows TCP, HyStart++ shows both goodput improvements as well as reductions in packet loss and retransmissions. For example across a variety of tests on a 100 Mbps link with a bottleneck buffer size of bandwidth-delay product, HyStart++ reduces bytes retransmitted by 50% and retransmission timeouts by 36%.

In an A/B test across a large Windows device population, out of 52 billion TCP connections, 0.7% of connections move from 1 RTO to 0 RTOs and another 0.7% connections move from 2 RTOs to 1 RTO with HyStart++. This test did not focus on send heavy connections and the impact on send heavy connections is likely much higher. We plan to conduct more such production experiments to gather more data in the future.

## 6. Security Considerations

HyStart++ enhances slow start and inherits the general security considerations discussed in [RFC5681].

## 7. IANA Considerations

This document has no actions for IANA.

## 8. References

### 8.1. 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>>.

- [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>>.

## 8.2. Informative References

- [HyStart] Ha, S. and I. Ree, "Hybrid Slow Start for High-Bandwidth and Long-Distance Networks", DOI 10.1145/1851182.1851192, International Workshop on Protocols for Fast Long-Distance Networks, 2008, <<https://pdfs.semanticscholar.org/25e9/ef3f03315782c7f1cbcd31b587857adae7d1.pdf>>.

## Authors' Addresses

Praveen Balasubramanian  
Microsoft  
One Microsoft Way  
Redmond, WA 98052  
USA

Phone: +1 425 538 2782  
Email: [pravb@microsoft.com](mailto:pravb@microsoft.com)

Yi Huang  
Microsoft

Phone: +1 425 703 0447  
Email: [huanyih@microsoft.com](mailto:huanyih@microsoft.com)

Matt Olson  
Microsoft

Phone: +1 425 538 8598  
Email: [maolson@microsoft.com](mailto:maolson@microsoft.com)

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: 27 July 2022

P. Balasubramanian  
Y. Huang  
M. Olson  
Microsoft  
23 January 2022

HyStart++: Modified Slow Start for TCP  
draft-ietf-tcpm-hystartplusplus-04

Abstract

This document describes HyStart++, a simple modification to the slow start phase of TCP congestion control algorithms. Traditional slow start can overshoot the ideal send rate in many cases, causing high packet loss and poor performance. HyStart++ uses a delay increase heuristic to find an exit point before possible overshoot. It also adds a mitigation to prevent jitter from causing premature slow start exit.

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 27 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 . . . . .	2
2. Terminology . . . . .	3
3. Definitions . . . . .	3
4. HyStart++ Algorithm . . . . .	3
4.1. Summary . . . . .	3
4.2. Algorithm Details . . . . .	4
4.3. Tuning constants . . . . .	6
5. Deployments and Performance Evaluations . . . . .	7
6. Security Considerations . . . . .	7
7. IANA Considerations . . . . .	7
8. References . . . . .	7
8.1. Normative References . . . . .	7
8.2. Informative References . . . . .	8
Authors' Addresses . . . . .	8

## 1. Introduction

[RFC5681] describes the slow start congestion control algorithm for TCP. The slow start algorithm is used when the congestion window (cwnd) is less than the slow start threshold (ssthresh). During slow start, in absence of packet loss signals, TCP increases cwnd exponentially to probe the network capacity. This fast growth can overshoot the ideal sending rate and cause significant packet loss which cannot always be recovered efficiently.

HyStart++ uses delay increase as a signal to exit slow start before potential packet loss occurs as a result of overshoot. This is one of two algorithms specified in [HyStart]. After the slow start exit, a novel Conservative Slow Start (CSS) phase is used to determine whether the slow start exit was premature and to resume slow start. This mitigation improves performance in presence of jitter. HyStart++ reduces packet loss and retransmissions, and improves goodput in lab measurements and real world deployments.

## 2. Terminology

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

## 3. Definitions

We repeat here some definition from [RFC5681] to aid the reader.

**SENDER MAXIMUM SEGMENT SIZE (SMSS):** The SMSS is the size of the largest segment that the sender can transmit. This value can be based on the maximum transmission unit of the network, the path MTU discovery [RFC1191, RFC4821] algorithm, RMSS (see next item), or other factors. The size does not include the TCP/IP headers and options.

**RECEIVER MAXIMUM SEGMENT SIZE (RMSS):** The RMSS is the size of the largest segment the receiver is willing to accept. This is the value specified in the MSS option sent by the receiver during connection startup. Or, if the MSS option is not used, it is 536 bytes [RFC1122]. The size does not include the TCP/IP headers and options.

**RECEIVER WINDOW (rwnd):** The most recently advertised receiver window.

**CONGESTION WINDOW (cwnd):** A TCP state variable that limits the amount of data a TCP can send. At any given time, a TCP MUST NOT send data with a sequence number higher than the sum of the highest acknowledged sequence number and the minimum of cwnd and rwnd.

## 4. HyStart++ Algorithm

### 4.1. Summary

[HyStart] specifies two algorithms (a "Delay Increase" algorithm and an "Inter-Packet Arrival" algorithm) to be run in parallel to detect that the sending rate has reached capacity. In practice, the Inter-Packet Arrival algorithm does not perform well and is not able to detect congestion early, primarily due to ACK compression. The idea of the Delay Increase algorithm is to look for spikes in RTT (round-trip time), which suggest that the bottleneck buffer is filling up.

In HyStart++, a TCP sender uses traditional slow start and then uses the "Delay Increase" algorithm to trigger an exit from slow start. But instead of going straight from slow start to congestion avoidance, the sender spends a number of RTTs in a Conservative Slow Start (CSS) phase to determine whether the exit from slow start was premature. During CSS, the congestion window is grown exponentially

like in regular slow start, but with a smaller exponential base, resulting in less aggressive growth. If the RTT reduces during CSS, it's concluded that the RTT spike was not related to congestion caused by the connection sending at a rate greater than the ideal send rate, and the connection resumes slow start. If the RTT inflation persists throughout CSS, the connection enters congestion avoidance.

#### 4.2. Algorithm Details

For the pseudocode, we assume that Appropriate Byte Counting (as described in [RFC3465]) is in use and L is the cwnd increase limit as discussed in RFC 3465.

lastRoundMinRTT and currentRoundMinRTT are initialized to infinity at the initialization time

Hystart++ measures rounds using sequence numbers, as follows:

Define windowEnd as a sequence number initialized to SND.UNA

When windowEnd is ACKed, the current round ends and windowEnd is set to SND.NXT

At the start of each round during standard slow start ([RFC5681]) and CSS:

lastRoundMinRTT = currentRoundMinRTT

currentRoundMinRTT = infinity

rttSampleCount = 0

For each arriving ACK in slow start, where N is the number of previously unacknowledged bytes acknowledged in the arriving ACK:

Update the cwnd

-  $cwnd = cwnd + \min(N, L * SMSS)$

Keep track of minimum observed RTT

-  $currentRoundMinRTT = \min(currentRoundMinRTT, currRTT)$

- where currRTT is the RTT sampled from the latest incoming ACK

-  $rttSampleCount += 1$

For rounds where `N_RTT_SAMPLE` RTT samples have been obtained and `currentRoundMinRTT` and `lastRoundMinRTT` are valid, check if delay increase triggers slow start exit

- if (`rttSampleCount`  $\geq$  `N_RTT_SAMPLE` AND `currentRoundMinRTT`  $\neq$  infinity AND `lastRoundMinRTT`  $\neq$  infinity)
  - o `RttThresh` = `clamp(MIN_RTT_THRESH, lastRoundMinRTT / 8, MAX_RTT_THRESH)`
  - o if (`currentRoundMinRTT`  $\geq$  (`lastRoundMinRTT` + `RttThresh`))
    - + `cssBaselineMinRtt` = `currentRoundMinRTT`
    - + exit slow start and enter CSS

CSS lasts at most `CSS_ROUNDS` rounds. If the transition into CSS happens in the middle of a round, that partial round counts towards the limit.

For each arriving ACK in CSS, where `N` is the number of previously unacknowledged bytes acknowledged in the arriving ACK:

Update the `cwnd`

- `cwnd` = `cwnd` + (`min` (`N`, `L * SMSS`) / `CSS_GROWTH_DIVISOR`)

Keep track of minimum observed RTT

- `currentRoundMinRTT` = `min`(`currentRoundMinRTT`, `currRTT`)
- where `currRTT` is the sampled RTT from the incoming ACK
- `rttSampleCount` += 1

For CSS rounds where `N_RTT_SAMPLE` RTT samples have been obtained, check if current round's `minRTT` drops below baseline indicating that HyStart exit was spurious.

- if (`currentRoundMinRTT` < `cssBaselineMinRtt`)
  - o `cssBaselineMinRtt` = infinity
  - o resume slow start including HyStart++

If `CSS_ROUNDS` rounds are complete, enter congestion avoidance.

\* `ssthresh` = `cwnd`

If loss or ECN-marking is observed anytime during standard slow start or CSS, enter congestion avoidance.

\* ssthresh = cwnd

#### 4.3. Tuning constants

It is RECOMMENDED that a HyStart++ implementation use the following constants:

\* MIN\_RTT\_THRESH = 4 msec

\* MAX\_RTT\_THRESH = 16 msec

\* N\_RTT\_SAMPLE = 8

\* CSS\_GROWTH\_DIVISOR = 4

\* CSS\_ROUNDS = 5

These constants have been determined with lab measurements and real world deployments. An implementation MAY tune them for different network characteristics.

The delay increase sensitivity is determined by MIN\_RTT\_THRESH and MAX\_RTT\_THRESH. Smaller values of MIN\_RTT\_THRESH may cause spurious exits from slow start. Larger values of MAX\_RTT\_THRESH may result in slow start not exiting until loss is encountered for connections on large RTT paths.

A TCP implementation is required to take at least one RTT sample each round. Using lower values of N\_RTT\_SAMPLE will lower the accuracy of the measured RTT for the round; higher values will improve accuracy at the cost of more processing.

The minimum value of CSS\_GROWTH\_DIVISOR MUST be at least 2. A value of 1 results in the same aggressive behavior as regular slow start. Values larger than 4 will cause the algorithm to be less aggressive and maybe less performant.

Smaller values of CSS\_ROUNDS may miss detecting jitter and larger values may limit performance.

An implementation SHOULD use HyStart++ only for the initial slow start (when ssthresh is at its initial value of arbitrarily high per [RFC5681]) and fall back to using traditional slow start for the remainder of the connection lifetime. This is acceptable because subsequent slow starts will use the discovered ssthresh value to exit

slow start and avoid the overshoot problem. An implementation MAY use HyStart++ to grow the restart window ([RFC5681]) after a long idle period.

## 5. Deployments and Performance Evaluations

As of the time of writing, HyStart++ as described in draft versions 01 through 04 was default enabled for all TCP connections in the Windows operating system for over three years. The original Hystart has been default-enabled for all TCP connections in the Linux operating system using the default congestion control module CUBIC ([RFC8312]) for a decade.

In lab measurements with Windows TCP, HyStart++ shows both goodput improvements as well as reductions in packet loss and retransmissions. For example across a variety of tests on a 100 Mbps link with a bottleneck buffer size of bandwidth-delay product, HyStart++ reduces bytes retransmitted by 50% and retransmission timeouts by 36%.

In an A/B test for HyStart++ draft 01 across a large Windows device population, out of 52 billion TCP connections, 0.7% of connections move from 1 RTO to 0 RTOs and another 0.7% connections move from 2 RTOs to 1 RTO with HyStart++. This test did not focus on send heavy connections and the impact on send heavy connections is likely much higher. We plan to conduct more such production experiments to gather more data in the future.

## 6. Security Considerations

HyStart++ enhances slow start and inherits the general security considerations discussed in [RFC5681].

## 7. IANA Considerations

This document has no actions for IANA.

## 8. References

### 8.1. 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>>.

- [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>>.

## 8.2. Informative References

- [HyStart] Ha, S. and I. Ree, "Hybrid Slow Start for High-Bandwidth and Long-Distance Networks", DOI 10.1145/1851182.1851192, International Workshop on Protocols for Fast Long-Distance Networks, 2008, <<https://pdfs.semanticscholar.org/25e9/ef3f03315782c7f1cbcd31b587857adae7d1.pdf>>.
- [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>>.

## Authors' Addresses

Praveen Balasubramanian  
Microsoft  
One Microsoft Way  
Redmond, WA 98052  
United States of America

Phone: +1 425 538 2782  
Email: [pravb@microsoft.com](mailto:pravb@microsoft.com)

Yi Huang  
Microsoft

Phone: +1 425 703 0447  
Email: [huanyi@microsoft.com](mailto:huanyi@microsoft.com)

Matt Olson  
Microsoft

Phone: +1 425 538 8598  
Email: [maolson@microsoft.com](mailto:maolson@microsoft.com)

TCPM  
Internet-Draft  
Obsoletes: 8312 (if approved)  
Intended status: Standards Track  
Expires: 6 November 2021

L. Xu  
UNL  
S. Ha  
Colorado  
I. Rhee  
Bowery  
V. Goel  
Apple Inc.  
L. Eggert, Ed.  
NetApp  
5 May 2021

CUBIC for Fast and Long-Distance Networks  
draft-ietf-tcpm-rfc8312bis-02

Abstract

CUBIC is a standard TCP congestion control algorithm that uses a cubic function instead of the linear window increase function on the sender side to improve scalability and stability over fast and long-distance networks. CUBIC has been adopted as the default TCP congestion control algorithm by the Linux, Windows, and Apple stacks.

This document updates the specification of CUBIC to include algorithmic improvements based on these implementations and recent academic work. Based on the extensive deployment experience with CUBIC, it also moves the specification to the Standards Track, obsoleting [RFC8312].

Note to Readers

Discussion of this draft takes place on the TCPM working group mailing list (<mailto:tcpm@ietf.org>), which is archived at <https://mailarchive.ietf.org/arch/browse/tcpm/>.

Working Group information can be found at <https://datatracker.ietf.org/wg/tcpm/>; source code and issues list for this draft can be found at <https://github.com/NTAP/rfc8312bis>.

Note to the RFC Editor

xml2rfc currently renders "<em></em>" in the XML by surrounding the corresponding text with underscores. This is highly distracting; please manually remove the underscores when doing the final edits to the text version of this document.

(There is an issue open against xml2rfc to stop doing this in the future: <https://trac.tools.ietf.org/tools/xml2rfc/trac/ticket/596>)

#### 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 6 November 2021.

#### Copyright Notice

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

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

#### Table of Contents

1. Introduction . . . . .	3
2. Conventions . . . . .	4
3. Design Principles of CUBIC . . . . .	4
3.1. Principle 1 for the CUBIC Increase Function . . . . .	5
3.2. Principle 2 for AIMD Friendliness . . . . .	6
3.3. Principle 3 for RTT Fairness . . . . .	6
3.4. Principle 4 for the CUBIC Decrease Factor . . . . .	7
4. CUBIC Congestion Control . . . . .	7
4.1. Definitions . . . . .	7
4.1.1. Constants of Interest . . . . .	8
4.1.2. Variables of Interest . . . . .	8
4.2. Window Increase Function . . . . .	9

4.3.	AIMD-Friendly Region . . . . .	11
4.4.	Concave Region . . . . .	12
4.5.	Convex Region . . . . .	12
4.6.	Multiplicative Decrease . . . . .	13
4.7.	Fast Convergence . . . . .	13
4.8.	Timeout . . . . .	14
4.9.	Spurious Congestion Events . . . . .	15
4.10.	Slow Start . . . . .	16
5.	Discussion . . . . .	16
5.1.	Fairness to AIMD TCP . . . . .	17
5.2.	Using Spare Capacity . . . . .	19
5.3.	Difficult Environments . . . . .	20
5.4.	Investigating a Range of Environments . . . . .	20
5.5.	Protection against Congestion Collapse . . . . .	21
5.6.	Fairness within the Alternative Congestion Control Algorithm . . . . .	21
5.7.	Performance with Misbehaving Nodes and Outside Attackers . . . . .	21
5.8.	Behavior for Application-Limited Flows . . . . .	21
5.9.	Responses to Sudden or Transient Events . . . . .	21
5.10.	Incremental Deployment . . . . .	21
6.	Security Considerations . . . . .	21
7.	IANA Considerations . . . . .	22
8.	References . . . . .	22
8.1.	Normative References . . . . .	22
8.2.	Informative References . . . . .	23
Appendix A.	Acknowledgments . . . . .	25
Appendix B.	Evolution of CUBIC . . . . .	25
B.1.	Since draft-ietf-tcpm-rfc8312bis-01 . . . . .	25
B.2.	Since draft-ietf-tcpm-rfc8312bis-00 . . . . .	25
B.3.	Since draft-eggert-tcpm-rfc8312bis-03 . . . . .	25
B.4.	Since draft-eggert-tcpm-rfc8312bis-02 . . . . .	25
B.5.	Since draft-eggert-tcpm-rfc8312bis-01 . . . . .	26
B.6.	Since draft-eggert-tcpm-rfc8312bis-00 . . . . .	26
B.7.	Since RFC8312 . . . . .	27
B.8.	Since the Original Paper . . . . .	27
Authors'	Addresses . . . . .	27

## 1. Introduction

CUBIC has been adopted as the default TCP congestion control algorithm in the Linux, Windows, and Apple stacks, and has been used and deployed globally. Extensive, decade-long deployment experience in vastly different Internet scenarios has convincingly demonstrated that CUBIC is safe for deployment on the global Internet and delivers substantial benefits over traditional AIMD congestion control. It is therefore to be regarded as the current standard for TCP congestion control.

The design of CUBIC was motivated by the well-documented problem traditional TCP has with low utilization over fast and long-distance networks [K03][RFC3649]. This problem arises from a slow increase of the congestion window following a congestion event in a network with a large bandwidth-delay product (BDP). [HKLRX06] indicates that this problem is frequently observed even in the range of congestion window sizes over several hundreds of packets. This problem is equally applicable to all Reno-style TCP standards and their variants, including TCP-Reno [RFC5681], TCP-NewReno [RFC6582][RFC6675], SCTP [RFC4960], and TFRC [RFC5348], which use the same linear increase function for window growth. We refer to all Reno-style TCP standards and their variants collectively as "AIMD TCP" below because they use the Additive Increase and Multiplicative Decrease algorithm (AIMD).

CUBIC, originally proposed in [HRX08], is a modification to the congestion control algorithm of traditional AIMD TCP to remedy this problem. This document describes the most recent specification of CUBIC. Specifically, CUBIC uses a cubic function instead of the linear window increase function of AIMD TCP to improve scalability and stability under fast and long-distance networks.

Binary Increase Congestion Control (BIC-TCP) [XHR04], a predecessor of CUBIC, was selected as the default TCP congestion control algorithm by Linux in the year 2005 and had been used for several years by the Internet community at large.

CUBIC uses a similar window increase function as BIC-TCP and is designed to be less aggressive and fairer to AIMD TCP in bandwidth usage than BIC-TCP while maintaining the strengths of BIC-TCP such as stability, window scalability, and round-trip time (RTT) fairness.

In the following sections, we first briefly explain the design principles of CUBIC, then provide the exact specification of CUBIC, and finally discuss the safety features of CUBIC following the guidelines specified in [RFC5033].

## 2. Conventions

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

## 3. Design Principles of CUBIC

CUBIC is designed according to the following design principles:

Principle 1: For better network utilization and stability, CUBIC uses both the concave and convex profiles of a cubic function to increase the congestion window size, instead of using just a convex function.

Principle 2: To be AIMD-friendly, CUBIC is designed to behave like AIMD TCP in networks with short RTTs and small bandwidth where AIMD TCP performs well.

Principle 3: For RTT-fairness, CUBIC is designed to achieve linear bandwidth sharing among flows with different RTTs.

Principle 4: CUBIC appropriately sets its multiplicative window decrease factor in order to balance between the scalability and convergence speed.

### 3.1. Principle 1 for the CUBIC Increase Function

For better network utilization and stability, CUBIC [HRX08] uses a cubic window increase function in terms of the elapsed time from the last congestion event. While most alternative congestion control algorithms to AIMD TCP increase the congestion window using convex functions, CUBIC uses both the concave and convex profiles of a cubic function for window growth.

After a window reduction in response to a congestion event is detected by duplicate ACKs or Explicit Congestion Notification-Echo (ECN-Echo, ECE) ACKs [RFC3168], CUBIC remembers the congestion window size where it received the congestion event and performs a multiplicative decrease of the congestion window. When CUBIC enters into congestion avoidance, it starts to increase the congestion window using the concave profile of the cubic function. The cubic function is set to have its plateau at the remembered congestion window size, so that the concave window increase continues until then. After that, the cubic function turns into a convex profile and the convex window increase begins.

This style of window adjustment (concave and then convex) improves the algorithm stability while maintaining high network utilization [CEHRX07]. This is because the window size remains almost constant, forming a plateau around the remembered congestion window size of the last congestion event, where network utilization is deemed highest. Under steady state, most window size samples of CUBIC are close to that remembered congestion window size, thus promoting high network utilization and stability.

Note that congestion control algorithms that only use convex functions to increase the congestion window size have their maximum increments around the remembered congestion window size of the last congestion event, and thus introduce many packet bursts around the saturation point of the network, likely causing frequent global loss synchronizations.

### 3.2. Principle 2 for AIMD Friendliness

CUBIC promotes per-flow fairness to AIMD TCP. Note that AIMD TCP performs well over paths with short RTTs and small bandwidths (or small BDPs). There is only a scalability problem in networks with long RTTs and large bandwidths (or large BDPs).

A congestion control algorithm designed to be friendly to AIMD TCP on a per-flow basis must increase its congestion window less aggressively in small BDP networks than in large BDP networks.

The aggressiveness of CUBIC mainly depends on the maximum window size before a window reduction, which is smaller in small-BDP networks than in large-BDP networks. Thus, CUBIC increases its congestion window less aggressively in small-BDP networks than in large-BDP networks.

Furthermore, in cases when the cubic function of CUBIC would increase the congestion window less aggressively than AIMD TCP, CUBIC simply follows the window size of AIMD TCP to ensure that CUBIC achieves at least the same throughput as AIMD TCP in small-BDP networks. We call this region where CUBIC behaves like AIMD TCP the "AIMD-friendly region".

### 3.3. Principle 3 for RTT Fairness

Two CUBIC flows with different RTTs have a throughput ratio that is linearly proportional to the inverse of their RTT ratio, where the throughput of a flow is approximately the size of its congestion window divided by its RTT.

Specifically, CUBIC maintains a window increase rate independent of RTTs outside the AIMD-friendly region, and thus flows with different RTTs have similar congestion window sizes under steady state when they operate outside the AIMD-friendly region.

This notion of a linear throughput ratio is similar to that of AIMD TCP under high statistical multiplexing where packet loss is independent of individual flow rates. However, under low statistical multiplexing, the throughput ratio of AIMD TCP flows with different RTTs is quadratically proportional to the inverse of their RTT ratio [XHR04].

CUBIC always ensures a linear throughput ratio independent of the amount of statistical multiplexing. This is an improvement over AIMD TCP. While there is no consensus on particular throughput ratios for different RTT flows, we believe that over wired Internet paths, use of a linear throughput ratio seems more reasonable than equal throughputs (i.e., the same throughput for flows with different RTTs) or a higher-order throughput ratio (e.g., a quadratical throughput ratio of AIMD TCP under low statistical multiplexing environments).

#### 3.4. Principle 4 for the CUBIC Decrease Factor

To balance between scalability and convergence speed, CUBIC sets the multiplicative window decrease factor to 0.7, whereas AIMD TCP uses 0.5.

While this improves the scalability of CUBIC, a side effect of this decision is slower convergence, especially under low statistical multiplexing. This design choice is following the observation that HighSpeed TCP (HSTCP) [RFC3649] and other approaches (e.g., [GV02]) made: the current Internet becomes more asynchronous with less frequent loss synchronizations under high statistical multiplexing.

In such environments, even strict Multiplicative-Increase Multiplicative-Decrease (MIMD) can converge. CUBIC flows with the same RTT always converge to the same throughput independent of statistical multiplexing, thus achieving intra-algorithm fairness. We also find that in environments with sufficient statistical multiplexing, the convergence speed of CUBIC is reasonable.

#### 4. CUBIC Congestion Control

In this section, we discuss how the congestion window is updated during the different stages of the CUBIC congestion controller.

##### 4.1. Definitions

The unit of all window sizes in this document is segments of the maximum segment size (MSS), and the unit of all times is seconds.

#### 4.1.1. Constants of Interest

`__cubic__`: CUBIC multiplication decrease factor as described in Section 4.6.

`__aimd__`: CUBIC additive increase factor used in AIMD-friendly region as described in Section 4.3.

`_C_`: constant that determines the aggressiveness of CUBIC in competing with other congestion control algorithms in high BDP networks. Please see Section 5 for more explanation on how it is set. The unit for `_C_` is

segment  
-----  
          3  
second

#### 4.1.2. Variables of Interest

This section defines the variables required to implement CUBIC:

`_RTT_`: Smoothed round-trip time in seconds, calculated as described in [RFC6298].

`_cwnd_`: Current congestion window in segments.

`_ssthresh_`: Current slow start threshold in segments.

`_W_max_`: Size of `_cwnd_` in segments just before `_cwnd_` was reduced in the last congestion event when fast convergence is disabled. However, if fast convergence is enabled, the size may be further reduced based on the current saturation point.

`_K_`: The time period in seconds it takes to increase the congestion window size at the beginning of the current congestion avoidance stage to `_W_max_`.

`_current_time_`: Current time of the system in seconds.

`_epoch_start_`: The time in seconds at which the current congestion avoidance stage started.

`_cwnd_start_`: The `_cwnd_` at the beginning of the current congestion avoidance stage, i.e., at time `_epoch_start_`.

`W_cubic(_t_)`: The congestion window in segments at time `_t_` in seconds based on the cubic increase function, as described in Section 4.2.

`_target_`: Target value of congestion window in segments after the next RTT, that is, `W_cubic(_t_ + _RTT_)`, as described in Section 4.2.

`_W_est_`: An estimate for the congestion window in segments in the AIMD-friendly region, that is, an estimate for the congestion window of AIMD TCP.

`_segments_acked_`: Number of segments acked when an ACK is received.

#### 4.2. Window Increase Function

CUBIC maintains the acknowledgment (ACK) clocking of AIMD TCP by increasing the congestion window only at the reception of an ACK. It does not make any changes to the TCP Fast Recovery and Fast Retransmit algorithms [RFC6582][RFC6675].

During congestion avoidance after a congestion event where a packet loss is detected by duplicate ACKs or by receiving packets carrying ECE flags [RFC3168], CUBIC changes the window increase function of AIMD TCP.

CUBIC uses the following window increase function:

$$W_{\text{cubic}}(t) = C * (t - K)^3 + W_{\text{max}}$$

Figure 1

where `_t_` is the elapsed time in seconds from the beginning of the current congestion avoidance stage, that is,

$$t = \text{current\_time} - \text{epoch\_start}$$

and where `_epoch_start_` is the time at which the current congestion avoidance stage starts. `_K_` is the time period that the above function takes to increase the congestion window size at the beginning of the current congestion avoidance stage to `_W_max_` if there are no further congestion events and is calculated using the following equation:

$$K = \frac{3}{\frac{W_{\max} - \text{cwnd}_{\text{start}}}{C}}$$

Figure 2

where `_cwnd_start_` is the congestion window at the beginning of the current congestion avoidance stage. For example, right after a congestion event, `_cwnd_start_` is equal to the new `cwnd` calculated as described in Section 4.6.

Upon receiving an ACK during congestion avoidance, CUBIC computes the `_target_` congestion window size after the next `_RTT_` using Figure 1 as follows, where `_RTT_` is the smoothed round-trip time. The lower and upper bounds below ensure that CUBIC's congestion window increase rate is non-decreasing and is less than the increase rate of slow start.

$$\text{target} = \begin{cases} \text{cwnd} & \text{if } W_{\text{cubic}}(t + \text{RTT}) < \text{cwnd} \\ 1.5 * \text{cwnd} & \text{if } W_{\text{cubic}}(t + \text{RTT}) > 1.5 * \text{cwnd} \\ W_{\text{cubic}}(t + \text{RTT}) & \text{otherwise} \end{cases}$$

Depending on the value of the current congestion window size `_cwnd_`, CUBIC runs in three different regions:

1. The AIMD-friendly region, which ensures that CUBIC achieves at least the same throughput as AIMD TCP.
2. The concave region, if CUBIC is not in the AIMD-friendly region and `_cwnd_` is less than `_W_max_`.
3. The convex region, if CUBIC is not in the AIMD-friendly region and `_cwnd_` is greater than `_W_max_`.

Below, we describe the exact actions taken by CUBIC in each region.

#### 4.3. AIMD-Friendly Region

AIMD TCP performs well in certain types of networks, for example, under short RTTs and small bandwidths (or small BDPs). In these networks, CUBIC remains in the AIMD-friendly region to achieve at least the same throughput as AIMD TCP.

The AIMD-friendly region is designed according to the analysis in [FHP00], which studies the performance of an AIMD algorithm with an additive factor of `__aimd_` (segments per `_RTT_`) and a multiplicative factor of `__aimd_`, denoted by `AIMD(__aimd_, __aimd_)`. Specifically, the average congestion window size of `AIMD(__aimd_, __aimd_)` can be calculated using Figure 3. The analysis shows that `AIMD(__aimd_, __aimd_)` with

$$\text{aimd} = 3 * \frac{1 - \text{cubic}}{1 + \text{cubic}}$$

achieves the same average window size as AIMD TCP that uses `AIMD(1, 0.5)`.

$$\text{AVG\_AIMD}(\text{aimd}, \text{aimd}) = \left| \frac{\frac{1}{\text{aimd}} * (1 + \frac{1}{\text{aimd}})}{\frac{1}{2 * (1 - \frac{1}{\text{aimd}})} * p} \right|$$

Figure 3

Based on the above analysis, CUBIC uses Figure 4 to estimate the window size `_W_est_` of `AIMD(__aimd_, __aimd_)` with

$$\begin{aligned} & \text{aimd} = 3 * \frac{1 - \text{cubic}}{1 + \text{cubic}} \\ & = \end{aligned}$$

which achieves the same average window size as AIMD TCP. When receiving an ACK in congestion avoidance (where `_cwnd_` could be greater than or less than `_W_max_`), CUBIC checks whether `W_cubic(_t_)` is less than `_W_est_`. If so, CUBIC is in the AIMD-friendly region and `_cwnd_` SHOULD be set to `_W_est_` at each reception of an ACK.

`_W_est_` is set equal to `_cwnd_start_` at the start of the congestion avoidance stage. After that, on every ACK, `_W_est_` is updated using Figure 4.

$$W_{est} = W_{est} + \text{aimd} * \frac{\text{segments\_acked}}{\text{cwnd}}$$

Figure 4

Note that once `_W_est_` reaches `_W_max_`, that is, `_W_est_ >= _W_max_`, `_aimd_` SHOULD be set to 1 to achieve the same congestion window increment as AIMD TCP, which uses AIMD(1, 0.5).

#### 4.4. Concave Region

When receiving an ACK in congestion avoidance, if CUBIC is not in the AIMD-friendly region and `_cwnd_` is less than `_W_max_`, then CUBIC is in the concave region. In this region, `_cwnd_` MUST be incremented by

$$\frac{\text{target} - \text{cwnd}}{\text{cwnd}}$$

for each received ACK, where `_target_` is calculated as described in Section 4.2.

#### 4.5. Convex Region

When receiving an ACK in congestion avoidance, if CUBIC is not in the AIMD-friendly region and `_cwnd_` is larger than or equal to `_W_max_`, then CUBIC is in the convex region.

The convex region indicates that the network conditions might have changed since the last congestion event, possibly implying more available bandwidth after some flow departures. Since the Internet is highly asynchronous, some amount of perturbation is always possible without causing a major change in available bandwidth.

In this region, CUBIC is very careful. The convex profile ensures that the window increases very slowly at the beginning and gradually increases its increase rate. We also call this region the "maximum probing phase", since CUBIC is searching for a new `_W_max_`. In this region, `_cwnd_` MUST be incremented by

$$\frac{\text{target} - \text{cwnd}}{\text{cwnd}}$$

for each received ACK, where `_target_` is calculated as described in Section 4.2.

#### 4.6. Multiplicative Decrease

When a packet loss is detected by duplicate ACKs or by receiving packets carrying ECE flags, CUBIC updates `_W_max_` and reduces `_cwnd_` and `_sssthresh_` immediately as described below. An implementation MAY set a smaller `_sssthresh_` than suggested below to accommodate rate-limited applications as described in [RFC7661]. For both packet loss and congestion detection through ECN, the sender MAY employ a Fast Recovery algorithm to gradually adjust the congestion window to its new reduced `_sssthresh_` value. The parameter `__cubic_` SHOULD be set to 0.7.

```
sssthresh = cwnd *          // new slow-start threshold
              cubic

sssthresh = max(sssthresh, 2) // threshold is at least 2 MSS

                                // window reduction
cwnd = sssthresh
```

A side effect of setting `__cubic_` to a value bigger than 0.5 is slower convergence. We believe that while a more adaptive setting of `__cubic_` could result in faster convergence, it will make the analysis of CUBIC much harder.

#### 4.7. Fast Convergence

To improve convergence speed, CUBIC uses a heuristic. When a new flow joins the network, existing flows need to give up some of their bandwidth to allow the new flow some room for growth, if the existing flows have been using all the network bandwidth. To speed up this bandwidth release by existing flows, the following "Fast Convergence" mechanism SHOULD be implemented.

With Fast Convergence, when a congestion event occurs, we update `_W_max_` as follows, before the window reduction as described in Section 4.6.

$$W_{\max} = \begin{cases} \text{cwnd} * \frac{1 + \text{cubic} \text{ if } \text{cwnd} < W_{\max} \text{ and fast convergence is enabled,}}{2} \\ \text{further reduce } W_{\max} \\ \text{otherwise, remember cwnd before reduction} \end{cases}$$

\cwnd

At a congestion event, if the current `_cwnd_` is less than `_W_max_`, this indicates that the saturation point experienced by this flow is getting reduced because of a change in available bandwidth. Then we allow this flow to release more bandwidth by reducing `_W_max_` further. This action effectively lengthens the time for this flow to increase its congestion window, because the reduced `_W_max_` forces the flow to plateau earlier. This allows more time for the new flow to catch up to its congestion window size.

Fast Convergence is designed for network environments with multiple CUBIC flows. In network environments with only a single CUBIC flow and without any other traffic, Fast Convergence SHOULD be disabled.

#### 4.8. Timeout

In case of a timeout, CUBIC follows AIMD TCP to reduce `_cwnd_` [RFC5681], but sets `_ssthresh_` using `__cubic_` (same as in Section 4.6) in a way that is different from AIMD TCP [RFC5681].

During the first congestion avoidance stage after a timeout, CUBIC increases its congestion window size using Figure 1, where `_t_` is the elapsed time since the beginning of the current congestion avoidance, `_K_` is set to 0, and `_W_max_` is set to the congestion window size at the beginning of the current congestion avoidance stage. In addition, for the AIMD-friendly region, `_W_est_` SHOULD be set to the congestion window size at the beginning of the current congestion avoidance.

#### 4.9. Spurious Congestion Events

In cases where CUBIC reduces its congestion window in response to having detected packet loss via duplicate ACKs or timeouts, there is a possibility that the missing ACK would arrive after the congestion window reduction and a corresponding packet retransmission. For example, packet reordering could trigger this behavior. A high degree of packet reordering could cause multiple congestion window reduction events, where spurious losses are incorrectly interpreted as congestion signals, thus degrading CUBIC's performance significantly.

When there is a congestion event, a CUBIC implementation SHOULD save the current value of the following variables before the congestion window reduction.

```
prior_cwnd = cwnd

prior_ssthresh = ssthresh

prior_W      = W
    max      max

prior_K = K

prior_epoch      = epoch
    start        start

prior_W_{est} = W
                est
```

CUBIC MAY implement an algorithm to detect spurious retransmissions, such as DSACK [RFC3708], Forward RTO-Recovery [RFC5682] or Eifel [RFC3522]. Once a spurious congestion event is detected, CUBIC SHOULD restore the original values of above mentioned variables as follows if the current `_cwnd_` is lower than `_prior_cwnd_`. Restoring the original values ensures that CUBIC's performance is similar to what it would be without spurious losses.

```

        cwnd = prior_cwnd
        ssthresh = prior_ssthresh

        W      = prior_W
        max      max

        K = prior_K

        epoch      = prior_epoch
        start      start

        W      = prior_W
        est      est
    \
    >if cwnd < prior_cwnd
    /

```

In rare cases, when the detection happens long after a spurious loss event and the current `_cwnd_` is already higher than `_prior_cwnd_`, CUBIC SHOULD continue to use the current and the most recent values of these variables.

#### 4.10. Slow Start

CUBIC MUST employ a slow-start algorithm, when `_cwnd_` is no more than `_ssthresh_`. Among the slow-start algorithms, CUBIC MAY choose the AIMD TCP slow start [RFC5681] in general networks, or the limited slow start [RFC3742] or hybrid slow start [HR08] for fast and long-distance networks.

When CUBIC uses hybrid slow start [HR08], it may exit the first slow start without incurring any packet loss and thus `_W_max_` is undefined. In this special case, CUBIC switches to congestion avoidance and increases its congestion window size using Figure 1, where `_t_` is the elapsed time since the beginning of the current congestion avoidance, `_K_` is set to 0, and `_W_max_` is set to the congestion window size at the beginning of the current congestion avoidance stage.

#### 5. Discussion

In this section, we further discuss the safety features of CUBIC following the guidelines specified in [RFC5033].

With a deterministic loss model where the number of packets between two successive packet losses is always `_1/p_`, CUBIC always operates with the concave window profile, which greatly simplifies the performance analysis of CUBIC. The average window size of CUBIC can be obtained by the following function:

$$\text{AVG\_W}_{\text{cubic}} = \sqrt[4]{\frac{\sqrt[3]{C * (3 + \sqrt[3]{\text{cubic}})}}{4 * (1 - \sqrt[3]{\text{cubic}})}} * \frac{\sqrt[3]{\frac{4}{\text{RTT}}}}{\sqrt[3]{p}}$$

Figure 5

With `__cubic_` set to 0.7, the above formula reduces to:

$$\text{AVG\_W}_{\text{cubic}} = \sqrt[4]{\frac{\sqrt[3]{C * 3.7}}{1.2}} * \frac{\sqrt[3]{\frac{4}{\text{RTT}}}}{\sqrt[3]{p}}$$

Figure 6

We will determine the value of `_C_` in the following subsection using Figure 6.

### 5.1. Fairness to AIMD TCP

In environments where AIMD TCP is able to make reasonable use of the available bandwidth, CUBIC does not significantly change this state.

AIMD TCP performs well in the following two types of networks:

1. networks with a small bandwidth-delay product (BDP)
2. networks with a short RTTs, but not necessarily a small BDP

CUBIC is designed to behave very similarly to AIMD TCP in the above two types of networks. The following two tables show the average window sizes of AIMD TCP, HSTCP, and CUBIC. The average window sizes of AIMD TCP and HSTCP are from [RFC3649]. The average window size of CUBIC is calculated using Figure 6 and the CUBIC AIMD-friendly region for three different values of `_C_`.

Loss Rate P	AIMD	HSTCP	CUBIC (C=0.04)	CUBIC (C=0.4)	CUBIC (C=4)
1.0e-02	12	12	12	12	12
1.0e-03	38	38	38	38	59
1.0e-04	120	263	120	187	333
1.0e-05	379	1795	593	1054	1874
1.0e-06	1200	12280	3332	5926	10538
1.0e-07	3795	83981	18740	33325	59261
1.0e-08	12000	574356	105383	187400	333250

Table 1: AIMD TCP, HSTCP, and CUBIC with RTT = 0.1 seconds

Table 1 describes the response function of AIMD TCP, HSTCP, and CUBIC in networks with `_RTT_` = 0.1 seconds. The average window size is in MSS-sized segments.

Loss Rate P	AIMD	HSTCP	CUBIC (C=0.04)	CUBIC (C=0.4)	CUBIC (C=4)
1.0e-02	12	12	12	12	12
1.0e-03	38	38	38	38	38
1.0e-04	120	263	120	120	120
1.0e-05	379	1795	379	379	379
1.0e-06	1200	12280	1200	1200	1874
1.0e-07	3795	83981	3795	5926	10538
1.0e-08	12000	574356	18740	33325	59261

Table 2: AIMD TCP, HSTCP, and CUBIC with RTT = 0.01 seconds

Table 2 describes the response function of AIMD TCP, HSTCP, and CUBIC in networks with `_RTT_` = 0.01 seconds. The average window size is in MSS-sized segments.

Both tables show that CUBIC with any of these three `_C_` values is more friendly to AIMD TCP than HSTCP, especially in networks with a short `_RTT_` where AIMD TCP performs reasonably well. For example, in a network with `_RTT_` = 0.01 seconds and  $p=10^{-6}$ , AIMD TCP has an average window of 1200 packets. If the packet size is 1500 bytes, then AIMD TCP can achieve an average rate of 1.44 Gbps. In this case, CUBIC with `_C_`=0.04 or `_C_`=0.4 achieves exactly the same rate as AIMD TCP, whereas HSTCP is about ten times more aggressive than AIMD TCP.

We can see that `_C_` determines the aggressiveness of CUBIC in competing with other congestion control algorithms for bandwidth. CUBIC is more friendly to AIMD TCP, if the value of `_C_` is lower. However, we do not recommend setting `_C_` to a very low value like 0.04, since CUBIC with a low `_C_` cannot efficiently use the bandwidth in fast and long-distance networks. Based on these observations and extensive deployment experience, we find `_C_`=0.4 gives a good balance between AIMD- friendliness and aggressiveness of window increase. Therefore, `_C_` SHOULD be set to 0.4. With `_C_` set to 0.4, Figure 6 is reduced to:

$$\text{AVG\_W}_{\text{cubic}} = 1.054 * \frac{3 \sqrt[4]{\text{RTT}}}{3 \sqrt[4]{p}}$$

Figure 7

Figure 7 is then used in the next subsection to show the scalability of CUBIC.

## 5.2. Using Spare Capacity

CUBIC uses a more aggressive window increase function than AIMD TCP for fast and long-distance networks.

The following table shows that to achieve the 10 Gbps rate, AIMD TCP requires a packet loss rate of  $2.0e-10$ , while CUBIC requires a packet loss rate of  $2.9e-8$ .

Throughput (Mbps)	Average W	AIMD P	HSTCP P	CUBIC P
1	8.3	2.0e-2	2.0e-2	2.0e-2
10	83.3	2.0e-4	3.9e-4	2.9e-4
100	833.3	2.0e-6	2.5e-5	1.4e-5
1000	8333.3	2.0e-8	1.5e-6	6.3e-7
10000	83333.3	2.0e-10	1.0e-7	2.9e-8

Table 3: Required packet loss rate for AIMD TCP, HSTCP, and CUBIC to achieve a certain throughput

Table 3 describes the required packet loss rate for AIMD TCP, HSTCP, and CUBIC to achieve a certain throughput. We use 1500-byte packets and an `_RTT_` of 0.1 seconds.

Our test results in [HKLRX06] indicate that CUBIC uses the spare bandwidth left unused by existing AIMD TCP flows in the same bottleneck link without taking away much bandwidth from the existing flows.

### 5.3. Difficult Environments

CUBIC is designed to remedy the poor performance of AIMD TCP in fast and long-distance networks.

### 5.4. Investigating a Range of Environments

There is decade-long deployment experience with CUBIC on the Internet. CUBIC has also been extensively studied by using both NS-2 simulation and testbed experiments, covering a wide range of network environments. More information can be found in [HKLRX06].

Same as AIMD TCP, CUBIC is a loss-based congestion control algorithm. Because CUBIC is designed to be more aggressive (due to a faster window increase function and bigger multiplicative decrease factor) than AIMD TCP in fast and long-distance networks, it can fill large drop-tail buffers more quickly than AIMD TCP and increases the risk of a standing queue [RFC8511]. In this case, proper queue sizing and management [RFC7567] could be used to reduce the packet queuing delay.

### 5.5. Protection against Congestion Collapse

With regard to the potential of causing congestion collapse, CUBIC behaves like AIMD TCP, since CUBIC modifies only the window adjustment algorithm of AIMD TCP. Thus, it does not modify the ACK clocking and timeout behaviors of AIMD TCP.

### 5.6. Fairness within the Alternative Congestion Control Algorithm

CUBIC ensures convergence of competing CUBIC flows with the same RTT in the same bottleneck links to an equal throughput. When competing flows have different RTT values, their throughput ratio is linearly proportional to the inverse of their RTT ratios. This is true independently of the level of statistical multiplexing on the link.

### 5.7. Performance with Misbehaving Nodes and Outside Attackers

This is not considered in the current CUBIC design.

### 5.8. Behavior for Application-Limited Flows

CUBIC does not increase its congestion window size if a flow is currently limited by the application instead of the congestion window. In case of long periods during which `_cwnd_` has not been updated due to such an application limit, such as idle periods, `_t_` in Figure 1 MUST NOT include these periods; otherwise, `W_cubic(_t_)` might be very high after restarting from these periods.

### 5.9. Responses to Sudden or Transient Events

If there is a sudden congestion, a routing change, or a mobility event, CUBIC behaves the same as AIMD TCP.

### 5.10. Incremental Deployment

CUBIC requires only changes to TCP senders, and it does not require any changes at TCP receivers. That is, a CUBIC sender works correctly with the AIMD TCP receivers. In addition, CUBIC does not require any changes to routers and does not require any assistance from routers.

## 6. Security Considerations

CUBIC makes no changes to the underlying security of TCP. More information about TCP security concerns can be found in [RFC5681].

## 7. IANA Considerations

This document does not require any IANA actions.

## 8. References

### 8.1. 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/rfc/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/rfc/rfc3168>>.
- [RFC5033] Floyd, S. and M. Allman, "Specifying New Congestion Control Algorithms", BCP 133, RFC 5033, DOI 10.17487/RFC5033, August 2007, <<https://www.rfc-editor.org/rfc/rfc5033>>.
- [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <<https://www.rfc-editor.org/rfc/rfc5348>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/rfc/rfc5681>>.
- [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/rfc/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/rfc/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/rfc/rfc6675>>.

- [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/rfc/rfc7567>>.
- [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/rfc/rfc8174>>.

## 8.2. Informative References

- [CEHRX07] Cai, H., Eun, D., Ha, S., Rhee, I., and L. Xu, "Stochastic Ordering for Internet Congestion Control and its Applications", IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications, DOI 10.1109/infcom.2007.111, 2007, <<https://doi.org/10.1109/infcom.2007.111>>.
- [FHP00] Floyd, S., Handley, M., and J. Padhye, "A Comparison of Equation-Based and AIMD Congestion Control", May 2000, <<https://www.icir.org/tfrc/aimd.pdf>>.
- [GV02] Gorinsky, S. and H. Vin, "Extended Analysis of Binary Adjustment Algorithms", Technical Report TR2002-29, Department of Computer Sciences, The University of Texas at Austin, 11 August 2002, <<http://www.cs.utexas.edu/ftp/techreports/tr02-39.ps.gz>>.
- [HKLRX06] Ha, S., Kim, Y., Le, L., Rhee, I., and L. Xu, "A Step toward Realistic Performance Evaluation of High-Speed TCP Variants", International Workshop on Protocols for Fast Long-Distance Networks, February 2006, <[https://pfld.net/2006/paper/s2\\_03.pdf](https://pfld.net/2006/paper/s2_03.pdf)>.
- [HR08] Ha, S. and I. Rhee, "Hybrid Slow Start for High-Bandwidth and Long-Distance Networks", International Workshop on Protocols for Fast Long-Distance Networks, March 2008, <[http://www.hep.man.ac.uk/g/GDARN-IT/pfldnet2008/paper/Sangate\\_Ha%20Final.pdf](http://www.hep.man.ac.uk/g/GDARN-IT/pfldnet2008/paper/Sangate_Ha%20Final.pdf)>.
- [HRX08] Ha, S., Rhee, I., and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant", ACM SIGOPS Operating Systems Review Vol. 42, pp. 64-74, DOI 10.1145/1400097.1400105, July 2008, <<https://doi.org/10.1145/1400097.1400105>>.

- [K03] Kelly, T., "Scalable TCP: improving performance in highspeed wide area networks", ACM SIGCOMM Computer Communication Review Vol. 33, pp. 83-91, DOI 10.1145/956981.956989, April 2003, <<https://doi.org/10.1145/956981.956989>>.
- [RFC3522] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC 3522, DOI 10.17487/RFC3522, April 2003, <<https://www.rfc-editor.org/rfc/rfc3522>>.
- [RFC3649] Floyd, S., "HighSpeed TCP for Large Congestion Windows", RFC 3649, DOI 10.17487/RFC3649, December 2003, <<https://www.rfc-editor.org/rfc/rfc3649>>.
- [RFC3708] Blanton, E. and M. Allman, "Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions", RFC 3708, DOI 10.17487/RFC3708, February 2004, <<https://www.rfc-editor.org/rfc/rfc3708>>.
- [RFC3742] Floyd, S., "Limited Slow-Start for TCP with Large Congestion Windows", RFC 3742, DOI 10.17487/RFC3742, March 2004, <<https://www.rfc-editor.org/rfc/rfc3742>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/rfc/rfc4960>>.
- [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/rfc/rfc5682>>.
- [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/rfc/rfc7661>>.
- [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/rfc/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/rfc/rfc8511>>.
- [SXEZ19] Sun, W., Xu, L., Elbaum, S., and D. Zhao, "Model-Agnostic and Efficient Exploration of Numerical State Space of Real-World TCP Congestion Control Implementations", USENIX NSDI 2019, February 2019, <<https://www.usenix.org/system/files/nsdi19-sun.pdf>>.
- [XHR04] Xu, L., Harfoush, K., and I. Rhee, "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks", IEEE INFOCOM 2004, DOI 10.1109/infcom.2004.1354672, March 2004, <<https://doi.org/10.1109/infcom.2004.1354672>>.

#### Appendix A. Acknowledgments

Richard Scheffenegger and Alexander Zimmermann originally co-authored [RFC8312].

#### Appendix B. Evolution of CUBIC

##### B.1. Since draft-ietf-tcpm-rfc8312bis-01

- \* address Michael Scharf's editorial suggestions. (#59 (<https://github.com/NTAP/rfc8312bis/issues/59>))
- \* add "Note to the RFC Editor" about removing underscores

##### B.2. Since draft-ietf-tcpm-rfc8312bis-00

- \* use updated xml2rfc with better text rendering of subscripts

##### B.3. Since draft-eggert-tcpm-rfc8312bis-03

- \* fix spelling nits
- \* rename to draft-ietf
- \* define `_W_max_` more clearly

##### B.4. Since draft-eggert-tcpm-rfc8312bis-02

- \* add definition for `segments_acked` and `alpha__aimd_`. (#47 (<https://github.com/NTAP/rfc8312bis/issues/47>))

- \* fix a mistake in `_W_max_` calculation in the fast convergence section. (#51 (<https://github.com/NTAP/rfc8312bis/issues/51>))
- \* clarity on setting `_ssthresh_` and `_cwnd_start_` during multiplicative decrease. (#53 (<https://github.com/NTAP/rfc8312bis/issues/53>))

B.5. Since draft-eggert-tcpm-rfc8312bis-01

- \* rename TCP-Friendly to AIMD-Friendly and rename Standard TCP to AIMD TCP to avoid confusion as CUBIC has been widely used on the Internet. (#38 (<https://github.com/NTAP/rfc8312bis/issues/38>))
- \* change introductory text to reflect the significant broader deployment of CUBIC on the Internet. (#39 (<https://github.com/NTAP/rfc8312bis/issues/39>))
- \* rephrase introduction to avoid referring to variables that have not been defined yet.

B.6. Since draft-eggert-tcpm-rfc8312bis-00

- \* acknowledge former co-authors (#15 (<https://github.com/NTAP/rfc8312bis/issues/15>))
- \* prevent `_cwnd_` from becoming less than two (#7 (<https://github.com/NTAP/rfc8312bis/issues/7>))
- \* add list of variables and constants (#5 (<https://github.com/NTAP/rfc8312bis/issues/5>), #6 (<https://github.com/NTAP/rfc8312bis/issues/6>))
- \* update `_K_'s` definition and add bounds for CUBIC `_target_ _cwnd_ [SXEZ19]` (#1 (<https://github.com/NTAP/rfc8312bis/issues/1>), #14 (<https://github.com/NTAP/rfc8312bis/issues/14>))
- \* update `_W_est_` to use AIMD approach (#20 (<https://github.com/NTAP/rfc8312bis/issues/20>))
- \* set `alpha_aimd_` to 1 once `_W_est_` reaches `_W_max_` (#2 (<https://github.com/NTAP/rfc8312bis/issues/2>))
- \* add Vidhi as co-author (#17 (<https://github.com/NTAP/rfc8312bis/issues/17>))
- \* note for Fast Recovery during `_cwnd_` decrease due to congestion event (#11 (<https://github.com/NTAP/rfc8312bis/issues/11>))

- \* add section for spurious congestion events (#23 (<https://github.com/NTAP/rfc8312bis/issues/23>))
- \* initialize `_W_est_` after timeout and remove variable `_W_(last_max)_` (#28 (<https://github.com/NTAP/rfc8312bis/issues/28>))

#### B.7. Since RFC8312

- \* converted to Markdown and xml2rfc v3
- \* updated references (as part of the conversion)
- \* updated author information
- \* various formatting changes
- \* move to Standards Track

#### B.8. Since the Original Paper

CUBIC has gone through a few changes since the initial release [HRX08] of its algorithm and implementation. Below we highlight the differences between its original paper and [RFC8312].

- \* The original paper [HRX08] includes the pseudocode of CUBIC implementation using Linux's pluggable congestion control framework, which excludes system-specific optimizations. The simplified pseudocode might be a good source to start with and understand CUBIC.
- \* [HRX08] also includes experimental results showing its performance and fairness.
- \* The definition of `beta__cubic_` constant was changed in [RFC8312]. For example, `beta__cubic_` in the original paper was the window decrease constant while [RFC8312] changed it to CUBIC multiplication decrease factor. With this change, the current congestion window size after a congestion event in [RFC8312] was `beta__cubic_ * _W_max_` while it was `(1-beta__cubic_) * _W_max_` in the original paper.
- \* Its pseudocode used `_W_(last_max)_` while [RFC8312] used `_W_max_`.
- \* Its AIMD-friendly window was `_W_tcp_` while [RFC8312] used `_W_est_`.

#### Authors' Addresses

Lisong Xu  
University of Nebraska-Lincoln  
Department of Computer Science and Engineering  
Lincoln, NE 68588-0115  
United States of America

Email: [xu@unl.edu](mailto:xu@unl.edu)  
URI: <https://cse.unl.edu/~xu/>

Sangtae Ha  
University of Colorado at Boulder  
Department of Computer Science  
Boulder, CO 80309-0430  
United States of America

Email: [sangtae.ha@colorado.edu](mailto:sangtae.ha@colorado.edu)  
URI: <https://netstech.org/sangtaeha/>

Injong Rhee  
Bowery Farming  
151 W 26TH Street, 12TH Floor  
New York, NY 10001  
United States of America

Email: [injongrhee@gmail.com](mailto:injongrhee@gmail.com)

Vidhi Goel  
Apple Inc.  
One Apple Park Way  
Cupertino, California 95014  
United States of America

Email: [vidhi\\_goel@apple.com](mailto:vidhi_goel@apple.com)

Lars Eggert (editor)  
NetApp  
Stenbergintie 12 B  
FI-02700 Kauniainen  
Finland

Email: [lars@eggert.org](mailto:lars@eggert.org)  
URI: <https://eggert.org/>

TCPM  
Internet-Draft  
Obsoletes: 8312 (if approved)  
Updates: 5681 (if approved)  
Intended status: Standards Track  
Expires: 5 September 2022

L. Xu  
UNL  
S. Ha  
Colorado  
I. Rhee  
Bowery  
V. Goel  
Apple Inc.  
L. Eggert, Ed.  
NetApp  
4 March 2022

CUBIC for Fast and Long-Distance Networks  
draft-ietf-tcpm-rfc8312bis-07

## Abstract

CUBIC is a standard TCP congestion control algorithm that uses a cubic function instead of a linear congestion window increase function to improve scalability and stability over fast and long-distance networks. CUBIC has been adopted as the default TCP congestion control algorithm by the Linux, Windows, and Apple stacks.

This document updates the specification of CUBIC to include algorithmic improvements based on these implementations and recent academic work. Based on the extensive deployment experience with CUBIC, it also moves the specification to the Standards Track, obsoleting RFC 8312. This also requires updating RFC 5681, to allow for CUBIC's occasionally more aggressive sending behavior.

## About This Document

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

Status information for this document may be found at  
<https://datatracker.ietf.org/doc/draft-ietf-tcpm-rfc8312bis/>.

Discussion of this document takes place on the TCPM Working Group mailing list (<mailto:tcpm@ietf.org>), which is archived at <https://mailarchive.ietf.org/arch/browse/tcpm/>.

Source for this draft and an issue tracker can be found at  
<https://github.com/NTAP/rfc8312bis>.

## Note to the RFC Editor

xml2rfc currently renders `<em></em>` in the XML by surrounding the corresponding text with underscores. This is highly distracting; please manually remove the underscores when doing the final edits to the text version of this document.

(There is an issue open against xml2rfc to stop doing this in the future: <https://trac.tools.ietf.org/tools/xml2rfc/trac/ticket/596>)

Also, please manually change "Figure" to "Equation" for all artwork with anchors beginning with "eq" - xml2rfc doesn't seem to be able to do this.

#### Status of This Memo

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

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

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

This Internet-Draft will expire on 5 September 2022.

#### Copyright Notice

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

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

#### Table of Contents

1. Introduction . . . . .	4
2. Conventions . . . . .	5
3. Design Principles of CUBIC . . . . .	5

3.1.	Principle 1 for the CUBIC Increase Function . . . . .	6
3.2.	Principle 2 for Reno-Friendliness . . . . .	6
3.3.	Principle 3 for RTT Fairness . . . . .	7
3.4.	Principle 4 for the CUBIC Decrease Factor . . . . .	7
4.	CUBIC Congestion Control . . . . .	8
4.1.	Definitions . . . . .	8
4.1.1.	Constants of Interest . . . . .	8
4.1.2.	Variables of Interest . . . . .	8
4.2.	Window Increase Function . . . . .	9
4.3.	Reno-Friendly Region . . . . .	11
4.4.	Concave Region . . . . .	13
4.5.	Convex Region . . . . .	13
4.6.	Multiplicative Decrease . . . . .	14
4.7.	Fast Convergence . . . . .	15
4.8.	Timeout . . . . .	16
4.9.	Spurious Congestion Events . . . . .	16
4.9.1.	Spurious timeout . . . . .	16
4.9.2.	Spurious loss detected by acknowledgements . . . . .	17
4.10.	Slow Start . . . . .	18
5.	Discussion . . . . .	18
5.1.	Fairness to Reno . . . . .	19
5.2.	Using Spare Capacity . . . . .	21
5.3.	Difficult Environments . . . . .	22
5.4.	Investigating a Range of Environments . . . . .	22
5.5.	Protection against Congestion Collapse . . . . .	23
5.6.	Fairness within the Alternative Congestion Control Algorithm . . . . .	23
5.7.	Performance with Misbehaving Nodes and Outside Attackers . . . . .	23
5.8.	Behavior for Application-Limited Flows . . . . .	23
5.9.	Responses to Sudden or Transient Events . . . . .	24
5.10.	Incremental Deployment . . . . .	24
6.	Security Considerations . . . . .	24
7.	IANA Considerations . . . . .	24
8.	References . . . . .	24
8.1.	Normative References . . . . .	24
8.2.	Informative References . . . . .	26
Appendix A.	Acknowledgments . . . . .	29
Appendix B.	Evolution of CUBIC . . . . .	30
B.1.	Since draft-ietf-tcpm-rfc8312bis-06 . . . . .	30
B.2.	Since draft-ietf-tcpm-rfc8312bis-05 . . . . .	30
B.3.	Since draft-ietf-tcpm-rfc8312bis-04 . . . . .	30
B.4.	Since draft-ietf-tcpm-rfc8312bis-03 . . . . .	31
B.5.	Since draft-ietf-tcpm-rfc8312bis-02 . . . . .	31
B.6.	Since draft-ietf-tcpm-rfc8312bis-01 . . . . .	32
B.7.	Since draft-ietf-tcpm-rfc8312bis-00 . . . . .	32
B.8.	Since draft-eggert-tcpm-rfc8312bis-03 . . . . .	32
B.9.	Since draft-eggert-tcpm-rfc8312bis-02 . . . . .	32

B.10. Since draft-eggert-tcpm-rfc8312bis-01 . . . . .	32
B.11. Since draft-eggert-tcpm-rfc8312bis-00 . . . . .	33
B.12. Since RFC8312 . . . . .	33
B.13. Since the Original Paper . . . . .	34
Authors' Addresses . . . . .	34

## 1. Introduction

CUBIC has been adopted as the default TCP congestion control algorithm in the Linux, Windows, and Apple stacks, and has been used and deployed globally. Extensive, decade-long deployment experience in vastly different Internet scenarios has convincingly demonstrated that CUBIC is safe for deployment on the global Internet and delivers substantial benefits over classical Reno congestion control [RFC5681]. It is therefore to be regarded as the currently most widely deployed standard for TCP congestion control. CUBIC can also be used for other transport protocols such as QUIC [RFC9000] and SCTP [RFC4960] as a default congestion controller.

The design of CUBIC was motivated by the well-documented problem classical Reno TCP has with low utilization over fast and long-distance networks [K03][RFC3649]. This problem arises from a slow increase of the congestion window following a congestion event in a network with a large bandwidth-delay product (BDP). [HLRX07] indicates that this problem is frequently observed even in the range of congestion window sizes over several hundreds of packets. This problem is equally applicable to all Reno-style standards and their variants, including TCP-Reno [RFC5681], TCP-NewReno [RFC6582][RFC6675], SCTP [RFC4960], TFRC [RFC5348], and QUIC congestion control [RFC9002], which use the same linear increase function for window growth. We refer to all Reno-style standards and their variants collectively as "Reno" below.

CUBIC, originally proposed in [HRX08], is a modification to the congestion control algorithm of classical Reno to remedy this problem. Specifically, CUBIC uses a cubic function instead of the linear window increase function of Reno to improve scalability and stability under fast and long-distance networks.

This document updates the specification of CUBIC to include algorithmic improvements based on the Linux, Windows, and Apple implementations and recent academic work. Based on the extensive deployment experience with CUBIC, it also moves the specification to the Standards Track, obsoleting [RFC8312]. This requires an update to [RFC5681], which limits the aggressiveness of Reno TCP implementations in its Section 3. Since CUBIC is occasionally more aggressive than the [RFC5681] algorithms, this document updates [RFC5681] to allow for CUBIC's behavior.

Binary Increase Congestion Control (BIC-TCP) [XHR04], a predecessor of CUBIC, was selected as the default TCP congestion control algorithm by Linux in the year 2005 and had been used for several years by the Internet community at large.

CUBIC uses a similar window increase function as BIC-TCP and is designed to be less aggressive and fairer to Reno in bandwidth usage than BIC-TCP while maintaining the strengths of BIC-TCP such as stability, window scalability, and round-trip time (RTT) fairness.

In the following sections, we first briefly explain the design principles of CUBIC, then provide the exact specification of CUBIC, and finally discuss the safety features of CUBIC following the guidelines specified in [RFC5033].

## 2. Conventions

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

## 3. Design Principles of CUBIC

CUBIC is designed according to the following design principles:

Principle 1: For better network utilization and stability, CUBIC uses both the concave and convex profiles of a cubic function to increase the congestion window size, instead of using just a convex function.

Principle 2: To be Reno-friendly, CUBIC is designed to behave like Reno in networks with short RTTs and small bandwidth where Reno performs well.

Principle 3: For RTT-fairness, CUBIC is designed to achieve linear bandwidth sharing among flows with different RTTs.

Principle 4: CUBIC appropriately sets its multiplicative window decrease factor in order to balance between the scalability and convergence speed.

### 3.1. Principle 1 for the CUBIC Increase Function

For better network utilization and stability, CUBIC [HRX08] uses a cubic window increase function in terms of the elapsed time from the last congestion event. While most alternative congestion control algorithms to Reno increase the congestion window using convex functions, CUBIC uses both the concave and convex profiles of a cubic function for window growth.

After a window reduction in response to a congestion event detected by duplicate ACKs, Explicit Congestion Notification-Echo (ECN-Echo, ECE) ACKs [RFC3168], TCP RACK [RFC8985] or QUIC loss detection [RFC9002], CUBIC remembers the congestion window size at which it received the congestion event and performs a multiplicative decrease of the congestion window. When CUBIC enters into congestion avoidance, it starts to increase the congestion window using the concave profile of the cubic function. The cubic function is set to have its plateau at the remembered congestion window size, so that the concave window increase continues until then. After that, the cubic function turns into a convex profile and the convex window increase begins.

This style of window adjustment (concave and then convex) improves the algorithm stability while maintaining high network utilization [CEHRX09]. This is because the window size remains almost constant, forming a plateau around the remembered congestion window size of the last congestion event, where network utilization is deemed highest. Under steady state, most window size samples of CUBIC are close to that remembered congestion window size, thus promoting high network utilization and stability.

Note that congestion control algorithms that only use convex functions to increase the congestion window size have their maximum increments around the remembered congestion window size of the last congestion event, and thus introduce many packet bursts around the saturation point of the network, likely causing frequent global loss synchronizations.

### 3.2. Principle 2 for Reno-Friendliness

CUBIC promotes per-flow fairness to Reno. Note that Reno performs well over paths with short RTTs and small bandwidths (or small BDPs). There is only a scalability problem in networks with long RTTs and large bandwidths (or large BDPs).

A congestion control algorithm designed to be friendly to Reno on a per-flow basis must increase its congestion window less aggressively in small BDP networks than in large BDP networks.

The aggressiveness of CUBIC mainly depends on the maximum window size before a window reduction, which is smaller in small-BDP networks than in large-BDP networks. Thus, CUBIC increases its congestion window less aggressively in small-BDP networks than in large-BDP networks.

Furthermore, in cases when the cubic function of CUBIC would increase the congestion window less aggressively than Reno, CUBIC simply follows the window size of Reno to ensure that CUBIC achieves at least the same throughput as Reno in small-BDP networks. We call this region where CUBIC behaves like Reno the "Reno-friendly region".

### 3.3. Principle 3 for RTT Fairness

Two CUBIC flows with different RTTs have a throughput ratio that is linearly proportional to the inverse of their RTT ratio, where the throughput of a flow is approximately the size of its congestion window divided by its RTT.

Specifically, CUBIC maintains a window increase rate independent of RTTs outside the Reno-friendly region, and thus flows with different RTTs have similar congestion window sizes under steady state when they operate outside the Reno-friendly region.

This notion of a linear throughput ratio is similar to that of Reno under high statistical multiplexing where packet loss is independent of individual flow rates. However, under low statistical multiplexing, the throughput ratio of Reno flows with different RTTs is quadratically proportional to the inverse of their RTT ratio [XHR04].

CUBIC always ensures a linear throughput ratio independent of the amount of statistical multiplexing. This is an improvement over Reno. While there is no consensus on particular throughput ratios for different RTT flows, we believe that over wired Internet paths, use of a linear throughput ratio seems more reasonable than equal throughputs (i.e., the same throughput for flows with different RTTs) or a higher-order throughput ratio (e.g., a quadratical throughput ratio of Reno under low statistical multiplexing environments).

### 3.4. Principle 4 for the CUBIC Decrease Factor

To balance between scalability and convergence speed, CUBIC sets the multiplicative window decrease factor to 0.7, whereas Reno uses 0.5.

While this improves the scalability of CUBIC, a side effect of this decision is slower convergence, especially under low statistical multiplexing. This design choice is following the observation that

HighSpeed TCP (HSTCP) [RFC3649] and other approaches (e.g., [GV02]) made: the current Internet becomes more asynchronous with less frequent loss synchronizations under high statistical multiplexing.

In such environments, even strict Multiplicative-Increase Multiplicative-Decrease (MIMD) can converge. CUBIC flows with the same RTT always converge to the same throughput independent of statistical multiplexing, thus achieving intra-algorithm fairness. We also find that in environments with sufficient statistical multiplexing, the convergence speed of CUBIC is reasonable.

#### 4. CUBIC Congestion Control

In this section, we discuss how the congestion window is updated during the different stages of the CUBIC congestion controller.

##### 4.1. Definitions

The unit of all window sizes in this document is segments of the maximum segment size (MSS), and the unit of all times is seconds. Implementations can use bytes to express window sizes, which would require factoring in the maximum segment size wherever necessary and replacing `_segments_acked_` with the number of bytes acknowledged in Figure 4.

###### 4.1.1. Constants of Interest

`__cubic_`: CUBIC multiplicative decrease factor as described in Section 4.6.

`__cubic_`: CUBIC additive increase factor used in Reno-friendly region as described in Section 4.3.

`_C_`: constant that determines the aggressiveness of CUBIC in competing with other congestion control algorithms in high BDP networks. Please see Section 5 for more explanation on how it is set. The unit for `_C_` is

segment  
-----  
3  
second

###### 4.1.2. Variables of Interest

This section defines the variables required to implement CUBIC:

`_RTT_`: Smoothed round-trip time in seconds, calculated as described in [RFC6298].

`_cwnd_`: Current congestion window in segments.

`_ssthresh_`: Current slow start threshold in segments.

`_W_max_`: Size of `_cwnd_` in segments just before `_cwnd_` was reduced in the last congestion event when fast convergence is disabled. However, if fast convergence is enabled, the size may be further reduced based on the current saturation point.

`_K_`: The time period in seconds it takes to increase the congestion window size at the beginning of the current congestion avoidance stage to `_W_max_`.

`_current_time_`: Current time of the system in seconds.

`_epoch_start_`: The time in seconds at which the current congestion avoidance stage started.

`_cwnd_start_`: The `_cwnd_` at the beginning of the current congestion avoidance stage, i.e., at time `_epoch_start_`.

`W_cubic(t_)`: The congestion window in segments at time `t_` in seconds based on the cubic increase function, as described in Section 4.2.

`_target_`: Target value of congestion window in segments after the next RTT, that is, `W_cubic(t_ + _RTT_)`, as described in Section 4.2.

`_W_est_`: An estimate for the congestion window in segments in the Reno-friendly region, that is, an estimate for the congestion window of Reno.

`_segments_acked_`: Number of MSS-sized segments acked when a "new ACK" is received, i.e., an ACK that cumulatively acknowledges the delivery of new data. This number will be a decimal value when a new ACK acknowledges an amount of data that is not MSS-sized. Specifically, it can be less than 1 when a new ACK acknowledges a segment smaller than the MSS.

#### 4.2. Window Increase Function

CUBIC maintains the acknowledgment (ACK) clocking of Reno by increasing the congestion window only at the reception of a new ACK. It does not make any changes to the TCP Fast Recovery and Fast Retransmit algorithms [RFC6582][RFC6675].

During congestion avoidance, after a congestion event is detected by mechanisms described in Section 3.1, CUBIC uses a window increase function different from Reno.

CUBIC uses the following window increase function:

$$W_{\text{cubic}}(t) = C * (t - K)^3 + W_{\text{max}}$$

Figure 1

where `_t_` is the elapsed time in seconds from the beginning of the current congestion avoidance stage, that is,

$$t = \text{current\_time} - \text{epoch\_start}$$

and where `_epoch_start_` is the time at which the current congestion avoidance stage starts. `_K_` is the time period that the above function takes to increase the congestion window size at the beginning of the current congestion avoidance stage to `_W_max_` if there are no further congestion events and is calculated using the following equation:

$$K = \sqrt[3]{\frac{W_{\text{max}} - \text{cwnd}_{\text{start}}}{C}}$$

Figure 2

where `_cwnd_start_` is the congestion window at the beginning of the current congestion avoidance stage.

Upon receiving a new ACK during congestion avoidance, CUBIC computes the `_target_` congestion window size after the next `_RTT_` using Figure 1 as follows, where `_RTT_` is the smoothed round-trip time. The lower and upper bounds below ensure that CUBIC's congestion window increase rate is non-decreasing and is less than the increase rate of slow start [SXEZ19].

$$\text{target} = \begin{cases} \text{cwnd} & \text{if } W(t + \text{RTT}) < \text{cwnd} \\ 1.5 * \text{cwnd} & \text{if } W(t + \text{RTT}) > 1.5 * \text{cwnd} \\ W(t + \text{RTT}) & \text{otherwise} \end{cases}$$

cubic                      cubic                      cubic                      otherwise

The elapsed time `_t_` in Figure 1 MUST NOT include periods during which `_cwnd_` has not been updated due to application-limited behavior (see Section 5.8).

Depending on the value of the current congestion window size `_cwnd_`, CUBIC runs in three different regions:

1. The Reno-friendly region, which ensures that CUBIC achieves at least the same throughput as Reno.
2. The concave region, if CUBIC is not in the Reno-friendly region and `_cwnd_` is less than `_W_max_`.
3. The convex region, if CUBIC is not in the Reno-friendly region and `_cwnd_` is greater than `_W_max_`.

Below, we describe the exact actions taken by CUBIC in each region.

#### 4.3. Reno-Friendly Region

Reno performs well in certain types of networks, for example, under short RTTs and small bandwidths (or small BDPs). In these networks, CUBIC remains in the Reno-friendly region to achieve at least the same throughput as Reno.

The Reno-friendly region is designed according to the analysis in [FHP00], which studies the performance of an AIMD algorithm with an additive factor of  $\frac{1}{2}$  (segments per `_RTT_`) and a multiplicative factor of  $\frac{1}{2}$ , denoted by AIMD( $\frac{1}{2}$ ,  $\frac{1}{2}$ ). `_p_` is the packet loss rate. Specifically, the average congestion window size of AIMD( $\frac{1}{2}$ ,  $\frac{1}{2}$ ) can be calculated using Figure 3.

$$\text{AVG\_AIMD}(, ) = \frac{\frac{1}{2} * (1 + )}{\frac{1}{2} * (1 - ) * p}$$

Figure 3

By the same analysis, to achieve the same average window size as Reno that uses AIMD(1, 0.5), must be equal to,

$$3 * \frac{1 -}{1 +}$$

Thus, CUBIC uses Figure 4 to estimate the window size `_W_est_` in the Reno-friendly region with

$$\text{cubic} = 3 * \frac{1 - \text{cubic}}{1 + \text{cubic}}$$

which achieves the same average window size as Reno. When receiving a new ACK in congestion avoidance (where `_cwnd_` could be greater than or less than `_W_max_`), CUBIC checks whether `W_cubic(t_)` is less than `_W_est_`. If so, CUBIC is in the Reno-friendly region and `_cwnd_` SHOULD be set to `_W_est_` at each reception of a new ACK.

`_W_est_` is set equal to `_cwnd_start_` at the start of the congestion avoidance stage. After that, on every new ACK, `_W_est_` is updated using Figure 4. Note that this equation is for a connection where Appropriate Byte Counting (ABC) [RFC3465] is disabled. For a connection with ABC enabled, this equation SHOULD be adjusted by using the number of acknowledged bytes instead of acknowledged segments. Also note that this equation works for connections with enabled or disabled Delayed ACKs [RFC5681], as `_segments_acked_` will be different based on the segments actually acknowledged by a new ACK.

$$W_{\text{est}} = W_{\text{est}} + \frac{\text{segments\_acked}}{\text{cubic} * \text{cwnd}}$$

Figure 4

Note that once `_W_est_` reaches `_W_max_`, that is, `_W_est_ >= _W_max_`, CUBIC needs to start probing to determine the new value of `_W_max_`. At this point, `__cubic_` SHOULD be set to 1 to ensure that CUBIC can achieve the same congestion window increment as Reno, which uses AIMD(1, 0.5).

#### 4.4. Concave Region

When receiving a new ACK in congestion avoidance, if CUBIC is not in the Reno-friendly region and `_cwnd_` is less than `_W_max_`, then CUBIC is in the concave region. In this region, `_cwnd_` MUST be incremented by

$$\frac{\text{target} - \text{cwnd}}{\text{cwnd}}$$

for each received new ACK, where `_target_` is calculated as described in Section 4.2.

#### 4.5. Convex Region

When receiving a new ACK in congestion avoidance, if CUBIC is not in the Reno-friendly region and `_cwnd_` is larger than or equal to `_W_max_`, then CUBIC is in the convex region.

The convex region indicates that the network conditions might have changed since the last congestion event, possibly implying more available bandwidth after some flow departures. Since the Internet is highly asynchronous, some amount of perturbation is always possible without causing a major change in available bandwidth.

Unless it is overridden by the AIMD window increase, CUBIC is very careful in this region. The convex profile aims to increase the window very slowly at the beginning when `_cwnd_` is around `_W_max_` and then gradually increases its rate of increase. We also call this region the "maximum probing phase", since CUBIC is searching for a new `_W_max_`. In this region, `_cwnd_` MUST be incremented by

$$\frac{\text{target} - \text{cwnd}}{\text{cwnd}}$$

for each received new ACK, where `_target_` is calculated as described in Section 4.2.

#### 4.6. Multiplicative Decrease

When a congestion event is detected by mechanisms described in Section 3.1, CUBIC updates `_W_max_` and reduces `_cwnd_` and `_sssthresh_` immediately as described below. In case of packet loss, the sender MUST reduce `_cwnd_` and `_sssthresh_` immediately upon entering loss recovery, similar to [RFC5681] (and [RFC6675]). Note that other mechanisms, such as Proportional Rate Reduction [RFC6937], can be used to reduce the sending rate during loss recovery more gradually. The parameter `__cubic_` SHOULD be set to 0.7, which is different from the multiplicative decrease factor used in [RFC5681] (and [RFC6675]) during fast recovery.

In Figure 5, `_flight_size_` is the amount of outstanding data in the network, as defined in [RFC5681]. Note that a rate-limited application with idle periods or periods when unable to send at the full rate permitted by `_cwnd_` may easily encounter notable variations in the volume of data sent from one RTT to another, resulting in `_flight_size_` that is significantly less than `_cwnd_` on a congestion event. This may decrease `_cwnd_` to a much lower value than necessary. To avoid suboptimal performance with such applications, the mechanisms described in [RFC7661] can be used to mitigate this issue as it would allow using a value between `_cwnd_` and `_flight_size_` to calculate the new `_sssthresh_` in Figure 5. The congestion window growth mechanism defined in [RFC7661] is safe to use even when `_cwnd_` is greater than the receive window as it validates `_cwnd_` based on the amount of data acknowledged by the network in an RTT which implicitly accounts for the allowed receive window. Some implementations of CUBIC currently use `_cwnd_` instead of `_flight_size_` when calculating a new `_sssthresh_` using Figure 5.

```

sssthresh =      flight_size *      // new ssthresh
                  cubic

                  /max(sssthresh, 2) // reduction on packet loss, cwnd is at least 2
MSS
cwnd =          |
                <
                |max(sssthresh, 1)   // reduction on ECE, cwnd is at least 1 MSS
                \

sssthresh =      max(sssthresh, 2)   // ssthresh is at least 2 MSS

```

Figure 5

A side effect of setting `__cubic__` to a value bigger than 0.5 is slower convergence. We believe that while a more adaptive setting of `__cubic__` could result in faster convergence, it will make the analysis of CUBIC much harder.

Note that CUBIC MUST continue to reduce `_cwnd_` in response to congestion events due to ECN-Echo ACKs until it reaches a value of 1 MSS. If congestion events indicated by ECN-Echo ACKs persist, a sender with a `_cwnd_` of 1 MSS MUST reduce its sending rate even further. It can achieve that by using a retransmission timer with exponential backoff, as described in [RFC3168].

#### 4.7. Fast Convergence

To improve convergence speed, CUBIC uses a heuristic. When a new flow joins the network, existing flows need to give up some of their bandwidth to allow the new flow some room for growth, if the existing flows have been using all the network bandwidth. To speed up this bandwidth release by existing flows, the following "Fast Convergence" mechanism SHOULD be implemented.

With Fast Convergence, when a congestion event occurs, we update `_W_max_` as follows, before the window reduction as described in Section 4.6.

$$W_{\max} = \begin{cases} \frac{1 + \text{cubic if } cwnd < W_{\max} \text{ and fast convergence is enabled,}}{2} \\ \text{further reduce } W_{\max} \\ \text{otherwise, remember } cwnd \text{ before reduction} \end{cases} \backslash cwnd$$

At a congestion event, if the current `_cwnd_` is less than `_W_max_`, this indicates that the saturation point experienced by this flow is getting reduced because of a change in available bandwidth. Then we allow this flow to release more bandwidth by reducing `_W_max_` further. This action effectively lengthens the time for this flow to increase its congestion window, because the reduced `_W_max_` forces the flow to plateau earlier. This allows more time for the new flow to catch up to its congestion window size.

Fast Convergence is designed for network environments with multiple CUBIC flows. In network environments with only a single CUBIC flow and without any other traffic, Fast Convergence SHOULD be disabled.

#### 4.8. Timeout

In case of a timeout, CUBIC follows Reno to reduce `_cwnd_` [RFC5681], but sets `_ssthresh_` using `__cubic_` (same as in Section 4.6) in a way that is different from Reno TCP [RFC5681].

During the first congestion avoidance stage after a timeout, CUBIC increases its congestion window size using Figure 1, where `_t_` is the elapsed time since the beginning of the current congestion avoidance, `_K_` is set to 0, and `_W_max_` is set to the congestion window size at the beginning of the current congestion avoidance stage. In addition, for the Reno-friendly region, `_W_est_` SHOULD be set to the congestion window size at the beginning of the current congestion avoidance.

#### 4.9. Spurious Congestion Events

In cases where CUBIC reduces its congestion window in response to having detected packet loss via duplicate ACKs or timeouts, there is a possibility that the missing ACK would arrive after the congestion window reduction and a corresponding packet retransmission. For example, packet reordering could trigger this behavior. A high degree of packet reordering could cause multiple congestion window reduction events, where spurious losses are incorrectly interpreted as congestion signals, thus degrading CUBIC's performance significantly.

For TCP, there are two types of spurious events - spurious timeouts and spurious fast retransmits. In case of QUIC, there are no spurious timeouts as the loss is only detected after receiving an ACK.

##### 4.9.1. Spurious timeout

An implementation MAY detect spurious timeouts based on the mechanisms described in Forward RTO-Recovery [RFC5682]. Experimental alternatives include Eifel [RFC3522]. When a spurious timeout is detected, a TCP implementation MAY follow the response algorithm described in [RFC4015] to restore the congestion control state and adapt the retransmission timer to avoid further spurious timeouts.

#### 4.9.2. Spurious loss detected by acknowledgements

Upon receiving an ACK, a TCP implementation MAY detect spurious losses either using TCP Timestamps or via D-SACK[RFC2883]. Experimental alternatives include Eifel detection algorithm [RFC3522] which uses TCP Timestamps and DSACK based detection [RFC3708] which uses DSACK information. A QUIC implementation can easily determine a spurious loss if a QUIC packet is acknowledged after it has been marked as lost and the original data has been retransmitted with a new QUIC packet.

In this section, we specify a simple response algorithm when a spurious loss is detected by acknowledgements. Implementations would need to carefully evaluate the impact of using this algorithm in different environments that may experience sudden change in available capacity (e.g., due to variable radio capacity, a routing change, or a mobility event).

When a packet loss is detected via acknowledgements, a CUBIC implementation MAY save the current value of the following variables before the congestion window is reduced.

```
prior_cwnd = cwnd

prior_ssthresh = ssthresh

prior_W      = W
    max      max

prior_K = K

prior_epoch      = epoch
    start        start

prior_W_{est} = W
                est
```

Once the previously declared packet loss is confirmed to be spurious, CUBIC MAY restore the original values of the above-mentioned variables as follows if the current `_cwnd_` is lower than `_prior_cwnd_`. Restoring the original values ensures that CUBIC's performance is similar to what it would be without spurious losses.

```

        cwnd = prior_cwnd
        ssthresh = prior_ssthresh

        W      = prior_W
        max      max

        K = prior_K

        epoch      = prior_epoch
        start      start

        W      = prior_W
        est      est
    \
    >if cwnd < prior_cwnd
    /

```

In rare cases, when the detection happens long after a spurious loss event and the current `_cwnd` is already higher than `_prior_cwnd`, CUBIC SHOULD continue to use the current and the most recent values of these variables.

#### 4.10. Slow Start

CUBIC MUST employ a slow-start algorithm, when `_cwnd` is no more than `_ssthresh`. In general, CUBIC SHOULD use the HyStart++ slow start algorithm [I-D.ietf-tcpm-hystartplusplus], or MAY use the Reno TCP slow start algorithm [RFC5681] in the rare cases when HyStart++ is not suitable. Experimental alternatives include hybrid slow start [HR11], a predecessor to HyStart++ that some CUBIC implementations have used as the default for the last decade, and limited slow start [RFC3742]. Whichever start-up algorithm is used, work might be needed to ensure that the end of slow start and the first multiplicative decrease of congestion avoidance work well together.

When CUBIC uses HyStart++ [I-D.ietf-tcpm-hystartplusplus], it may exit the first slow start without incurring any packet loss and thus `_W_max` is undefined. In this special case, CUBIC switches to congestion avoidance and increases its congestion window size using Figure 1, where `_t` is the elapsed time since the beginning of the current congestion avoidance, `_K` is set to 0, and `_W_max` is set to the congestion window size at the beginning of the current congestion avoidance stage.

#### 5. Discussion

In this section, we further discuss the safety features of CUBIC following the guidelines specified in [RFC5033].

With a deterministic loss model where the number of packets between two successive packet losses is always  $\frac{1}{p}$ , CUBIC always operates with the concave window profile, which greatly simplifies the performance analysis of CUBIC. The average window size of CUBIC can be obtained by the following function:

$$AVG\_W_{cubic} = \frac{4}{\sqrt[4]{\frac{C * (3 + \frac{1}{p})}{cubic}}} * \frac{\sqrt[4]{\frac{3}{RTT}}}{\sqrt[4]{\frac{3}{p}}}$$

Figure 6

With `__cubic_` set to 0.7, the above formula reduces to:

$$AVG\_W_{cubic} = \frac{4}{\sqrt[4]{\frac{C * 3.7}{1.2}}} * \frac{\sqrt[4]{\frac{3}{RTT}}}{\sqrt[4]{\frac{3}{p}}}$$

Figure 7

We will determine the value of `_C_` in the following subsection using Figure 7.

### 5.1. Fairness to Reno

In environments where Reno is able to make reasonable use of the available bandwidth, CUBIC does not significantly change this state.

Reno performs well in the following two types of networks:

1. networks with a small bandwidth-delay product (BDP)
2. networks with a short RTTs, but not necessarily a small BDP

CUBIC is designed to behave very similarly to Reno in the above two types of networks. The following two tables show the average window sizes of Reno TCP, HSTCP, and CUBIC TCP. The average window sizes of Reno TCP and HSTCP are from [RFC3649]. The average window size of CUBIC is calculated using Figure 7 and the CUBIC Reno-friendly region for three different values of `_C_`.

Loss Rate P	Reno	HSTCP	CUBIC (C=0.04)	CUBIC (C=0.4)	CUBIC (C=4)
1.0e-02	12	12	12	12	12
1.0e-03	38	38	38	38	59
1.0e-04	120	263	120	187	333
1.0e-05	379	1795	593	1054	1874
1.0e-06	1200	12280	3332	5926	10538
1.0e-07	3795	83981	18740	33325	59261
1.0e-08	12000	574356	105383	187400	333250

Table 1: Reno TCP, HSTCP, and CUBIC with RTT = 0.1 seconds

Table 1 describes the response function of Reno TCP, HSTCP, and CUBIC in networks with `_RTT_` = 0.1 seconds. The average window size is in MSS-sized segments.

Loss Rate P	Reno	HSTCP	CUBIC (C=0.04)	CUBIC (C=0.4)	CUBIC (C=4)
1.0e-02	12	12	12	12	12
1.0e-03	38	38	38	38	38
1.0e-04	120	263	120	120	120
1.0e-05	379	1795	379	379	379
1.0e-06	1200	12280	1200	1200	1874
1.0e-07	3795	83981	3795	5926	10538
1.0e-08	12000	574356	18740	33325	59261

Table 2: Reno TCP, HSTCP, and CUBIC with RTT = 0.01 seconds

Table 2 describes the response function of Reno TCP, HSTCP, and CUBIC in networks with `_RTT_` = 0.01 seconds. The average window size is in MSS-sized segments.

Both tables show that CUBIC with any of these three `_C_` values is more friendly to Reno TCP than HSTCP, especially in networks with a short `_RTT_` where Reno TCP performs reasonably well. For example, in a network with `_RTT_` = 0.01 seconds and  $p=10^{-6}$ , Reno TCP has an average window of 1200 packets. If the packet size is 1500 bytes, then Reno TCP can achieve an average rate of 1.44 Gbps. In this case, CUBIC with `_C_=0.04` or `_C_=0.4` achieves exactly the same rate as Reno TCP, whereas HSTCP is about ten times more aggressive than Reno TCP.

We can see that `_C_` determines the aggressiveness of CUBIC in competing with other congestion control algorithms for bandwidth. CUBIC is more friendly to Reno TCP, if the value of `_C_` is lower. However, we do not recommend setting `_C_` to a very low value like 0.04, since CUBIC with a low `_C_` cannot efficiently use the bandwidth in fast and long-distance networks. Based on these observations and extensive deployment experience, we find `_C_=0.4` gives a good balance between Reno-friendliness and aggressiveness of window increase. Therefore, `_C_ SHOULD` be set to 0.4. With `_C_` set to 0.4, Figure 7 is reduced to:

$$\text{AVG\_W}_{\text{cubic}} = 1.054 * \frac{4 / \sqrt[3]{\text{RTT}}}{4 / \sqrt[3]{p}}$$

Figure 8

Figure 8 is then used in the next subsection to show the scalability of CUBIC.

## 5.2. Using Spare Capacity

CUBIC uses a more aggressive window increase function than Reno for fast and long-distance networks.

The following table shows that to achieve the 10 Gbps rate, Reno TCP requires a packet loss rate of  $2.0e-10$ , while CUBIC TCP requires a packet loss rate of  $2.9e-8$ .

Throughput (Mbps)	Average W	Reno P	HSTCP P	CUBIC P
1	8.3	2.0e-2	2.0e-2	2.0e-2
10	83.3	2.0e-4	3.9e-4	2.9e-4
100	833.3	2.0e-6	2.5e-5	1.4e-5
1000	8333.3	2.0e-8	1.5e-6	6.3e-7
10000	83333.3	2.0e-10	1.0e-7	2.9e-8

Table 3: Required packet loss rate for Reno TCP, HSTCP, and CUBIC to achieve a certain throughput

Table 3 describes the required packet loss rate for Reno TCP, HSTCP, and CUBIC to achieve a certain throughput. We use 1500-byte packets and an `_RTT_` of 0.1 seconds.

Our test results in [HLRX07] indicate that CUBIC uses the spare bandwidth left unused by existing Reno TCP flows in the same bottleneck link without taking away much bandwidth from the existing flows.

### 5.3. Difficult Environments

CUBIC is designed to remedy the poor performance of Reno in fast and long-distance networks.

### 5.4. Investigating a Range of Environments

CUBIC has been extensively studied using simulations, testbed emulations, Internet experiments, and Internet measurements, covering a wide range of network environments [HLRX07][H16][CEHRX09][HR11][BSCLU13][LBEWK16]. They have convincingly demonstrated that CUBIC delivers substantial benefits over classical Reno congestion control [RFC5681].

Same as Reno, CUBIC is a loss-based congestion control algorithm. Because CUBIC is designed to be more aggressive (due to a faster window increase function and bigger multiplicative decrease factor) than Reno in fast and long-distance networks, it can fill large drop-tail buffers more quickly than Reno and increases the risk of a standing queue [RFC8511]. In this case, proper queue sizing and management [RFC7567] could be used to mitigate the risk to some extent and reduce the packet queuing delay. Also, in large-BDP

networks after a congestion event, CUBIC, due its cubic window increase function, recovers quickly to the highest link utilization point. This means that link utilization is less sensitive to an active queue management (AQM) target that is lower than the amplitude of the whole sawtooth.

Similar to Reno, the performance of CUBIC as a loss-based congestion control algorithm suffers in networks where a packet loss is not a good indication of bandwidth utilization, such as wireless or mobile networks [LIU16].

#### 5.5. Protection against Congestion Collapse

With regard to the potential of causing congestion collapse, CUBIC behaves like Reno, since CUBIC modifies only the window adjustment algorithm of Reno. Thus, it does not modify the ACK clocking and timeout behaviors of Reno.

CUBIC also satisfies the "full backoff" requirement as described in [RFC5033]. After reducing the sending rate to one packet per RTT in response to congestion events due to ECN-Echo ACKs, CUBIC then exponentially increases the transmission timer for each packet retransmission while congestion persists.

#### 5.6. Fairness within the Alternative Congestion Control Algorithm

CUBIC ensures convergence of competing CUBIC flows with the same RTT in the same bottleneck links to an equal throughput. When competing flows have different RTT values, their throughput ratio is linearly proportional to the inverse of their RTT ratios. This is true independently of the level of statistical multiplexing on the link. The convergence time depends on the network environments (e.g., bandwidth, RTT) and the level of statistical multiplexing, as mentioned in Section 3.4.

#### 5.7. Performance with Misbehaving Nodes and Outside Attackers

This is not considered in the current CUBIC design.

#### 5.8. Behavior for Application-Limited Flows

A flow is application-limited if it is currently sending less than what is allowed by the congestion window. This can happen if the flow is limited by either the sender application or the receiver application (via the receiver advertised window) and thus sends less data than what is allowed by the sender's congestion window.

CUBIC does not increase its congestion window if a flow is application-limited. Section 4.2 requires that `_t_` in Figure 1 does not include application-limited periods, such as idle periods, otherwise `W_cubic(_t_)` might be very high after restarting from these periods.

#### 5.9. Responses to Sudden or Transient Events

If there is a sudden increase in capacity, e.g., due to variable radio capacity, a routing change, or a mobility event, CUBIC is designed to utilize the newly available capacity faster than Reno.

On the other hand, if there is a sudden decrease in capacity, CUBIC reduces more slowly than Reno. This remains true whether or not CUBIC is in Reno-friendly mode and whether or not fast convergence is enabled.

#### 5.10. Incremental Deployment

CUBIC requires only changes to the congestion control at the sender, and it does not require any changes at receivers. That is, a CUBIC sender works correctly with Reno receivers. In addition, CUBIC does not require any changes to routers and does not require any assistance from routers.

### 6. Security Considerations

CUBIC makes no changes to the underlying security of TCP. More information about TCP security concerns can be found in [RFC5681].

### 7. IANA Considerations

This document does not require any IANA actions.

### 8. References

#### 8.1. Normative References

[I-D.ietf-tcpm-hystartplusplus]  
Balasubramanian, P., Huang, Y., and M. Olson, "HyStart++: Modified Slow Start for TCP", Work in Progress, Internet-Draft, draft-ietf-tcpm-hystartplusplus-04, 23 January 2022, <<https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-hystartplusplus-04>>.

- [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/rfc/rfc2119>>.
- [RFC2883] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, DOI 10.17487/RFC2883, July 2000, <<https://www.rfc-editor.org/rfc/rfc2883>>.
- [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/rfc/rfc3168>>.
- [RFC4015] Ludwig, R. and A. Gurtov, "The Eifel Response Algorithm for TCP", RFC 4015, DOI 10.17487/RFC4015, February 2005, <<https://www.rfc-editor.org/rfc/rfc4015>>.
- [RFC5033] Floyd, S. and M. Allman, "Specifying New Congestion Control Algorithms", BCP 133, RFC 5033, DOI 10.17487/RFC5033, August 2007, <<https://www.rfc-editor.org/rfc/rfc5033>>.
- [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <<https://www.rfc-editor.org/rfc/rfc5348>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/rfc/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/rfc/rfc5682>>.
- [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/rfc/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/rfc/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/rfc/rfc6675>>.
- [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/rfc/rfc7567>>.
- [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/rfc/rfc8174>>.
- [RFC8985] Cheng, Y., Cardwell, N., Dukkkipati, N., and P. Jha, "The RACK-TLP Loss Detection Algorithm for TCP", RFC 8985, DOI 10.17487/RFC8985, February 2021, <<https://www.rfc-editor.org/rfc/rfc8985>>.
- [RFC9002] Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", RFC 9002, DOI 10.17487/RFC9002, May 2021, <<https://www.rfc-editor.org/rfc/rfc9002>>.

## 8.2. Informative References

- [BSCLU13] Belhareth, S., Sassatelli, L., Collange, D., Lopez-Pacheco, D., and G. Urvoy-Keller, "Understanding TCP cubic performance in the cloud: A mean-field approach", 2013 IEEE 2nd International Conference on Cloud Networking (CloudNet), DOI 10.1109/cloudnet.2013.6710576, November 2013, <<https://doi.org/10.1109/cloudnet.2013.6710576>>.
- [CEHRX09] Cai, H., Eun, D., Ha, S., Rhee, I., and L. Xu, "Stochastic convex ordering for multiplicative decrease internet congestion control", Computer Networks Vol. 53, pp. 365-381, DOI 10.1016/j.comnet.2008.10.012, February 2009, <<https://doi.org/10.1016/j.comnet.2008.10.012>>.
- [FHP00] Floyd, S., Handley, M., and J. Padhye, "A Comparison of Equation-Based and AIMD Congestion Control", May 2000, <<https://www.icir.org/tfrc/aimd.pdf>>.

- [GV02] Gorinsky, S. and H. Vin, "Extended Analysis of Binary Adjustment Algorithms", Technical Report TR2002-29, Department of Computer Sciences, The University of Texas at Austin, 11 August 2002, <<https://www.cs.utexas.edu/ftp/techreports/tr02-39.ps.gz>>.
- [H16] Sangtae Ha, "Simulation, Testbed, and Deployment Testing Results of CUBIC", 3 November 2016, <[https://web.archive.org/web/20161118125842/http://netsrv.csc.ncsu.edu/wiki/index.php/TCP\\_Testing](https://web.archive.org/web/20161118125842/http://netsrv.csc.ncsu.edu/wiki/index.php/TCP_Testing)>.
- [HLRX07] Ha, S., Le, L., Rhee, I., and L. Xu, "Impact of background traffic on performance of high-speed TCP variant protocols", Computer Networks Vol. 51, pp. 1748-1762, DOI 10.1016/j.comnet.2006.11.005, May 2007, <<https://doi.org/10.1016/j.comnet.2006.11.005>>.
- [HR11] Ha, S. and I. Rhee, "Taming the elephants: New TCP slow start", Computer Networks Vol. 55, pp. 2092-2110, DOI 10.1016/j.comnet.2011.01.014, June 2011, <<https://doi.org/10.1016/j.comnet.2011.01.014>>.
- [HRX08] Ha, S., Rhee, I., and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant", ACM SIGOPS Operating Systems Review Vol. 42, pp. 64-74, DOI 10.1145/1400097.1400105, July 2008, <<https://doi.org/10.1145/1400097.1400105>>.
- [K03] Kelly, T., "Scalable TCP: improving performance in highspeed wide area networks", ACM SIGCOMM Computer Communication Review Vol. 33, pp. 83-91, DOI 10.1145/956981.956989, April 2003, <<https://doi.org/10.1145/956981.956989>>.
- [LBEWK16] Lukaseder, T., Bradatsch, L., Erb, B., Van Der Heijden, R., and F. Kargl, "A Comparison of TCP Congestion Control Algorithms in 10G Networks", 2016 IEEE 41st Conference on Local Computer Networks (LCN), DOI 10.1109/lcn.2016.121, November 2016, <<https://doi.org/10.1109/lcn.2016.121>>.
- [LIU16] Liu, K. and J. Lee, "On Improving TCP Performance over Mobile Data Networks", IEEE Transactions on Mobile Computing Vol. 15, pp. 2522-2536, DOI 10.1109/tmc.2015.2500227, October 2016, <<https://doi.org/10.1109/tmc.2015.2500227>>.
- [RFC3465] Allman, M., "TCP Congestion Control with Appropriate Byte Counting (ABC)", RFC 3465, DOI 10.17487/RFC3465, February 2003, <<https://www.rfc-editor.org/rfc/rfc3465>>.

- [RFC3522] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC 3522, DOI 10.17487/RFC3522, April 2003, <<https://www.rfc-editor.org/rfc/rfc3522>>.
- [RFC3649] Floyd, S., "HighSpeed TCP for Large Congestion Windows", RFC 3649, DOI 10.17487/RFC3649, December 2003, <<https://www.rfc-editor.org/rfc/rfc3649>>.
- [RFC3708] Blanton, E. and M. Allman, "Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions", RFC 3708, DOI 10.17487/RFC3708, February 2004, <<https://www.rfc-editor.org/rfc/rfc3708>>.
- [RFC3742] Floyd, S., "Limited Slow-Start for TCP with Large Congestion Windows", RFC 3742, DOI 10.17487/RFC3742, March 2004, <<https://www.rfc-editor.org/rfc/rfc3742>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/rfc/rfc4960>>.
- [RFC6937] Mathis, M., Dukkkipati, N., and Y. Cheng, "Proportional Rate Reduction for TCP", RFC 6937, DOI 10.17487/RFC6937, May 2013, <<https://www.rfc-editor.org/rfc/rfc6937>>.
- [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/rfc/rfc7661>>.
- [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/rfc/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/rfc/rfc8511>>.
- [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/rfc/rfc9000>>.

- [SXEZ19] Sun, W., Xu, L., Elbaum, S., and D. Zhao, "Model-Agnostic and Efficient Exploration of Numerical Congestion Control State Space of Real-World TCP Implementations", IEEE/ACM Transactions on Networking Vol. 29, pp. 1990–2004, DOI 10.1109/tnet.2021.3078161, October 2021, <<https://doi.org/10.1109/tnet.2021.3078161>>.
- [XHR04] Xu, L., Harfoush, K., and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks", IEEE INFOCOM 2004, DOI 10.1109/infcom.2004.1354672, n.d., <<https://doi.org/10.1109/infcom.2004.1354672>>.

#### Appendix A. Acknowledgments

Richard Scheffenegger and Alexander Zimmermann originally co-authored [RFC8312].

These individuals suggested improvements to this document:

- \* Bob Briscoe
- \* Christian Huitema
- \* Gorrry Fairhurst
- \* Jonathan Morton
- \* Juhamatti Kuusisaari
- \* Junho Choi
- \* Markku Kojo
- \* Martin Thomson
- \* Matt Mathis
- \* Matt Olson
- \* Michael Welzl
- \* Mirja Kuehlewind
- \* Mohit P. Tahiliani
- \* Neal Cardwell
- \* Praveen Balasubramanian

- \* Randall Stewart
- \* Richard Scheffenegger
- \* Rod Grimes
- \* Tom Henderson
- \* Tom Petch
- \* Wesley Rosenblum
- \* Yoshifumi Nishida
- \* Yuchung Cheng

#### Appendix B. Evolution of CUBIC

##### B.1. Since draft-ietf-tcpm-rfc8312bis-06

- \* RFC7661 is safe even when cwnd grows beyond rwnd (#143 (<https://github.com/NTAP/rfc8312bis/issues/143>))

##### B.2. Since draft-ietf-tcpm-rfc8312bis-05

- \* Clarify meaning of "application-limited" in Section 5.8 (#137 (<https://github.com/NTAP/rfc8312bis/issues/137>))
- \* Create new subsections for spurious timeouts and spurious loss via ACK (#90 (<https://github.com/NTAP/rfc8312bis/issues/90>))
- \* Brief discussion of convergence in Section 5.6 (#96 (<https://github.com/NTAP/rfc8312bis/issues/96>))
- \* Add more test results to Section 5 and update some references (#91 (<https://github.com/NTAP/rfc8312bis/issues/91>))
- \* Change wording around setting ssthresh (#131 (<https://github.com/NTAP/rfc8312bis/issues/131>))

##### B.3. Since draft-ietf-tcpm-rfc8312bis-04

- \* Fix incorrect math (#106 (<https://github.com/NTAP/rfc8312bis/issues/106>))
- \* Update RFC5681 (#99 (<https://github.com/NTAP/rfc8312bis/issues/99>))

- \* Rephrase text around algorithmic alternatives, add HyStart++ (#85 (<https://github.com/NTAP/rfc8312bis/issues/85>), #86 (<https://github.com/NTAP/rfc8312bis/issues/86>), #90 (<https://github.com/NTAP/rfc8312bis/issues/90>))
  - \* Clarify what we mean by "new ACK" and use it in the text in more places. (#101 (<https://github.com/NTAP/rfc8312bis/issues/101>))
  - \* Rewrite the Responses to Sudden or Transient Events section (#98 (<https://github.com/NTAP/rfc8312bis/issues/98>))
  - \* Remove confusing text about `_cwnd_start_` in Section 4.2 (#100 (<https://github.com/NTAP/rfc8312bis/issues/100>))
  - \* Change terminology from "AIMD" to "Reno" (#108 (<https://github.com/NTAP/rfc8312bis/issues/108>))
  - \* Moved MUST NOT from app-limited section to main cubic AI section (#97 (<https://github.com/NTAP/rfc8312bis/issues/97>))
  - \* Clarify cwnd decrease during multiplicative decrease (#102 (<https://github.com/NTAP/rfc8312bis/issues/102>))
  - \* Clarify text around queuing and slow adaptation of CUBIC in wireless environments (#94 (<https://github.com/NTAP/rfc8312bis/issues/94>))
  - \* Set lower bound of cwnd to 1 MSS and use retransmit timer thereafter (#83 (<https://github.com/NTAP/rfc8312bis/issues/83>))
  - \* Use FlightSize instead of cwnd to update ssthresh (#114 (<https://github.com/NTAP/rfc8312bis/issues/114>))
- B.4. Since draft-ietf-tcpm-rfc8312bis-03
- \* Remove reference from abstract (#82 (<https://github.com/NTAP/rfc8312bis/pull/82>))
- B.5. Since draft-ietf-tcpm-rfc8312bis-02
- \* Description of packet loss rate `_p_` (#65 (<https://github.com/NTAP/rfc8312bis/issues/65>))
  - \* Clarification of TCP Friendly Equation for ABC and Delayed ACK (#66 (<https://github.com/NTAP/rfc8312bis/issues/66>))
  - \* add applicability to QUIC and SCTP (#61 (<https://github.com/NTAP/rfc8312bis/issues/61>))

- \* clarity on setting `alpha__aimd_` to 1 (#68 (<https://github.com/NTAP/rfc8312bis/issues/68>))
  - \* introduce `alpha__cubic_` (#64 (<https://github.com/NTAP/rfc8312bis/issues/64>))
  - \* clarify `_cwnd_` growth in convex region (#69 (<https://github.com/NTAP/rfc8312bis/issues/69>))
  - \* add guidance for using bytes and mention that segments count is decimal (#67 (<https://github.com/NTAP/rfc8312bis/issues/67>))
  - \* add loss events detected by RACK and QUIC loss detection (#62 (<https://github.com/NTAP/rfc8312bis/issues/62>))
- B.6. Since draft-ietf-tcpm-rfc8312bis-01
- \* address Michael Scharf's editorial suggestions. (#59 (<https://github.com/NTAP/rfc8312bis/issues/59>))
  - \* add "Note to the RFC Editor" about removing underscores
- B.7. Since draft-ietf-tcpm-rfc8312bis-00
- \* use updated xml2rfc with better text rendering of subscripts
- B.8. Since draft-eggert-tcpm-rfc8312bis-03
- \* fix spelling nits
  - \* rename to draft-ietf
  - \* define `_W_max_` more clearly
- B.9. Since draft-eggert-tcpm-rfc8312bis-02
- \* add definition for `segments_acked` and `alpha__aimd_`. (#47 (<https://github.com/NTAP/rfc8312bis/issues/47>))
  - \* fix a mistake in `_W_max_` calculation in the fast convergence section. (#51 (<https://github.com/NTAP/rfc8312bis/issues/51>))
  - \* clarity on setting `_ssthresh_` and `_cwnd_start_` during multiplicative decrease. (#53 (<https://github.com/NTAP/rfc8312bis/issues/53>))
- B.10. Since draft-eggert-tcpm-rfc8312bis-01

- \* rename TCP-Friendly to AIMD-Friendly and rename Standard TCP to AIMD TCP to avoid confusion as CUBIC has been widely used on the Internet. (#38 (<https://github.com/NTAP/rfc8312bis/issues/38>))
- \* change introductory text to reflect the significant broader deployment of CUBIC on the Internet. (#39 (<https://github.com/NTAP/rfc8312bis/issues/39>))
- \* rephrase introduction to avoid referring to variables that have not been defined yet.

B.11. Since draft-eggert-tcpm-rfc8312bis-00

- \* acknowledge former co-authors (#15 (<https://github.com/NTAP/rfc8312bis/issues/15>))
- \* prevent `_cwnd_` from becoming less than two (#7 (<https://github.com/NTAP/rfc8312bis/issues/7>))
- \* add list of variables and constants (#5 (<https://github.com/NTAP/rfc8312bis/issues/5>), #6 (<https://github.com/NTAP/rfc8312bis/issues/6>))
- \* update `_K_`'s definition and add bounds for CUBIC `_target_ _cwnd_` [SXEZ19] (#1 (<https://github.com/NTAP/rfc8312bis/issues/1>), #14 (<https://github.com/NTAP/rfc8312bis/issues/14>))
- \* update `_W_est_` to use AIMD approach (#20 (<https://github.com/NTAP/rfc8312bis/issues/20>))
- \* set `alpha__aimd_` to 1 once `_W_est_` reaches `_W_max_` (#2 (<https://github.com/NTAP/rfc8312bis/issues/2>))
- \* add Vidhi as co-author (#17 (<https://github.com/NTAP/rfc8312bis/issues/17>))
- \* note for Fast Recovery during `_cwnd_` decrease due to congestion event (#11 (<https://github.com/NTAP/rfc8312bis/issues/11>))
- \* add section for spurious congestion events (#23 (<https://github.com/NTAP/rfc8312bis/issues/23>))
- \* initialize `_W_est_` after timeout and remove variable `_W_(last_max)_` (#28 (<https://github.com/NTAP/rfc8312bis/issues/28>))

B.12. Since RFC8312

- \* converted to Markdown and xml2rfc v3
- \* updated references (as part of the conversion)
- \* updated author information
- \* various formatting changes
- \* move to Standards Track

#### B.13. Since the Original Paper

CUBIC has gone through a few changes since the initial release [HRX08] of its algorithm and implementation. Below we highlight the differences between its original paper and [RFC8312].

- \* The original paper [HRX08] includes the pseudocode of CUBIC implementation using Linux's pluggable congestion control framework, which excludes system-specific optimizations. The simplified pseudocode might be a good source to start with and understand CUBIC.
- \* [HRX08] also includes experimental results showing its performance and fairness.
- \* The definition of `beta__cubic_` constant was changed in [RFC8312]. For example, `beta__cubic_` in the original paper was the window decrease constant while [RFC8312] changed it to CUBIC multiplication decrease factor. With this change, the current congestion window size after a congestion event in [RFC8312] was `beta__cubic_ * _W_max_` while it was `(1-beta__cubic_) * _W_max_` in the original paper.
- \* Its pseudocode used `_W_(last_max)_` while [RFC8312] used `_W_max_`.
- \* Its AIMD-friendly window was `_W_tcp_` while [RFC8312] used `_W_est_`.

#### Authors' Addresses

Lisong Xu  
University of Nebraska-Lincoln  
Department of Computer Science and Engineering  
Lincoln, NE 68588-0115  
United States of America  
Email: [xu@unl.edu](mailto:xu@unl.edu)  
URI: <https://cse.unl.edu/~xu/>

Sangtae Ha  
University of Colorado at Boulder  
Department of Computer Science  
Boulder, CO 80309-0430  
United States of America  
Email: sangtae.ha@colorado.edu  
URI: <https://netstech.org/sangtaeha/>

Injong Rhee  
Bowery Farming  
151 W 26TH Street, 12TH Floor  
New York, NY 10001  
United States of America  
Email: injongrhee@gmail.com

Vidhi Goel  
Apple Inc.  
One Apple Park Way  
Cupertino, California 95014  
United States of America  
Email: vidhi\_goel@apple.com

Lars Eggert (editor)  
NetApp  
Stenbergintie 12 B  
FI-02700 Kauniainen  
Finland  
Email: lars@eggert.org  
URI: <https://eggert.org/>

TCPM  
Internet-Draft  
Intended status: Standards Track  
Expires: 9 January 2022

M. Scharf  
Hochschule Esslingen  
V. Murgai  
Samsung  
M. Jethanandani  
Kloud Services  
8 July 2021

YANG Model for Transmission Control Protocol (TCP) Configuration  
draft-ietf-tcpm-yang-tcp-02

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 9 January 2022.

Copyright Notice

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

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

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

## Table of Contents

1. Introduction . . . . .	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 . . . . .	6
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 . . . . .	14
7.1. Normative References . . . . .	14
7.2. Informative References . . . . .	16
Appendix A. Acknowledgements . . . . .	17
Appendix B. Changes compared to previous versions . . . . .	17
Appendix C. Examples . . . . .	18
C.1. Keepalive Configuration . . . . .	18
C.2. TCP-AO Configuration . . . . .	19
Appendix D. Complete Tree Diagram . . . . .	20
Authors' Addresses . . . . .	21

## 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 orthogonal to this specification. Examples are:

- \* 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 [RFC8783].
- \* 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.

2021-07-06 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:

- \* Global configuration: Many TCP implementations have configuration parameters that affect all TCP connections. Typical examples include enabling or disabling optional protocol features.
- \* 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.
- \* 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].
- \* 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. The model includes the required TCP parameters for TCP-AO configuration. The key chain for TCP-AO can be modeled by the YANG Data Model for Key Chains [RFC8177].

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.

- \* Statistics: Counters for the number of active/passive opens, sent and received segments, errors, and possibly other detailed debugging information
- \* 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. Note that the TCP MIB does not include means to reset statistics, which are defined in this document. This is not a major addition, as a reset can simply be implemented by storing offset values for the counters.

### 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
    |   ...
    +--ro statistics {statistics}?
    |   ...
```

## 4. TCP YANG Model

```
<CODE BEGINS> file "ietf-tcp@2021-07-06.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-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](mailto:tcpm@ietf.org)>

Authors: Michael Scharf ([michael.scharf at hs-esslingen dot de](mailto:michael.scharf@hs-esslingen.de))

Vishal Murgai ([vmurgai at gmail dot com](mailto:vmurgai@gmail.com))

Mahesh Jethanandani ([mjethanandani at gmail dot com](mailto:mjethanandani@gmail.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.

Copyright (c) 2020 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX (<https://www.rfc-editor.org/info/rfcXXXX>); see the RFC itself for full legal notices.

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 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here.";

```
revision "2021-07-06" {  
  description  
    "Initial Version";  
  reference  
    "RFC XXXX, TCP Configuration.";  
}
```

```
// Features  
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. The SendID must match the RecvID
      at the other endpoint.";
    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. The RecvID must match the SendID at the other
      endpoint.";
    reference
      "RFC 5925: The TCP Authentication Option.";
  }

  leaf include-tcp-options {
    type boolean;
    must "../enable-ao = 'true'";
    default true;
    description
      "Include TCP options in MAC calculation.";
  }

  leaf accept-key-mismatch {
    type boolean;
    must "../enable-ao = 'true'";
    description
      "Accept TCP segments with a Master Key Tuple (MKT) that is not
      configured.";
```

```
    }
    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
  "List of TCP connections with their parameters. The list
  is modelled as writeable, but implementations may not allow
  creation of new TCP connections by adding entries to the
  list. Furthermore, the behavior upon removal is
  implementation-specific. Implementations may support
  closing or resetting a TCP connection upon an operation that
  removes the entry from the list.";
}
description
  "A container of all TCP connections.";
```

```
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.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., "config true", which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

#### TODO

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

- \* Unrestricted access to connection information of the client or server can be used by a malicious user to launch an attack, e.g. MITM.
- \* Similarly, unrestricted access to statistics of the client or server can be used by a malicious user to exploit any vulnerabilities of the system.

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

- \* The YANG module allows for the statistics to be cleared by executing the reset action. This action should be restricted to users with the right permission.

## 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", Work in Progress, Internet-Draft, draft-ietf-netconf-tcp-client-server-09, 10 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-netconf-tcp-client-server-09.txt>>.
- [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>>.
- [RFC8177] Lindem, A., Ed., Qu, Y., Yeung, D., Chen, I., and J. Zhang, "YANG Data Model for Key Chains", RFC 8177, DOI 10.17487/RFC8177, June 2017, <<https://www.rfc-editor.org/info/rfc8177>>.

- [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-idr-bgp-model]  
Jethanandani, M., Patel, K., Hares, S., and J. Haas, "BGP YANG Model for Service Provider Networks", Work in Progress, Internet-Draft, draft-ietf-idr-bgp-model-10, 15 November 2020, <<https://www.ietf.org/archive/id/draft-ietf-idr-bgp-model-10.txt>>.
- [I-D.ietf-taps-interface]  
Trammell, B., Welzl, M., Enghardt, T., Fairhurst, G., Kuehlewind, M., Perkins, C., Tiesel, P. S., Wood, C. A., Pauly, T., and K. Rose, "An Abstract Application Layer Interface to Transport Services", Work in Progress, Internet-Draft, draft-ietf-taps-interface-12, 9 April 2021, <<https://www.ietf.org/archive/id/draft-ietf-taps-interface-12.txt>>.
- [I-D.touch-tcpm-ao-test-vectors]  
Touch, J. and J. Kuusisaari, "TCP-AO Test Vectors", Work in Progress, Internet-Draft, draft-touch-tcpm-ao-test-vectors-02, 23 December 2020, <<https://www.ietf.org/archive/id/draft-touch-tcpm-ao-test-vectors-02.txt>>.
- [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>>.
- [RFC8783] Boucadair, M., Ed. and T. Reddy.K, Ed., "Distributed Denial-of-Service Open Threat Signaling (DOTS) Data Channel Specification", RFC 8783, DOI 10.17487/RFC8783, May 2020, <<https://www.rfc-editor.org/info/rfc8783>>.

#### 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

- \* Removed congestion control
- \* Removed global stack parameters

Changes compared to draft-scharf-tcpm-yang-tcp-03

- \* Updated TCP-AO grouping
- \* Added congestion control

Changes compared to draft-scharf-tcpm-yang-tcp-02

- \* Initial proposal of a YANG model including base configuration parameters, TCP-AO configuration, and a connection list
- \* Editorial bugfixes and outdated references reported by Mohamed Boucadair
- \* Additional co-author Mahesh Jethanandani

Changes compared to draft-scharf-tcpm-yang-tcp-01

- \* Alignment with [I-D.ietf-netconf-tcp-client-server]
- \* Removing backward-compatibility to the TCP MIB
- \* Additional co-author Vishal Murgai

Changes compared to draft-scharf-tcpm-yang-tcp-00

- \* Editorial improvements

## Appendix C. Examples

### C.1. Keepalive Configuration

This particular example demonstrates how both a particular connection can be configured for keepalives.

[note: '\ ' line wrapping for formatting only]

```
<?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).
-->
<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>
</tcp>
```

## C.2. TCP-AO Configuration

The following example demonstrates how to model a TCP-AO [RFC5925] configuration for the example in TCP-AO Test Vectors [I-D.touch-tcpm-ao-test-vectors], Section 5.1.1.

[note: '\' line wrapping for formatting only]

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
This example sets TCP-AO configuration parameters as
demonstrated by examples in draft-touch-tcpm-ao-test-vectors.
-->

<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>
        <enable-ao>true</enable-ao>
      </common>
    </connection>
  </connections>
</tcp>

<key-chains
  xmlns="urn:ietf:params:xml:ns:yang:ietf-key-chain">
  <key-chain>
    <name>ao-config</name>
    <description>"An example for TCP-AO configuration."</description>

    <key>
      <key-id>61</key-id>
      <crypto-algorithm>hmac-sha-1</crypto-algorithm>
      <key-string>
        <hexadecimal-string>01:23:a5:93:b9:db:70:62:9b:be:2c:a6:77:cd:fd:ea:\
6f:e0:ac:ad</hexadecimal-string>
      </key-string>
    </key>
  </key-chain>
</key-chains>
```

#### Appendix D. Complete Tree Diagram

Here is the complete tree diagram for the TCP YANG model.

```

module: ietf-tcp
+--rw tcp!
  +--rw connections
    +--rw connection*
      [local-address remote-address local-port remote-port]
      +--rw local-address      inet:ip-address
      +--rw remote-address     inet:ip-address
      +--rw local-port         inet:port-number
      +--rw remote-port        inet:port-number
      +--rw common
        +--rw keepalives!
          +--rw idle-time      uint16
          +--rw max-probes      uint16
          +--rw probe-interval uint16
        +--rw (authentication)?
          +--:(ao)
            +--rw enable-ao?    boolean
            +--rw send-id?      uint8
            +--rw rcv-id?       uint8
            +--rw include-tcp-options? boolean
            +--rw accept-key-mismatch? boolean
          +--:(md5)
            +--rw enable-md5?    boolean
      +--ro statistics {statistics}?
        +--ro active-opens?      yang:counter32
        +--ro passive-opens?     yang:counter32
        +--ro attempt-fails?     yang:counter32
        +--ro establish-resets?   yang:counter32
        +--ro currently-established? yang:gauge32
        +--ro in-segments?       yang:counter64
        +--ro out-segments?      yang:counter64
        +--ro retransmitted-segments? yang:counter32
        +--ro in-errors?         yang:counter32
        +--ro out-resets?        yang:counter32
      +---x reset
        +---w input
          | +---w reset-at?    yang:date-and-time
        +--ro output
          +--ro reset-finished-at? yang:date-and-time

```

#### Authors' Addresses

Michael Scharf  
 Hochschule Esslingen - University of Applied Sciences  
 Flandernstr. 101  
 73732 Esslingen  
 Germany

Email: michael.scharf@hs-esslingen.de

Vishal Murgai  
Samsung

Email: vmurgai@gmail.com

Mahesh Jethanandani  
Kloud Services

Email: mjethanandani@gmail.com

TCPM  
Internet-Draft  
Intended status: Standards Track  
Expires: 7 August 2022

M. Scharf  
Hochschule Esslingen  
M. Jethanandani  
Kloud Services  
V. Murgai  
Samsung  
3 February 2022

A YANG Model for Transmission Control Protocol (TCP) Configuration  
draft-ietf-tcpm-yang-tcp-06

Abstract

This document specifies a minimal 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 authentication parameters that can be imported and used in TCP implementations or by other models that need to configure TCP parameters. The model also includes basic TCP statistics. The model is compliant with Network Management Datastore Architecture (NMDA) (RFC 8342).

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 7 August 2022.

Copyright Notice

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

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

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

## Table of Contents

1. Introduction . . . . .	2
2. Requirements Language . . . . .	4
2.1. Note to RFC Editor . . . . .	4
3. YANG Module Overview . . . . .	4
3.1. Scope . . . . .	4
3.2. Model Design . . . . .	6
3.3. Tree Diagram . . . . .	6
4. TCP YANG Model . . . . .	6
5. IANA Considerations . . . . .	14
5.1. The IETF XML Registry . . . . .	14
5.2. The YANG Module Names Registry . . . . .	15
6. Security Considerations . . . . .	15
7. References . . . . .	16
7.1. Normative References . . . . .	16
7.2. Informative References . . . . .	18
Appendix A. Acknowledgements . . . . .	20
Appendix B. Examples . . . . .	20
B.1. Keepalive Configuration . . . . .	20
B.2. TCP-AO Configuration . . . . .	21
Appendix C. Complete Tree Diagram . . . . .	23
Authors' Addresses . . . . .	23

## 1. Introduction

The Transmission Control Protocol (TCP) [I-D.ietf-tcpm-rfc793bis] is used by many applications in the Internet, including control and management protocols. As such, 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 minimal YANG 1.1 [RFC7950] model for configuring TCP on network elements that support YANG. This YANG module is compliant with Network Management Datastore Architecture (NMDA) [RFC8342].

The YANG module has a narrow scope and focuses on a subset of fundamental TCP functions and basic statistics. It defines a container for TCP connection that includes definitions from YANG Groupings for TCP Clients and TCP Servers [I-D.ietf-netconf-tcp-client-server]. This model adheres to the

recommendation in BGP/MPLS IP Virtual Private Networks [RFC4364] as it allows enabling of TCP-AO [RFC5925], and accommodates the installed base that makes use of MD5. The module can be augmented or updated to address more advanced or implementation-specific TCP features in the future.

Many protocol stacks on IP 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. Such other means 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. The YANG module also omits some selected parameters included in TCP MIB, most notably the configured Retransmission Timeout (RTO) algorithm. This is conscious decision as these parameters hardly matter in a state-of-the-art TCP implementation. It would also be possible also to translate a MIB into a YANG module, 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, because a translated model would not be up-to-date.

There are other existing TCP-related YANG models, which are orthogonal to this specification. Examples are:

- \* TCP header attributes are modeled in other security-related models, such as YANG Data Model for Network Access Control Lists (ACLs) [RFC8519], Distributed Denial-of-Service Open Thread Signaling (DOTS) Data Channel Specification [RFC8783], or I2NSF Capability YANG Data Model [I-D.ietf-i2nsf-capability-data-model].
- \* 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].
- \* TCP-AO and TCP MD5 configuration for Layer 3 VPNs is modeled in A Layer 3 VPN Network YANG Model [I-D.ietf-opsawg-l3sm-l3nm]. This model assumes that TCP-AO specific parameters are preconfigured in addition to the keychain parameters. This issue is further discussed below.

## 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.

2022-02-04 with the actual date of the publication of this document.

## 3. YANG Module Overview

### 3.1. Scope

TCP is implemented on different system architectures. As a result, there are many different and often implementation-specific ways to configure parameters of the TCP engine. In addition, in many TCP/IP stacks configuration exists for different scopes:

- \* Global configuration: Many TCP implementations have configuration parameters that affect all TCP connections. Typical examples include enabling or disabling optional protocol features.
- \* 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.
- \* Connection parameters: Many implementations have means to influence the behavior of each TCP connection, e.g., on the programming interface used by applications. Typical examples 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].

- \* 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 implementation 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.

This in particular applies to the support of TCP-AO [RFC5925]. TCP Authentication Option (TCP-AO) is used on routers to secure routing protocols such as BGP. In that case, a YANG model for TCP-AO configuration is required. The model defined in this document includes the required parameters for TCP-AO configuration, such as the values of SendID and RecvID. The keychain for TCP-AO can be modeled by the YANG Data Model for Key Chains [RFC8177]. The groupings defined in this document can be imported and used as part of such a preconfiguration.

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.

- \* Statistics: Counters for the number of active/passive opens, sent and received segments, errors, and possibly other detailed debugging information

- \* TCP connection table: Access to status information for all TCP connections. Note, the connection table is modeled as a list that is read-writeable, even though a connection cannot be created by adding entries to the table. Similarly, deletion of connections from this list is implementation-specific.

This allows implementations of TCP MIB [RFC4022] to migrate to the YANG model defined in this memo. Note that the TCP MIB does not include means to reset statistics, which are defined in this document. This is not a major addition, as a reset can simply be implemented by storing offset values for the counters.

This version of the module does not cover Multipath TCP [RFC8684].

### 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 an 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
    |   ...
    +--ro statistics {statistics}?
    |   ...
    ...
```

## 4. TCP YANG Model

This YANG module references The TCP Authentication Option [RFC5925], Protection of BGP Sessions via the TCP MD5 Signature [RFC2385], Transmission Control Protocol (TCP) Specification [I-D.ietf-tcpm-rfc793bis], and imports Common YANG Data Types [RFC6991], The NETCONF Access Control Model [RFC8341], and YANG Groupings for TCP Clients and TCP Servers [I-D.ietf-netconf-tcp-client-server].

```
<CODE BEGINS> file "ietf-tcp@2022-02-04.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-common {
    prefix "tcpcmn";
    reference
      "I-D.ietf-netconf-tcp-client-server: YANG Groupings for TCP
      Clients and TCP Servers.";
  }
  import ietf-inet-types {
    prefix "inet";
    reference
      "RFC 6991: Common YANG Data Types.";
  }
  import ietf-netconf-acm {
    prefix nacm;
    reference
      "RFC 8341: Network Configuration Access Control Model";
  }

  organization
    "IETF TCPM Working Group";

  contact
    "WG Web:  <https://datatracker.ietf.org/wg/tcpm/about>
    WG List:  <tcpm@ietf.org>

    Authors: Michael Scharf (michael.scharf at hs-esslingen dot de)
             Mahesh Jethanandani (mjethanandani at gmail dot com)
             Vishal Murgai (vmurgai at gmail dot com)";

  description
    "This module focuses on fundamental TCP functions and basic
    statistics. The model can be augmented to address more advanced
    or implementation specific TCP features.

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

    Redistribution and use in source and binary forms, with or
```

without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC XXXX (<https://www.rfc-editor.org/info/rfcXXXX>); see the RFC itself for full legal notices.

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 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here.";

```
revision "2022-02-04" {
  description
    "Initial Version";
  reference
    "RFC XXXX, A YANG Model for Transmission Control Protocol (TCP)
      Configuration.";
}

// Features
feature statistics {
  description
    "This implementation supports statistics reporting.";
}

// TCP-AO Groupings
grouping ao {
  leaf enable-ao {
    type boolean;
    default "false";
    description
      "When set to true, TCP-Authentication Option (TCP-AO) is
        enabled.";
  }

  leaf send-id {
    type uint8 {
      range "0..max";
    }
    must "../enable-ao = 'true'";
    description
      "The SendID is inserted as the KeyID of the TCP-AO option
```

```
        of outgoing segments. The SendID must match the RecvID
        at the other endpoint.";
    reference
        "RFC 5925: The TCP Authentication Option, Section 3.1.";
}

leaf recv-id {
    type uint8 {
        range "0..max";
    }
    must "../enable-ao = 'true'";
    description
        "The RecvID is matched against the TCP-AO KeyID of incoming
        segments. The RecvID must match the SendID at the other
        endpoint.";
    reference
        "RFC 5925: The TCP Authentication Option, Section 3.1.";
}

leaf include-tcp-options {
    type boolean;
    must "../enable-ao = 'true'";
    default true;
    description
        "When set to true, TCP options are included in MAC
        calculation.";
    reference
        "RFC 5925: The TCP Authentication Option, Section 3.1.";
}

leaf accept-key-mismatch {
    type boolean;
    must "../enable-ao = 'true'";
    description
        "Accept, when set to true, TCP segments with a Master Key
        Tuple (MKT) that is not configured.";
    reference
        "RFC 5925: The TCP Authentication Option, Section 7.3.";
}
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
    "Enables, when set to true, support of MD5 to authenticate a
    TCP session. As the TCP MD5 signature option is obsoleted by
    TCP-AO, it is strongly RECOMMENDED to use TCP-AO instead.";
}

// 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
          "Identifies the address that is used by the local
          endpoint for the connection, and is one of the four
          elements that form the connection identifier.";
      }

      leaf remote-address {
        type inet:ip-address;
        description
          "Identifies the address that is used by the remote
          endpoint for the connection, and is one of the four
          elements that form the connection identifier.";
      }

      leaf local-port {
        type inet:port-number;
        description
          "Identifies the local TCP port used for the connection,
```

```
        and is one of the four elements that form the
        connection identifier.";
    }

    leaf remote-port {
        type inet:port-number;
        description
            "Identifies the remote TCP port used for the connection,
            and is one of the four elements that form 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 TCP authentication.";
        }
        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
        "List of TCP connections with their parameters. The list
        is modeled as writeable, but implementations may not
        allow creation of new TCP connections by adding entries to
        the list. Furthermore, the behavior upon removal is
        implementation-specific. Implementations may support
        closing or resetting a TCP connection upon an operation
        that removes the entry from the list.";
}
description
    "A container of all TCP connections.";
```

```
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.";
    reference
      "I-D.ietf-tcpm-rfc793bis: Transmission Control Protocol
       (TCP) Specification.";
  }

  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.";
    reference
      "I-D.ietf-tcpm-rfc793bis: Transmission Control Protocol
       (TCP) Specification.";
  }

  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.";
    reference
      "I-D.ietf-tcpm-rfc793bis: Transmission Control Protocol
       (TCP) Specification.";
  }

  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.";
    reference
      "I-D.ietf-tcpm-rfc793bis: Transmission Control Protocol
       (TCP) Specification.";
  }
}
```

```
leaf currently-established {
  type yang:gauge32;
  description
    "The number of TCP connections for which the current state
    is either ESTABLISHED or CLOSE-WAIT.";
  reference
    "I-D.ietf-tcpm-rfc793bis: Transmission Control Protocol
    (TCP) Specification.";
}

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.";
  reference
    "I-D.ietf-tcpm-rfc793bis: Transmission Control Protocol
    (TCP) Specification.";
}

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.";
  reference
    "I-D.ietf-tcpm-rfc793bis: Transmission Control Protocol
    (TCP) Specification.";
}

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.";
  reference
    "I-D.ietf-tcpm-rfc793bis: Transmission Control Protocol
    (TCP) Specification.";
}

leaf in-errors {
  type yang:counter32;
  description
    "The total number of segments received in error (e.g., bad
    TCP checksums).";
```

```
        reference
        "I-D.ietf-tcpm-rfc793bis: Transmission Control Protocol
        (TCP) Specification.";
    }

    leaf out-resets {
        type yang:counter32;
        description
            "The number of TCP segments sent containing the RST flag.";
        reference
            "I-D.ietf-tcpm-rfc793bis: Transmission Control Protocol
            (TCP) Specification.";
    }

    action reset {
        nacm:default-deny-all;
        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 an URI in the "ns" subregistry of the IETF XML Registry [RFC3688]. Following the format in IETF XML Registry [RFC3688], the following registration is requested:

URI: urn:ietf:params:xml:ns:yang:ietf-tcp  
Registrant Contact: The IESG.  
XML: N/A, the requested URI is an XML namespace.

## 5.2. The YANG Module Names Registry

This document registers a YANG module in the "YANG Module Names" registry YANG - A Data Modeling Language [RFC6020]. Following the format in YANG - A Data Modeling Language [RFC6020], the following registration is requested:

name:	ietf-tcp
namespace:	urn:ietf:params:xml:ns:yang:ietf-tcp
prefix:	tcp
reference:	RFC XXXX

The registration is not maintained by IANA.

## 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.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., "config true", which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

- \* Common configuration included from NETCONF Client and Server Models [I-D.ietf-netconf-tcp-client-server]. Unrestricted access to all the nodes, e.g., keepalive idle-timer, can cause connections to fail or to timeout prematurely.

- \* Authentication configuration. Unrestricted access to the nodes under authentication configuration can prevent the use of authenticated communication and cause connection setups to fail. This can result in massive security vulnerabilities and service disruption for the traffic requiring authentication.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

- \* Unrestricted access to connection information of the client or server can be used by a malicious user to launch an attack, e.g. MITM.
- \* Similarly, unrestricted access to statistics of the client or server can be used by a malicious user to exploit any vulnerabilities of the system.

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

- \* The YANG module allows for the statistics to be cleared by executing the reset action. This action should be restricted to users with the right permission.

The module specified in this document supports MD5 to basically accommodate the installed BGP base. MD5 suffers from the security weaknesses discussed in Section 2 of RFC 6151 [RFC6151] or Section 2.1 of RFC 6952 [RFC6952].

## 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", Work in Progress, Internet-Draft, draft-ietf-netconf-tcp-client-server-11, 14 December 2021, <<https://www.ietf.org/archive/id/draft-ietf-netconf-tcp-client-server-11.txt>>.

[I-D.ietf-tcpm-rfc793bis]

Eddy, W. M., "Transmission Control Protocol (TCP) Specification", Work in Progress, Internet-Draft, draft-

ietf-tcpm-rfc793bis-25, 7 September 2021,  
<<https://www.ietf.org/archive/id/draft-ietf-tcpm-rfc793bis-25.txt>>.

- [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>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [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>>.
- [RFC8177] Lindem, A., Ed., Qu, Y., Yeung, D., Chen, I., and J. Zhang, "YANG Data Model for Key Chains", RFC 8177, DOI 10.17487/RFC8177, June 2017, <<https://www.rfc-editor.org/info/rfc8177>>.
- [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-i2nsf-capability-data-model]  
Hares, S., Jeong, J. (., Kim, J. (., Moskowitz, R., and Q. Lin, "I2NSF Capability YANG Data Model", Work in Progress, Internet-Draft, draft-ietf-i2nsf-capability-data-model-22, 22 January 2022, <<https://www.ietf.org/archive/id/draft-ietf-i2nsf-capability-data-model-22.txt>>.
- [I-D.ietf-idr-bgp-model]  
Jethanandani, M., Patel, K., Hares, S., and J. Haas, "BGP YANG Model for Service Provider Networks", Work in Progress, Internet-Draft, draft-ietf-idr-bgp-model-12, 25 October 2021, <<https://www.ietf.org/archive/id/draft-ietf-idr-bgp-model-12.txt>>.

- [I-D.ietf-opsawg-l3sm-l3nm]  
Barguil, S., Dios, O. G. D., Boucadair, M., Munoz, L. A.,  
and A. Aguado, "A Layer 3 VPN Network YANG Model", Work in  
Progress, Internet-Draft, draft-ietf-opsawg-l3sm-l3nm-18,  
8 October 2021, <<https://www.ietf.org/archive/id/draft-ietf-opsawg-l3sm-l3nm-18.txt>>.
- [I-D.ietf-taps-interface]  
Trammell, B., Welzl, M., Enghardt, T., Fairhurst, G.,  
Kuehlewind, M., Perkins, C., Tiesel, P. S., Wood, C. A.,  
Pauly, T., and K. Rose, "An Abstract Application Layer  
Interface to Transport Services", Work in Progress,  
Internet-Draft, draft-ietf-taps-interface-14, 3 January  
2022, <<https://www.ietf.org/archive/id/draft-ietf-taps-interface-14.txt>>.
- [I-D.ietf-tcpm-ao-test-vectors]  
Touch, J. and J. Kuusisaari, "TCP-AO Test Vectors", Work  
in Progress, Internet-Draft, draft-ietf-tcpm-ao-test-  
vectors-06, 30 January 2022,  
<<https://www.ietf.org/archive/id/draft-ietf-tcpm-ao-test-vectors-06.txt>>.
- [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>>.
- [RFC4364] Rosen, E. and Y. Rekhter, "BGP/MPLS IP Virtual Private  
Networks (VPNs)", RFC 4364, DOI 10.17487/RFC4364, February  
2006, <<https://www.rfc-editor.org/info/rfc4364>>.
- [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>>.
- [RFC6151] Turner, S. and L. Chen, "Updated Security Considerations  
for the MD5 Message-Digest and the HMAC-MD5 Algorithms",  
RFC 6151, DOI 10.17487/RFC6151, March 2011,  
<<https://www.rfc-editor.org/info/rfc6151>>.
- [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>>.

- [RFC6952] Jethanandani, M., Patel, K., and L. Zheng, "Analysis of BGP, LDP, PCEP, and MSDP Issues According to the Keying and Authentication for Routing Protocols (KARP) Design Guide", RFC 6952, DOI 10.17487/RFC6952, May 2013, <<https://www.rfc-editor.org/info/rfc6952>>.
- [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>>.
- [RFC8684] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 8684, DOI 10.17487/RFC8684, March 2020, <<https://www.rfc-editor.org/info/rfc8684>>.
- [RFC8783] Boucadair, M., Ed. and T. Reddy.K, Ed., "Distributed Denial-of-Service Open Threat Signaling (DOTS) Data Channel Specification", RFC 8783, DOI 10.17487/RFC8783, May 2020, <<https://www.rfc-editor.org/info/rfc8783>>.

## 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, and Tom Petch.

## Appendix B. Examples

### B.1. Keepalive Configuration

This particular example demonstrates how both a particular connection can be configured for keepalives.

NOTE: '\ ' line wrapping per RFC 8792

```
<?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).
-->
<tcp
  xmlns="urn:ietf:params:xml:ns:yang:ietf-tcp">
  <connections>
    <connection>
      <local-address>192.0.2.1</local-address>
      <remote-address>192.0.2.2</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>
</tcp>
```

## B.2. TCP-AO Configuration

The following example demonstrates how to model a TCP-AO [RFC5925] configuration for the example in TCP-AO Test Vectors [I-D.ietf-tcpm-ao-test-vectors].

NOTE: '\ ' line wrapping per RFC 8792

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
This example sets TCP-AO configuration parameters as
demonstrated by examples in draft-ietf-tcpm-ao-test-vectors.
-->

<tcp
  xmlns="urn:ietf:params:xml:ns:yang:ietf-tcp">
  <connections>
    <connection>
      <local-address>fd00::1</local-address>
      <remote-address>fd00::2</remote-address>
      <local-port>1025</local-port>
      <remote-port>179</remote-port>
      <common>
        <enable-ao>true</enable-ao>
        <send-id>61</send-id>
        <recv-id>84</recv-id>
      </common>
    </connection>
  </connections>
</tcp>

<key-chains
  xmlns="urn:ietf:params:xml:ns:yang:ietf-key-chain">
  <key-chain>
    <name>ao-config</name>
    <description>"An example for TCP-AO configuration."</description>

    <key>
      <key-id>61</key-id>
      <crypto-algorithm>hmac-sha-1</crypto-algorithm>
      <key-string>
        <keystring>testvector</keystring>
      </key-string>
    </key>
    <key>
      <key-id>84</key-id>
      <crypto-algorithm>hmac-sha-1</crypto-algorithm>
      <key-string>
        <keystring>testvector</keystring>
      </key-string>
    </key>
  </key-chain>
</key-chains>
```

## Appendix C. Complete Tree Diagram

Here is the complete tree diagram for the TCP YANG model.

```

module: ietf-tcp
  +--rw tcp!
    +--rw connections
      +--rw connection*
        [local-address remote-address local-port remote-port]
        +--rw local-address      inet:ip-address
        +--rw remote-address     inet:ip-address
        +--rw local-port         inet:port-number
        +--rw remote-port        inet:port-number
      +--rw common
        +--rw keepalives!
          +--rw idle-time        uint16
          +--rw max-probes        uint16
          +--rw probe-interval    uint16
        +--rw (authentication)?
          +--:(ao)
            +--rw enable-ao?      boolean
            +--rw send-id?        uint8
            +--rw recv-id?        uint8
            +--rw include-tcp-options? boolean
            +--rw accept-key-mismatch? boolean
          +--:(md5)
            +--rw enable-md5?      boolean
      +--ro statistics {statistics}?
        +--ro active-opens?      yang:counter32
        +--ro passive-opens?     yang:counter32
        +--ro attempt-fails?     yang:counter32
        +--ro establish-resets?   yang:counter32
        +--ro currently-established? yang:gauge32
        +--ro in-segments?        yang:counter64
        +--ro out-segments?       yang:counter64
        +--ro retransmitted-segments? yang:counter32
        +--ro in-errors?          yang:counter32
        +--ro out-resets?         yang:counter32
      +---x reset
        +---w input
          | +---w reset-at?      yang:date-and-time
        +--ro output
          +--ro reset-finished-at? yang:date-and-time
  
```

Authors' Addresses

Michael Scharf  
Hochschule Esslingen - University of Applied Sciences  
Flandernstr. 101  
73732 Esslingen  
Germany

Email: michael.scharf@hs-esslingen.de

Mahesh Jethanandani  
Kloud Services

Email: mjethanandani@gmail.com

Vishal Murgai  
Samsung

Email: vmurgai@gmail.com