

TCP Maintenance and Minor Extensions  
Internet-Draft  
Intended status: Experimental  
Expires: 8 September 2022

M. Amend  
DT  
J. Kang  
Huawei  
7 March 2022

Multipath TCP Extension for Robust Session Establishment  
draft-amend-tcpm-mptcp-robe-02

Abstract

Multipath TCP extends the plain, single-path limited, TCP towards the capability of multipath transmission. This greatly improves the reliability and performance of TCP communication. For backwards compatibility reasons the Multipath TCP was designed to setup successfully an initial path first, after which subsequent paths can be added for multipath transmission. For that reason the Multipath TCP has the same limitations as the plain TCP during connection setup, in case the selected path is not functional.

This document proposes a set of implementations and possible combinations thereof, that provide a more Robust Establishment (RobE) of MPTCP sessions. It includes RobE\_TIMER, RobE\_SIM, RobE\_eSIM and RobE\_IPS.

RobE\_TIMER is designed to stay close to MPTCP in that standard functionality is used wherever possible. Resiliency against network outages is achieved by modifying the SYN retransmission timer: If one path is defective, another path is used.

RobE\_SIM and RobE\_eSIM provides the ability to simultaneously use multiple paths for connection setup. They ensure connectivity if at least one functional path out of a bunch of paths is given and offers beside that the opportunity to significantly improve loading times of Internet services.

RobE\_IPS provides a heuristic to select properly an initial path for connection establishment with a remote host based on empirical data derived from previous connection information.

In practice, these independent solutions can be complementary used. This document also presents the design and protocol procedure for those combinations in addition to the respective stand-alone solutions.

## 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 8 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

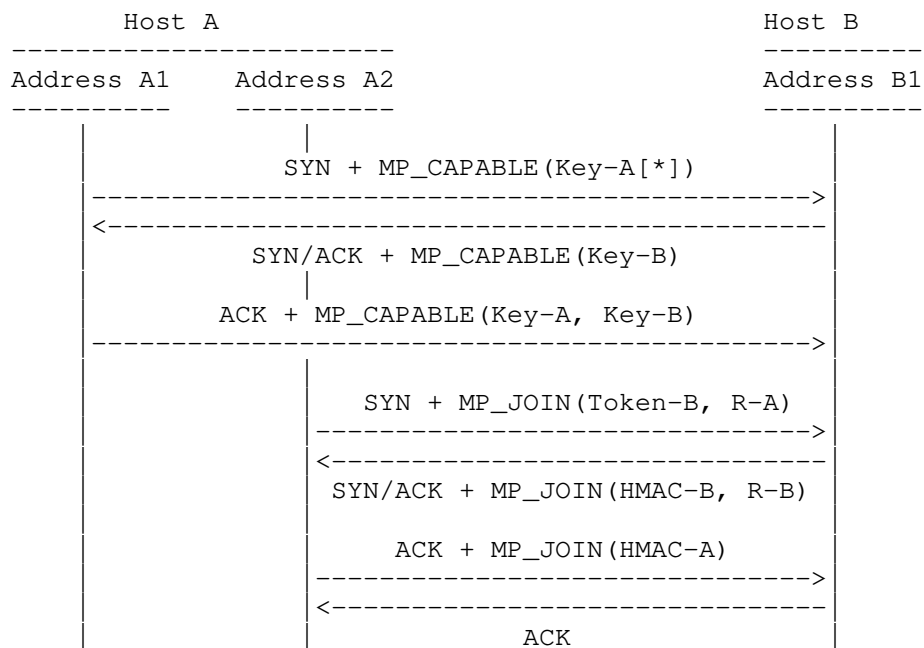
1. Introduction . . . . .	3
1.1. Terminology . . . . .	7
2. Implementation without MPTCP protocol adaptation . . . . .	8
2.1. Re-transmission Timer(RobE_TIMER) . . . . .	8
2.2. Simultaneous Initial Paths Simple Version (RobE_SIM) . . . . .	9
2.3. Heuristic Initial Path Selection (RobE_IPS) . . . . .	10
2.3.1. Architecture . . . . .	10
2.3.2. Typical Scenarios . . . . .	11
2.3.3. Path decision information . . . . .	14
2.3.4. Initial Path Selection use local RTT information . . . . .	15
2.4. Combination of RobE_SIM and RobE_IPS . . . . .	15
2.5. Combination of RobE_TIMER and RobE_IPS . . . . .	16
3. Implementation with Bi-directional MPTCP Support . . . . .	17
3.1. Simultaneous Initial Paths Extended Version (RobE_eSIM) . . . . .	18

3.1.1.	RobE_eSIM implicit Negotiation and Procedure . . . .	18
3.1.2.	RobE_eSIM explicit Negotiation and Procedure . . . .	20
3.1.3.	Protocol Adaptation . . . . .	20
3.1.4.	Fallback Mechanisms . . . . .	21
3.1.5.	Comparison Robe_SIM and RobE_eSIM . . . . .	23
3.1.6.	Security Consideration . . . . .	24
3.2.	Heuristic Initial Path Selection with remote RTT Measurement . . . . .	24
3.2.1.	Description . . . . .	24
3.2.2.	Protocol Adaptation . . . . .	25
3.2.3.	Fallback Mechanism . . . . .	26
3.2.4.	Security Consideration . . . . .	26
4.	IANA Considerations . . . . .	26
5.	References . . . . .	27
5.1.	Normative References . . . . .	27
5.2.	Informative References . . . . .	27
	Authors' Addresses . . . . .	27

## 1. Introduction

Multipath TCP Robust Session Establishment (MPTCP RobE) is a set of extensions to regular MPTCP [RFC6824] and its next version [RFC8684], which releases single path limitations during the initial connection setup. Several scenarios require and benefit from a reliable and in time connection setup which is not covered by [RFC6824] and [RFC8684] so far. MPTCP was designed to be compliant with the TCP standard [RFC0793] and introduced therefore the concept of an initial TCP flow while adding subsequent flows after successful multipath negotiation on the initial path. While fulfilling its purpose, MPTCP is however fully dependent on the transmission characteristics of the communication link selected for initiating MPTCP.

Figure 1 shows the traditional way of MPTCP handshaking with an MP\_CAPABLE exchanged first, followed when successfully negotiated by additional flows engaging MP\_JOIN. [RFC6824] and the next MPTCP [RFC8684] differ in that a Key-A is sent with the first MP\_CAPABLE or not.



[\*] Key-A in the first MP-capable is related to RFC6824 only and does not exist in RFC8684.

Figure 1: MPTCP connection setup

Multipath TCP itself enables hosts to exchange packets belonging to a single connection over several paths. Implemented in mobile phones (UEs), these paths are usually assigned to different network interfaces within the UE and correspond to different access networks such as cellular and WiFi. The path or network interface for initiating the initial subflow setup is most often provided by the operation system of the UE. For example, if both a cellular connection and WiFi are present in a mobile phone, WiFi is usually the interface offered to initiate the MPTCP session.

This design falls short in situations where the default path does not provide the best performance compared to other available paths. In a worst case the default path is not even capable of setting up the initial flow letting any other functional path unused. For example, if the WiFi signal is weak, broken or cannot forward traffic to the destination, the establishment of the subflow will be delayed or impossible. This in turn, leads to a longer startup delay or no communication at all for services using MPTCP even if other functional paths are available. Even in scenarios where all paths are functional but services would benefit from a setup over the path with the lowest latency, MPTCP has no mean to support this demand.

It can be concluded, that sequential path establishment relying with an initial path establishment over an externally given default route will result in experience reduction when using MPTCP. So this document proposes solutions to overcome the aforementioned limitations and provides a more robust connection setup compared to traditional MPTCP.

Introduction of RobE\_SIM and RobE\_eSIM aims to overcome the limitations of [RFC6824] and [RFC8684], using one initial flow and introduces the concept of multiple potential initial flows triggered simultaneously. Potential initial flows give the freedom to use more than one path to request multipath capability and select the initial flow at a later point. Potential initial flow mechanisms and the gain of robustness and performance over the traditional MPTCP connection setup are evaluated in [RobE\_slides] and [RobE\_paper]. RobE\_SIM is a break-before-make mechanism, guaranteeing at least the robust connection establishment, however the RobE\_eSIM reuses every potential initial flow request to combine it with less overhead and accelerated multipath availability, leveraging a new MPTCP option MP\_JOIN\_CAP. From a standardization perspective, the RobE\_SIM is fully compliant with [RFC6824] and [RFC8684] and is herein more of a descriptive and procedural nature. The RobE\_eSIM requires a new MPTCP option but offers the potential to significantly improve the MPTCP experience.

For the limitation of the default initial path, RobE\_IPS makes no changes to standard MPTCP procedure and improves the performance of connection establishment by introducing an initial path selection strategy and required algorithms. The input for strategy and algorithms is the transmission status information which represents the transmission performance of each available path or network interface. The transmission status information is characterized by at least one of the parameters: signal strength, throughput, round-trip time (RTT), and link success rate. In this way, a path with better transmission performance can be learned and determined and the respective network interface can be used for connection establishment.

The most simple approach for a robust MPTCP session establishment is RobE\_TIMER, iterating the process of initial path establishment over all available paths, if the previous try has failed. Triggering a new try on a next path is depending on an expiration timer, preferably re-use TCP's in-built expiration timer.

Table 1 summarizes the impact of RobE\_TIMER, RobE\_SIM, RobE\_eSIM, and RobE\_IPS compared to [RFC6824] and [RFC8684].

Scenario	MPTCP	RobE_TIMER	RobE_SIM	RobE_eSIM	RobE_IPS
IP packet loss	Delayed connection	In the scope of timer	No impact	No impact	Delayed connection
IP broken	No connection	In the scope of timer	No impact	No impact	No connection
IP setup duration dependency	Default route	Default route (+ path 1..n)	Fastest path	Fastest path	Selected path
MP avail-ability duration	MP_CAPABLE HS + MP_JOIN HS	sum <sub>1..n</sub> (MP_CAPABLE <sub>n</sub> HS) + MP_JOIN HS	MP_CAPABLE HS + MP_JOIN HS	max(MP_CAPABLE <sub>1</sub> .. MP_CAPABLE <sub>n</sub> HS)	MP_CAPABLE HS + MP_JOIN HS
Guaranteeing session setup	Depends on the default route	Yes	Yes	Yes	Depends on selection

Table 1: Overview RobE features during initial connection setup

| IP: Initial Path; MP: Multi-Path; HS: Handshake

### 1.1. Terminology

This document makes use of a number of terms that are either MPTCP-specific or have defined meaning in the context of MPTCP, as follows:

**Path:** A sequence of links between a sender and a receiver, defined in this context by a 4-tuple of source and destination address/port pairs.

**Subflow:** A flow of TCP segments operating over an individual path, which forms part of a larger MPTCP connection. A subflow is started and terminated similar to a regular TCP connection.

## 2. Implementation without MPTCP protocol adaptation

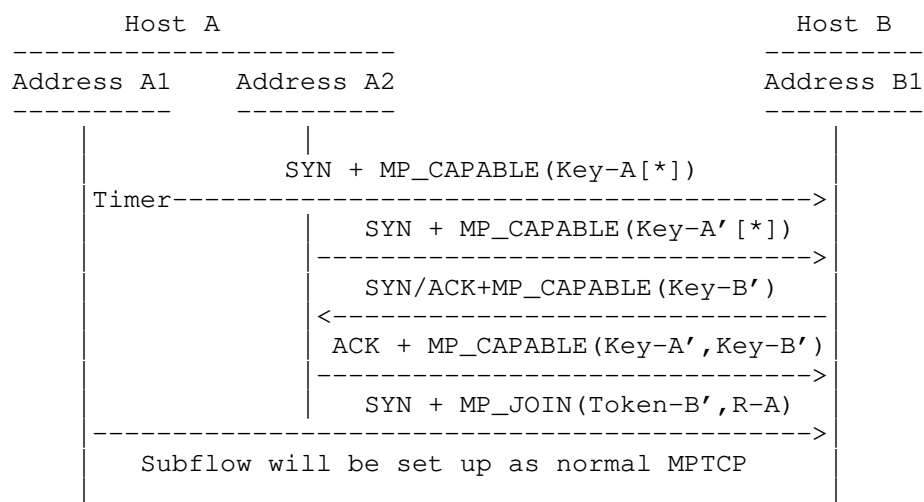
RobE\_TIMER, RobE\_SIM, and RobE\_IPS are compatible with the current MPTCP protocol definitions in [RFC6824] and [RFC8684] but may lack of the full optimization potential which requires protocol adaptation as detailed in Section 3. Following sections will describe the newly introduced mechanisms in detail.

### 2.1. Re-transmission Timer(RobE\_TIMER)

In RobE\_TIMER, a new connection is initiated by sending a SYN+MP\_CAPABLE along the initial path. If this path is functional, the solution will perform in the same way as classic MPTCP: the initial flow will be established, and subsequent flows can be created afterwards. If however the initial path is faulty, the retransmission will be triggered on another path. This path might circumvent the dysfunctional network, and allow the client to create an initial subflow. The first path is now seen as a subsequent path and the client sends SYN+MP\_JOIN messages to create a subsequent flow.

In high latency networks, the initial SYN+MP\_CAPABLE messages might be delayed until the client retries sending them on another path. Once the second SYN arrives at the server, it will try to complete the three-way handshake. If the first SYN was delayed by more than the retransmission time plus half a Round Trip Time (RTT) of the second path, it will arrive at the server after the second SYN. The server could now treat the segment as obsolete and drop it.





[\*] Key-A in the first MP-capable is related to RFC6824 only and does not exist in RFC8684.

Figure 2: The RobE\_TIMER Solution

Immediately after sending the final ACK of the initial handshake, subflows are established on the remaining paths as defined in [RFC6824] and [RFC8684]

[Notes: How to set the Timer is TBD. If there is the case that the first SYN on default path arrives earlier than that from the second path, the MPTCP connection will be initialized on the path of the first SYN. The server could treat the second SYN as obsolete and drop it.]

## 2.2. Simultaneous Initial Paths Simple Version (RobE\_SIM)

RobE\_SIM is a sender only implementation and no prior negotiation with the receiver side is required. In RobE\_SIM, the MPTCP connection setup benefits from the fastest path. As shown in Figure 3, host A initiates the connection handshake on more than one path independently (SA1 and SA2). The paths selected for RobE\_SIM and referred to as potential initial flows, can belong to the number of interfaces on the device or a subset selected on experience. When Host A receives the first SYN/ACK back from Host B (SA3), the path carrying this message is identified as the normal initial path. Host A sends then immediately a TCP RST message (SA6.1) on any other path used for simultaneous connection setup causing an immediate termination of assigned flows (break-before-make). The terminated ones are merged as subsequent subflows following the JOIN procedure

described in [RFC6824] and [RFC8684]. The process is equivalent to any other scenario where the SYN/ACK arrives on an other path than depicted in Figure 3.

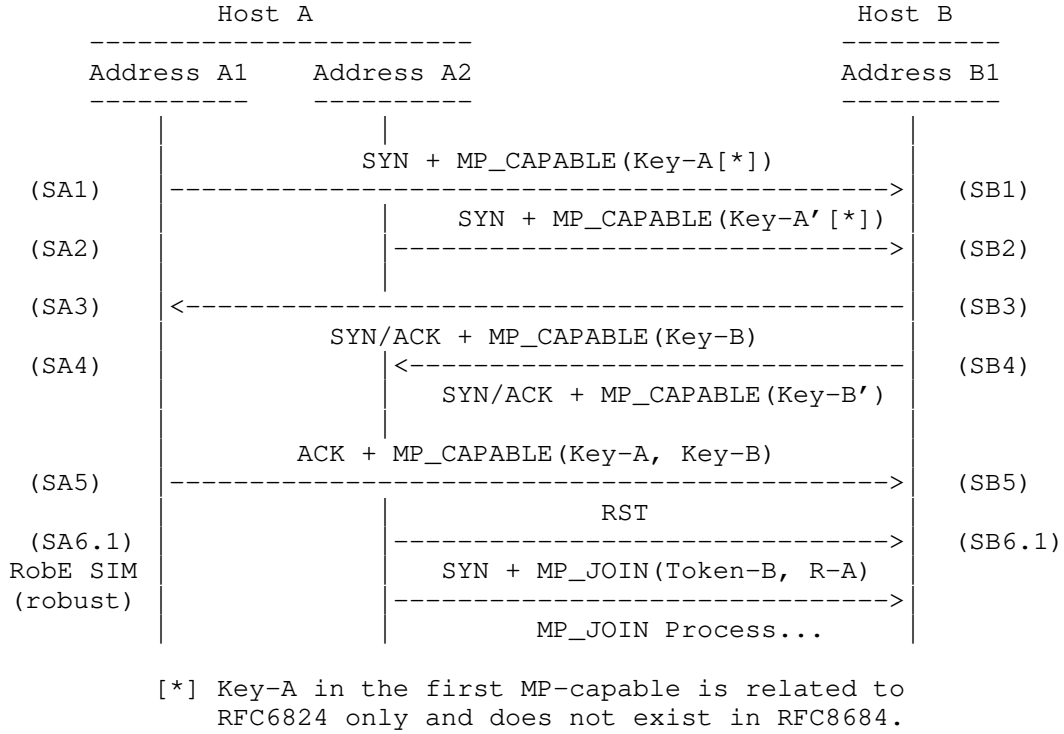


Figure 3: MPTCP RobE\_SIM Connection Setup

### 2.3. Heuristic Initial Path Selection (RobE\_IPS)

#### 2.3.1. Architecture

Figure 4 provides the architecture for RobE\_IPS and employs an "Initial Path Selection" logic which can be integrated into the MPTCP stack or exists as an isolated module in the terminal. The IPS logic has access to a set of transmission status information for each available path or its belonging network interfaces. When an application starts a first communication, IPS selects based on the available path transmission characteristics the path with the highest probability to succeed.

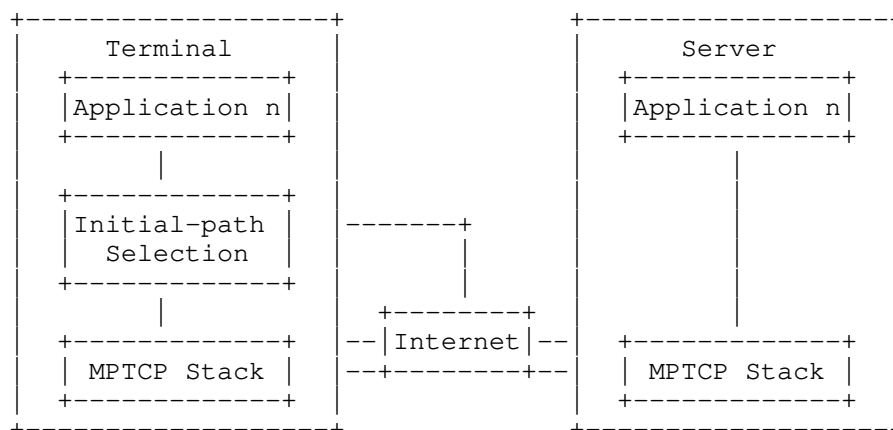


Figure 4: Architecture for Initial-path Selection

### 2.3.2. Typical Scenarios

Two typical RobE\_IPS scenarios are presented in this section. Figure 5 shows the "Initial Path Selection" logic executed for each MPTCP connection establishment. On the other hand Figure 6 describes that "Initial Path Selection" in case no path information is available. Considering the fact that no heuristics are given before a recent MPTCP connection was established, the default initial path can be adopted. Further combinations and implementations with more or less sophisticated heuristics are possible.

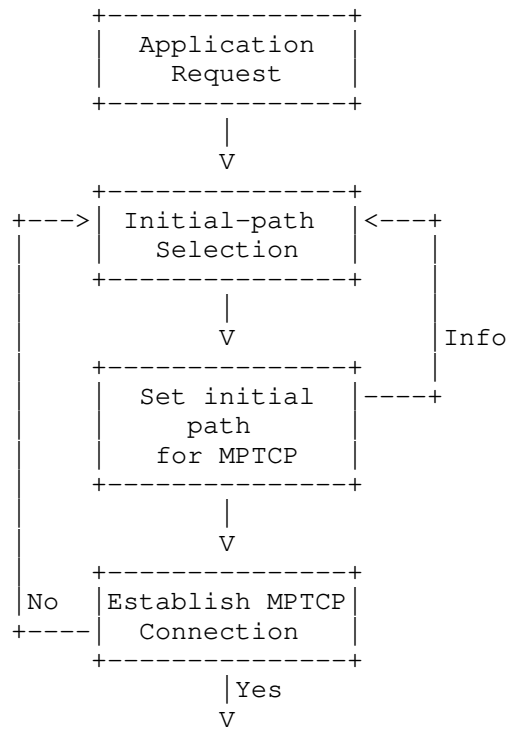


Figure 5: RobE\_IPS for each connection establishment

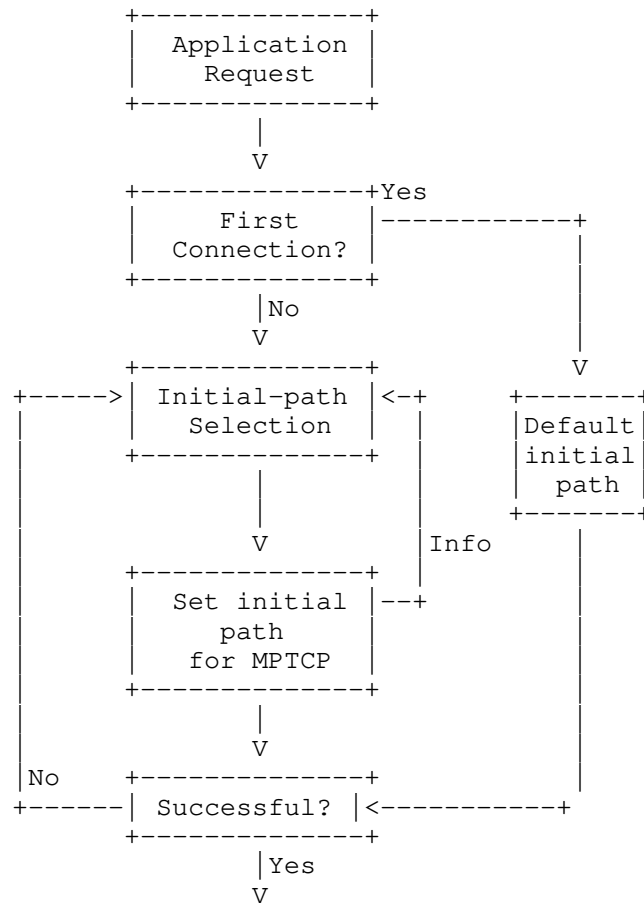


Figure 6: RobE\_IPS using default route when no meaningful heuristic available

Figure 7 shows the process flow of "Initial Path Selection". Upon a request from an application, the IPS logic will acquire transmission status information which represents the transmission performance of each available path or network interface and evaluate it. The transmission status information is characterized by at least one of the parameters: signal strength, throughput, round-trip time (RTT), and link success rate. In this way, the path with the best transmission performance can be determined and used for connection establishment.

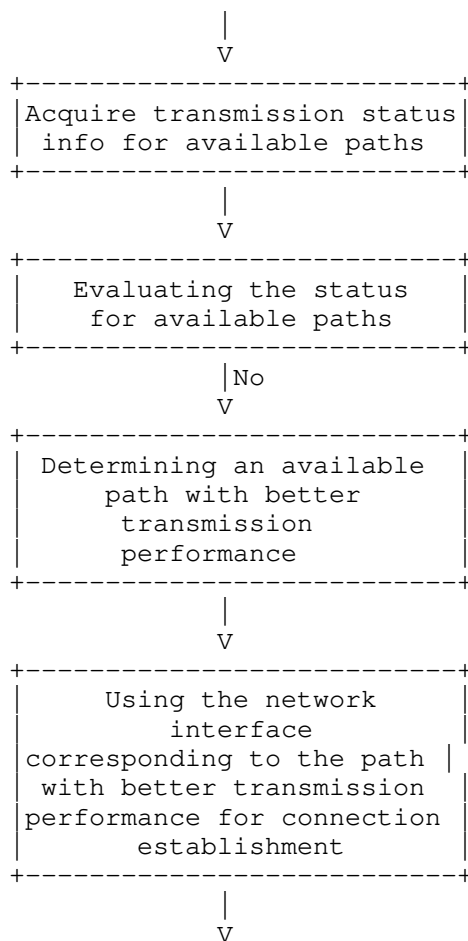


Figure 7: Implementation process for Initial Path Selection

### 2.3.3. Path decision information

The level of heuristic can be mainly divided into three layers: application level, transport-layer level and link-layer level based on the information acquisition method. For example, RTT can be calculated for each path within an MPTCP connection and belongs thereof to the transport-layer level. The transmission status information for each available path SHOULD be characterized by at least one of the parameters: signal strength, throughput, RTT, and link success rate. Application level information are more seen for statistical purposes.

- \* Application level: application name, domain name, port number, and location.
- \* Transport-layer level: RTT, CWND, Error rate.

#### 2.3.4. Initial Path Selection use local RTT information

Figure 8 presents an "Initial Path Selection" logic based on RTT, e.g. assuming two paths over LTE and WiFi access. RTT calculation on the transport layer usually reflects the time when an information is sent and a related acknowledgment received. For an asymmetric usage (e.g. download only) of a communication it might happen that recent RTT calculation is only available on sender side which is possibly not the side which employs the IPS logic. A solution for this can be found in Section 3.2. Instead of using the most recent RTT value of a path a filtered value consisting of several measured RTTs can be used. A RTT can also be derived from link layer information but may have a limited meaning only when it does not represent the end-to-end latency.

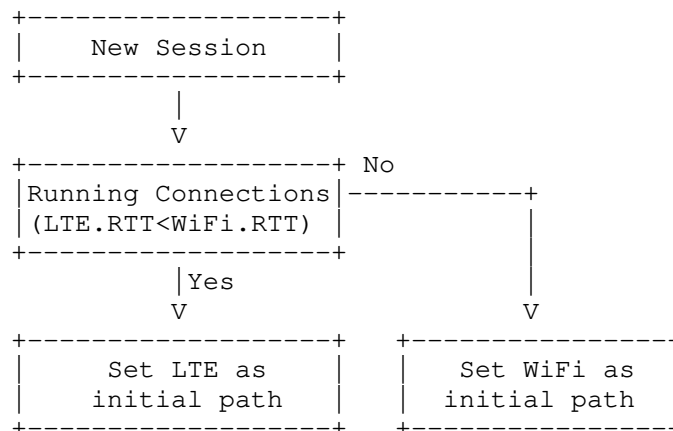


Figure 8: Initial-path Selection based on RTT

#### 2.4. Combination of RobE\_SIM and RobE\_IPS

In an implementation, a single solution may not be sufficient to achieve an expected behavior. Combination of approaches to improve robustness is recommended therefore. Figure 9 shows the combination of RobE\_SIM and RobE\_IPS. RobE\_SIM can be used at the very beginning when the sender is without any path information followed by RobE\_IPS for consecutive connections.

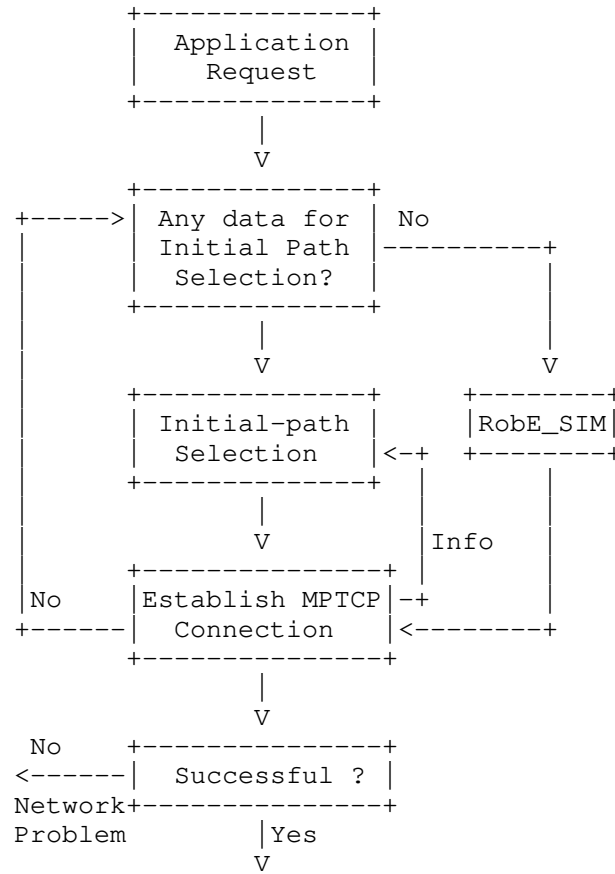


Figure 9: Combination of RobE\_SIM and RobE\_IPS

## 2.5. Combination of RobE\_TIMER and RobE\_IPS

Since RobE\_IPS solely does not guarantee that a session can be set up based on the selection of initial path, it can also be combined with RobE\_TIMER which generates less overhead compared to the combination with RobE\_SIM in Section 2.4 and guarantees session setup. RobE\_TIMER can be introduced to optimize the control of path switching when the initial path selected by RobE\_IPS is dysfunctional. When the system enables RobE\_IPS and uses the selected initial path for session establishment, it sets the timer for path switching. When timer is expired, the system will change to another path to re-establish connection according to Section 2.1.



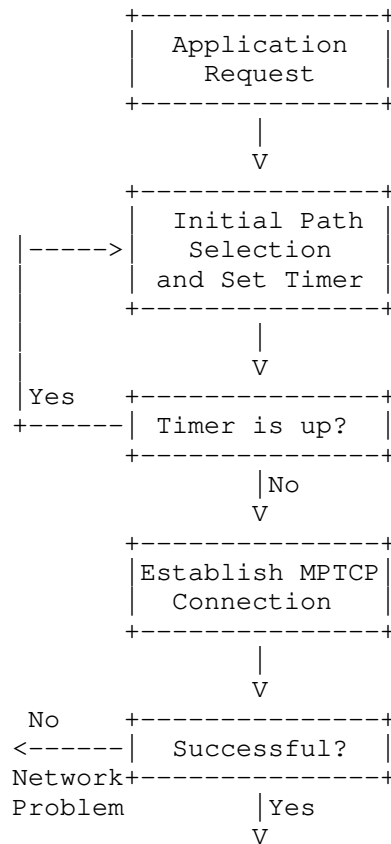


Figure 10: Combination of RobE\_Timer and RobE\_IPS

### 3. Implementation with Bi-directional MPTCP Support

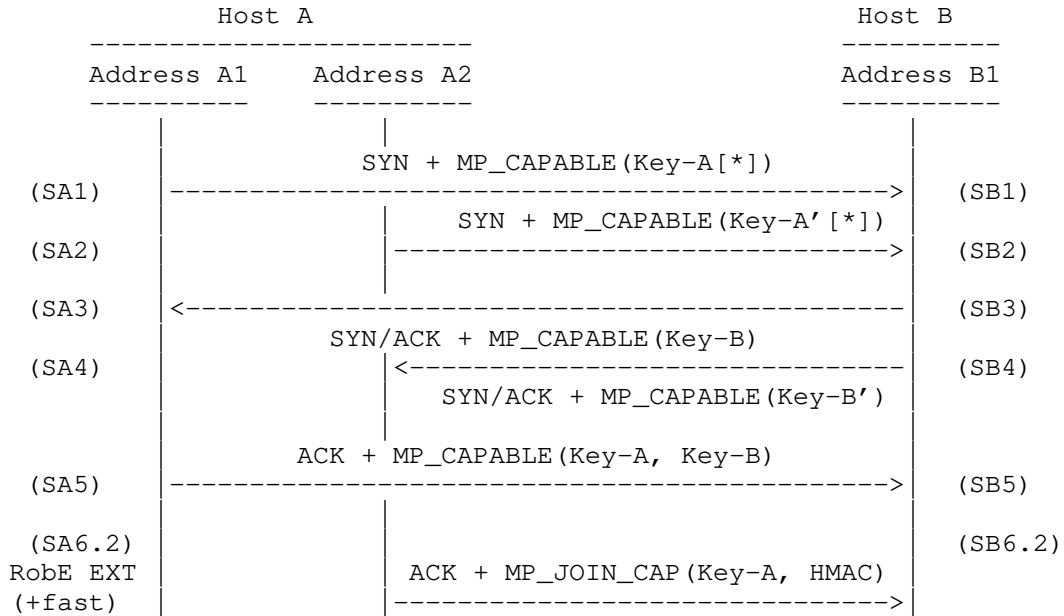
Solutions which requires bi-directional support between two MPTCP hosts promise to have better and possibly more features. However, they cannot be defined without extending current standards in [RFC6824] and [RFC8684]. The RobE\_SIM and RobE\_IPS approach are both capable of profiting from an explicit support of the remote end host and will be defined within this section.

### 3.1. Simultaneous Initial Paths Extended Version (RobE\_eSIM)

RobE\_eSIM extends RobE\_SIM by reusing the potential initial flows. This eliminates the overhead from RobE\_SIM by introducing a new option MP\_JOIN\_CAP and accelerate the transmission speed by early availability of multiple paths. Further it relaxes the dependency on a reliable third ACK of the 3-way handshake in [RFC8684]. Remote endpoint support can be negotiated in two ways, an implicit one described in Section 3.1.1 or an explicit one which is described in Section 3.1.2.

#### 3.1.1. RobE\_eSIM implicit Negotiation and Procedure

Similar to RobE\_SIM in Section 2.2, the establishment process of [RFC6824] or [RFC8684] is applied independently on multiple paths simultaneously. In Figure 11 this is shown in SA1 and SA2. The first path which returns a SYN/ACK (e.g. SA3) is selected as the initial path and proceeds with the traditional establishment process (SA5). Any other path which has to send the final ACK of the 3-way handshake includes a new option MP\_JOIN\_CAP (see definition in Section 3.1.3.2) instead of an MP\_CAPABLE (SA6.2).



[\*] Key-A in the first MP-capable is related to RFC6824 only and does not exist in RFC8684.

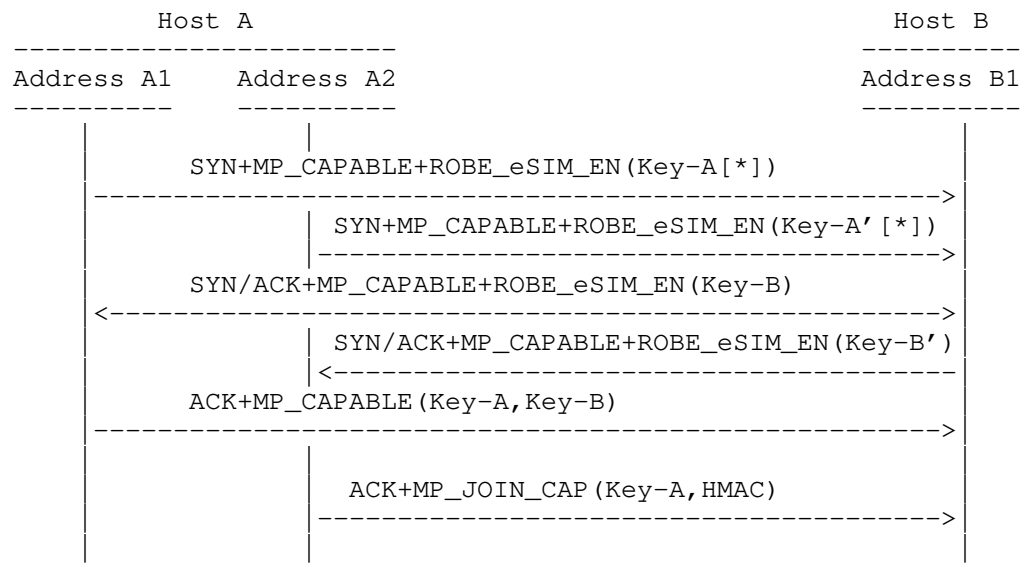
Figure 11: MPTCP RobE\_eSIM implicit Connection Setup

Following the possible process in Figure 11, two further constellations are imaginable and elaborated below.

1. In the flow diagram Figure 11, A1<->B1 is assumed to be the initial flow. A2<->B1 shall be recycled and the ACK is sent with MP\_JOIN\_CAP. Furthermore, the MP\_CAPABLE arrives first at Host B (SB5) and the MP\_JOIN\_CAP afterwards (SB6.2). When the MP\_JOIN\_CAP is received, Host B has to iterate over the connection list once (like MP\_JOIN) and check for Key-A availability. If a Key-A connection is found, this one is validated against the HMAC value. The validation has two reasons: first, several Key-A can exist, because different hosts may choose the same Key-A by accident. Furthermore, no one can join a connection by just recording/brute-forcing Key-A and duplicating the request.
2. Like above, but MP\_JOIN\_CAP arrives before last MP\_CAPABLE at Host B
  - \* [RFC8684]; Based on Key-A, Host B will iterate over the connection list, but it will not find a match, because Key-A of the previous selected initial flow (SA3, SA5) has not arrived yet. So it will continue with a fast iteration only over the connections which are still in establishment phase using the 10 bit Key-B fast hash (crcl6(Key-B) & 0x3FF). If it matches against a (precomputed) existing Key-B\_fast\_hash in the connection list, it will validate the request using the HMAC(Key-A+B+B') to ensure legitimation. If successful, both, the initial flow and the MP\_JOIN\_CAP flow, can be immediately established. This is true, because without the knowledge of Key-B, Host A could not calculate the HMAC. So it is clear, that Host A had received the SYN/ACK (SB3). This also mitigates the exchange of a reliable ACK during the handshake process. MPTCP sends the Key-A only with the last ACK and therefore prevents subsequent flow establishment until successful reception at Host B. Using RobE\_EXT, the reception of an MP\_JOIN\_CAP ([RFC8684]) is sufficient to establish both, the path carrying Key-B and Key-B'.
  - \* [RFC6824]; Can match based on Key-A, same effort as for an MP\_JOIN.
3. A2<->B1 is selected as initial flow, because the respective SYN/ACK returns earlier at Host A. It is the same as above, just the other way round.

3.1.2. RobE\_eSIM explicit Negotiation and Procedure

The process of an explicit negotiation of RobE\_eSIM follows Figure 11 but uses the ROBE\_eSIM\_EN option Figure 13 additionally during the handshake procedure.



[\*] Key-A in the first MP-capable is related to RFC6824 only and does not exist in RFC8684.

Figure 12: MPTCP RobE\_eSIM explicit Connection Setup

3.1.3. Protocol Adaptation

3.1.3.1. ROBE\_eSIM\_EN Option

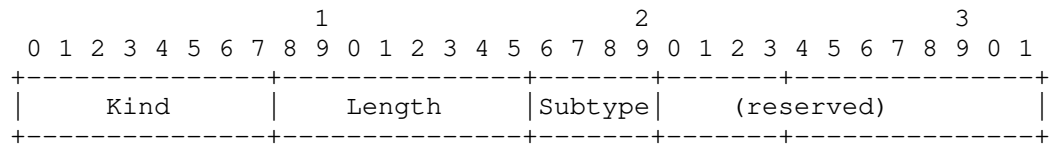


Figure 13: ROBE\_eSIM\_EN\_OPTION

3.1.3.2. MP\_JOIN\_CAP Option

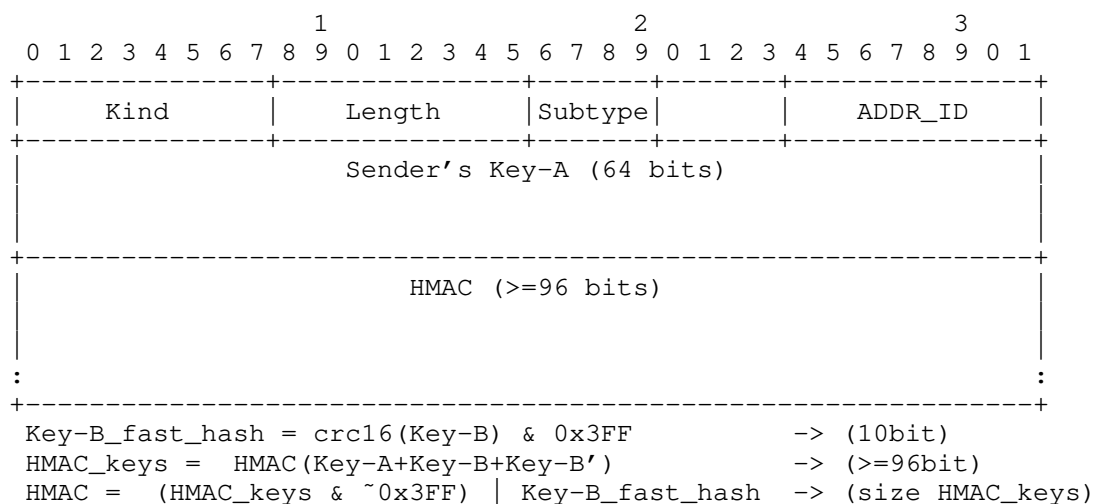


Figure 14: MP\_JOIN\_CAP

Computational effort on receiver side is most often expected to be the same as with MP\_JOIN. Key-A ensures identification of related flows Key-B\_fast\_hash enables MP session even when selected initial flow is not fully established yet (slight computational overhead). HMAC authenticates relationship of initial and potential initial flows.

#### 3.1.4. Fallback Mechanisms

##### 3.1.4.1. Fallback mechanism for implicit RobE\_eSIM

[TBD]

##### 3.1.4.2. Fallback mechanism for explicit RobE\_eSIM

This mechanism considers that both sides support MPTCP capability but the receiver is not equipped with RobE\_eSIM. MPTCP session with RobE\_eSIM negotiation will seamlessly fallback to normal MPTCP process.

[Requires further check how an unaware Host B reacts on possible ROBE\_eSIM\_EN; Ignore or RST? See also RFC6824 Sec. 3.6 "Should fallback [...] the path does not support the MPTCP options"]

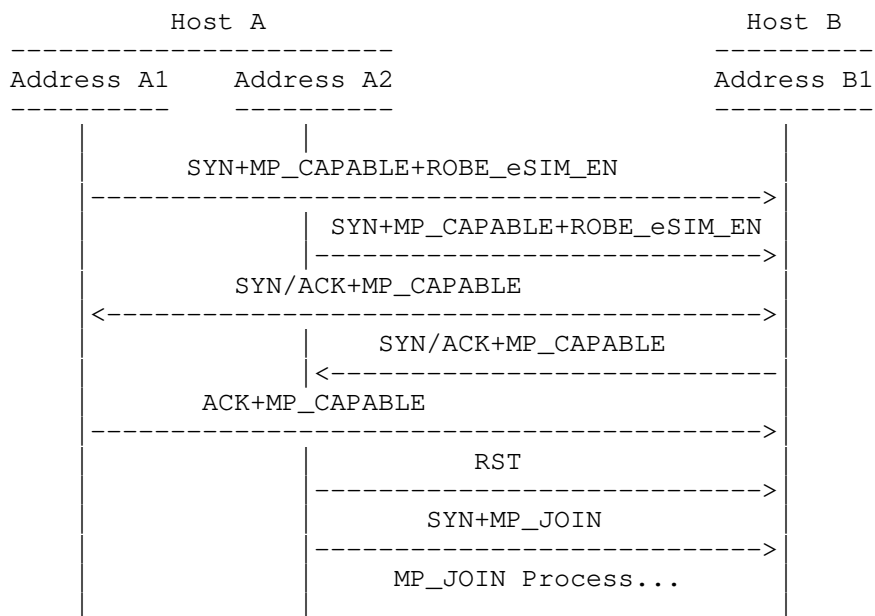


Figure 15: Fallback to MPTCP when missing RobE\_eSIM support

#### 3.1.4.3. Fallback to regular TCP when missing MPTCP support

When the receiver is not MPTCP enabled, MPTCP session with RobE\_eSIM negotiation will seamlessly fallback to regular process which is illustrated in this section.

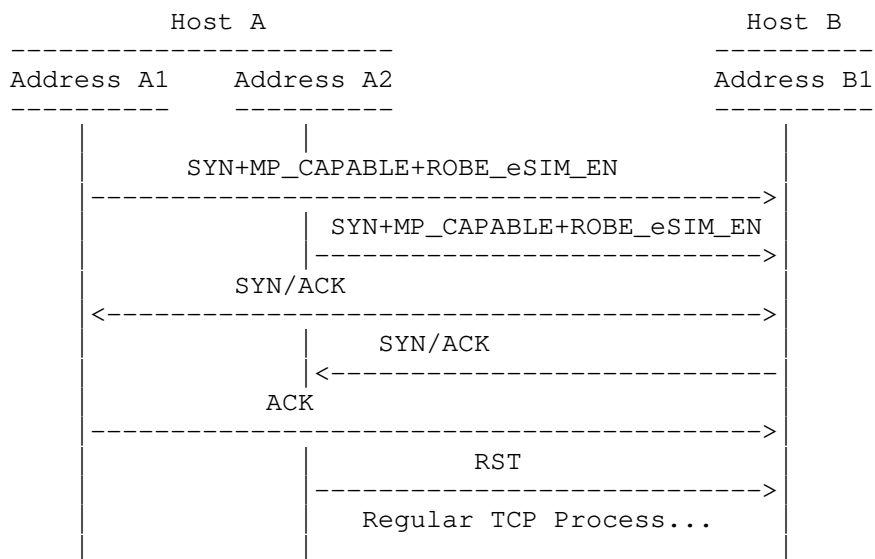
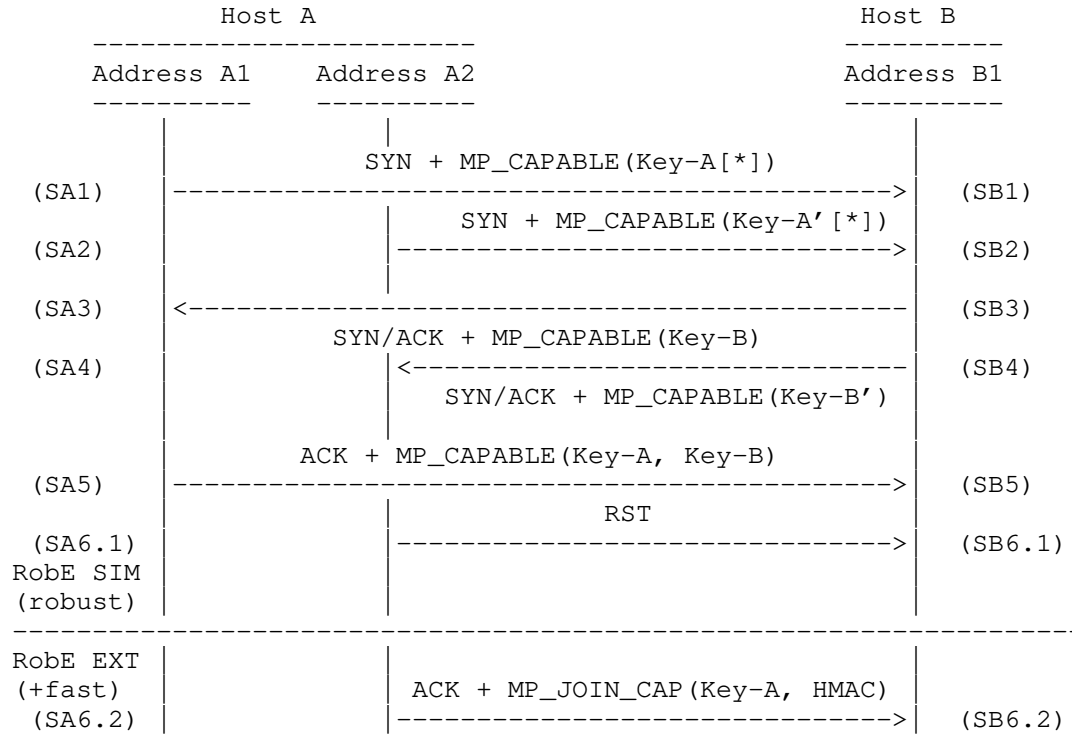


Figure 16: Fallback to TCP without MPTCP support

### 3.1.5. Comparison RobE\_SIM and RobE\_eSIM

Potential initial flows in RobE\_SIM Section 2.2 and RobE\_eSIM Section 3.1 guarantee MPTCP session establishment if at least one selected path for session establishment is functional. Figure 17 makes the differences between both approaches visible and points to the latest decision possibility during session setup when RobE\_SIM or RobE\_eSIM can be selected. Until SA5 in Figure 17 traditional MPTCP connection setup is independently applied on multiple paths simultaneously and offers to select the initial flow later (potential initial flows). The final decision which path is selected as the main one and the handling of the remaining flow(s) differs in SA6.1 when RobE\_SIM is applied or instead SA6.2 RobE\_eSIM.



[\*] Key-A in the first MP-capable is related to RFC6824 only and does not exist in RFC8684.

Figure 17: MPTCP RobE\_SIM and RobE\_eSIM connection setup

### 3.1.6. Security Consideration

[Tbd, however no differences to [RFC6824] and [RFC8684] are expected]

## 3.2. Heuristic Initial Path Selection with remote RTT Measurement

### 3.2.1. Description

Usually the path RTT can be determined by a time difference between sending a package and receiving an ACK and is integrated into the TCP protocol. For asymmetric transmission, the latest RTT for TCP flows is calculated by the side which sends data at latest and possible does not correspond to the site which employs RobE\_IPS. This problem is already elaborated in Section 2.3.4 and can be solved by transmitting the RTT information per subflow. The negotiation procedure is depicted in Figure 18 and uses the MPTCP option L\_RTT\_EN defined in Section 3.2.2.



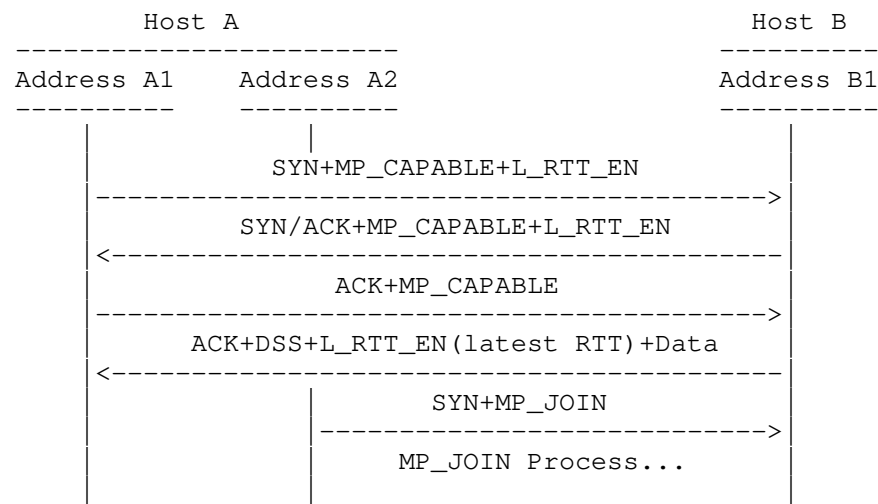


Figure 18: Negotiation procedure for RTT exchange

A successful negotiation allows the exchange of the measured RTT value from one subflow of an MPTCP host to another using the "Latest RTT" field within the L\_RTT\_EN option.

3.2.2. Protocol Adaptation

Calculating the "Latest RTT" by a remote host in an asymmetry transmission scenario should be transferred from remote host to the client running RobE\_IPS. So a new MPTCP subtype option named L\_RTT\_EN is allocated for this function. During the three-way handshake L\_RTT\_EN is used for negotiation of remote RTT measurement capability between client and server (in Section 3.2.1). When both parts support the usage of remote RTT measurement, the "Latest RTT" field in L\_RTT\_EN is applied for carrying the value of latest RTT computed by the remote host.

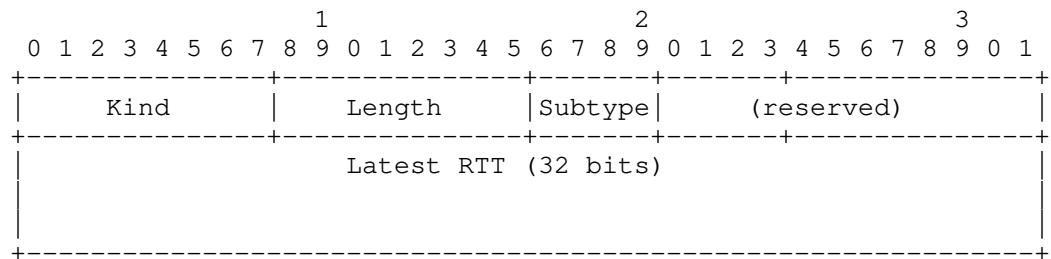


Figure 19: ROBE\_L\_RTT\_EN OPTION

### 3.2.3. Fallback Mechanism

When the receiver is not `L_RTT_EN` capable, MPTCP session with `L_RTT_EN` negotiation will seamlessly fallback to normal MPTCP process.

[TBD, Need same checks as Section 3.1.4.2]

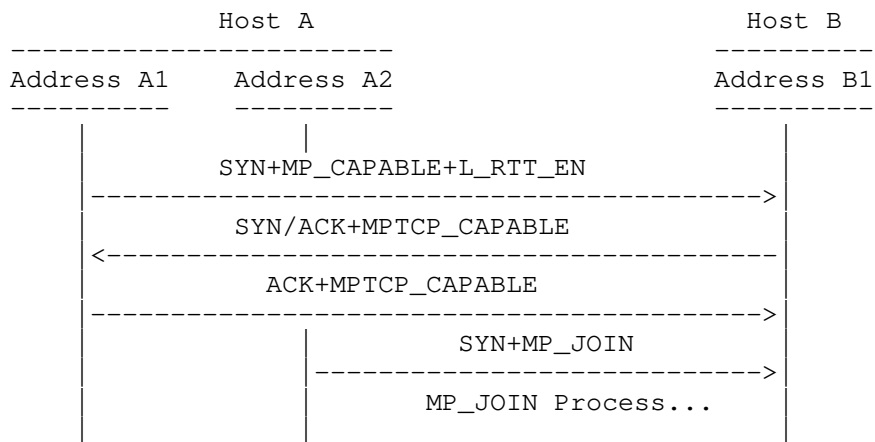


Figure 20: Fallback to MPTCP without RobE\_IPS

### 3.2.4. Security Consideration

[Tbd]

## 4. IANA Considerations

This document defines three new values to MPTCP Option Subtype as following.

Value	Symbol	Name	Reference
TBD	ROBE_eSIM_EN	RobE_eSIM enabled	Section 3.1
TBD	MP_JOIN_CAP	Join connection directly in RobE_eSIM	Section 3.1
TBD	L_RTT_EN	Server RTT enabled	Section 3.2

Table 2: RobE Option Subtypes

## 5. References

### 5.1. 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>>.
- [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>>.
- [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>>.

### 5.2. Informative References

- [RobE\_paper] Amend, M., Rakocevic, V., Matz, A.P., and E. Bogenfeld, "RobE: Robust Connection Establishment for Multipath TCP", ANRW '18 p. 76-82, 16 July 2018, <<http://doi.acm.org/10.1145/3232755.3232762>>.
- [RobE\_slides] Amend, M., Matz, A.P., and E. Bogenfeld, "A proposal for MPTCP Robust session Establishment (MPTCP RobE)", IETF 99 Multipath TCP WG session, 18 July 2017, <<https://datatracker.ietf.org/meeting/99/materials/slides-99-mptcp-a-proposal-for-mptcp-robust-session-establishment-mptcp-robe-01>>.

### Authors' Addresses

Markus Amend  
Deutsche Telekom  
Deutsche-Telekom-Allee 9  
64295 Darmstadt  
Germany  
Email: Markus.Amend@telekom.de

Jiao Kang  
Huawei  
D2-03, Huawei Industrial Base  
Shenzhen  
Guangdong, 518129  
China  
Email: kangjiao@huawei.com

TCPM  
Internet-Draft  
Obsoletes: 8312 (if approved)  
Intended status: Standards Track  
Expires: 11 September 2021

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

CUBIC for Fast and Long-Distance Networks  
draft-eggert-tcpm-rfc8312bis-03

Abstract

CUBIC is an extension to the traditional TCP standards. It differs from the traditional TCP standards only in the congestion control algorithm on the sender side. In particular, it uses a cubic function instead of the linear window increase function of the traditional TCP standards to improve scalability and stability under 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>.

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 11 September 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 . . . . .	7
4.1.2. Variables of Interest . . . . .	8
4.2. Window Increase Function . . . . .	9
4.3. AIMD-Friendly Region . . . . .	10
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 . . . . .	14
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. Acknowledgements	25
	Appendix B. Evolution of CUBIC	25
	B.1. Since draft-eggert-tcpm-rfc8312bis-02	25
	B.2. Since draft-eggert-tcpm-rfc8312bis-01	25
	B.3. Since draft-eggert-tcpm-rfc8312bis-00	26
	B.4. Since RFC8312	26
	B.5. Since the Original Paper	27
	Authors' Addresses	27

## 1. Introduction

The low utilization problem of traditional TCP in fast and long-distance networks is well documented in [K03] and [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. 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.

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 a large number of 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 of 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.

`_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:

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

Figure 2

where `_cwnd_(start)_` is the congestion window at the beginning of the current congestion avoidance stage. 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

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

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

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

Figure 3

Based on the above analysis, CUBIC uses Figure 4 to estimate the window size `_W(est)_` of `AIMD(__aimd__, __aimd__)` with

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

$$= \text{aimd} * \text{cubic}$$

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.

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

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 `_ssthresh_` immediately as described below. An implementation MAY set a smaller `_ssthresh_` than suggested below to accomodate 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 `_ssthresh_` value. The parameter `__(cubic)_` SHOULD be set to 0.7.

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

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

                                // window reduction
cwnd = ssthresh
```

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.

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

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:

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

Figure 5

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

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

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

CUBIC has 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]. Additionally, there is decade-long deployment experience with CUBIC on the Internet.

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., Sathiaselan, 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. Acknowledgements

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

#### Appendix B. Evolution of CUBIC

##### B.1. 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.2. 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 in the Internet. (#38 (<https://github.com/NTAP/rfc8312bis/issues/38>))
- \* change introductory text to reflect the significant broader deployment of CUBIC in the Internet. (#39 (<https://github.com/NTAP/rfc8312bis/issues/39>))
- \* rephrase introduction to avoid referring to variables that have not been defined yet.

## B.3. 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.4. 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.5. 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 Working Group  
Internet-Draft  
Intended status: Experimental  
Expires: August 23, 2021

C. Gomez  
UPC  
J. Crowcroft  
University of Cambridge  
February 19, 2021

TCP ACK Rate Request Option  
draft-gomez-tcpm-ack-rate-request-02

Abstract

TCP Delayed Acknowledgments (ACKs) is a widely deployed mechanism that allows reducing protocol overhead in many scenarios. However, Delayed ACKs may also contribute to suboptimal performance. When a relatively large congestion window (cwnd) can be used, less frequent ACKs may be desirable. On the other hand, in relatively small cwnd scenarios, eliciting an immediate ACK may avoid unnecessary delays that may be incurred by the Delayed ACKs mechanism. This document specifies the TCP ACK Rate Request (TARR) option. This option allows a sender to indicate the ACK rate to be used by a receiver, and it also allows to request immediate ACKs from a receiver.

Status of This Memo

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

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

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

This Internet-Draft will expire on August 23, 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 . . . . .	2
2. Conventions used in this document . . . . .	3
3. TCP ACK Rate Request Functionality . . . . .	4
3.1. Sender behavior . . . . .	4
3.2. Receiver behavior . . . . .	4
4. Option Format . . . . .	5
5. IANA Considerations . . . . .	6
6. Security Considerations . . . . .	7
7. Acknowledgments . . . . .	7
8. References . . . . .	7
8.1. Normative References . . . . .	7
8.2. Informative References . . . . .	8
Authors' Addresses . . . . .	8

## 1. Introduction

Delayed Acknowledgments (ACKs) were specified for TCP with the aim to reduce protocol overhead [RFC1122]. With Delayed ACKs, a TCP delays sending an ACK by up to 500 ms (often 200 ms, with lower values in recent implementations such as ~50 ms also reported), and typically sends an ACK for at least every second segment received in a stream of full-sized segments. This allows combining several segments into a single one (e.g. the application layer response to an application layer data message, and the corresponding ACK), and also saves up to one of every two ACKs, under many traffic patterns (e.g. bulk transfers). The "SHOULD" requirement level for implementing Delayed ACKs in RFC 1122, along with its expected benefits, has led to a widespread deployment of this mechanism.

However, there exist scenarios where Delayed ACKs contribute to suboptimal performance. We next roughly classify such scenarios into two main categories, in terms of the congestion window (cwnd) size and the Maximum Segment Size (MSS) that would be used therein: i) "large" cwnd scenarios (i.e.  $cwnd \gg MSS$ ), and ii) "small" cwnd scenarios (e.g. cwnd up to  $\sim MSS$ ).

In "large" cwnd scenarios, increasing the number of data segments after which a receiver transmits an ACK beyond the typical one (i.e. 2 when Delayed ACKs are used) may provide significant benefits. One

example is mitigating performance limitations due to asymmetric path capacity (e.g. when the reverse path is significantly limited in comparison to the forward path) [RFC3449]. Another advantage is reducing the computational cost both at the sender and the receiver, and reducing network packet load, due to the lower number of ACKs involved.

In many "small" cwnd scenarios, a sender may want to request the receiver to acknowledge a data segment immediately (i.e. without the additional delay incurred by the Delayed ACKs mechanism). In high bit rate environments (e.g. data centers), a flow's fair share of the available Bandwidth Delay Product (BDP) may be in the order of one MSS, or even less. For an accordingly set cwnd value (e.g. cwnd up to MSS), Delayed ACKs would incur a delay that is several orders of magnitude greater than the RTT, severely degrading performance. Note that the Nagle algorithm may produce the same effect for some traffic patterns in the same type of environments [RFC8490]. In addition, when transactional data exchanges are performed over TCP, or when the cwnd size has been reduced, eliciting an immediate ACK from the receiver may avoid idle times and allow timely continuation of data transmission and/or cwnd growth, contributing to maintaining low latency.

Further "small" cwnd scenarios can be found in Internet of Things (IoT) environments. Many IoT devices exhibit significant memory constraints, such as only enough RAM for a send buffer size of 1 MSS. In that case, if the data segment does not elicit an application-layer response, the Delayed ACKs mechanism unnecessarily contributes a delay equal to the Delayed ACK timer to ACK transmission. The sender cannot transmit a new data segment until the ACK corresponding to the previous data segment is received and processed.

With the aim to provide a tool for performance improvement in both "large" and "small" cwnd scenarios, this document specifies the TCP ACK Rate request (TARR) option. This option allows a sender to indicate the ACK rate to be used by a receiver, and it also allows to request immediate ACKs from a receiver.

## 2. Conventions used in this document

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

### 3. TCP ACK Rate Request Functionality

A TCP endpoint announces that it supports the TARR option by including the TARR option format (with the appropriate Length value, see Section 4) in packets that have the SYN bit set.

The next two subsections define the sender and receiver behaviors for devices that support the TARR option, respectively.

#### 3.1. Sender behavior

A TCP sender **MUST NOT** include the TARR option in TCP data segments to be sent if the TCP receiver does not support the TARR option.

A TCP sender **MAY** request a TARR-option-capable receiver to modify the ACK rate of the latter to one ACK every R full-sized data segments received from the sender. This request is performed by the sender by including the TARR option in the TCP header of a data segment. The TARR option carries the R value requested by the sender (see section 4). For the described purpose, the value of R **MUST NOT** be zero. The TARR option also carries the N field, which **MUST** be ignored when R is not set to zero.

When a TCP sender needs a data segment to be acknowledged immediately by a TARR-option-capable receiving TCP, the sender includes the TARR option in the TCP header of the data segment, with a value of R equal to zero. When R is set to zero, the N field of the same option indicates the number of subsequent data segments for which the sender also requests immediate ACKs.

A TCP sender **MAY** indicate that it has a reordering tolerance of R packets by setting the Ignore Order field of the TARR option to True (see Section 4).

#### 3.2. Receiver behavior

A receiving TCP conforming to this specification **MUST** process a TARR option present in a received data segment.

When the TARR option of a received segment carries an R value different from zero, a TARR-option-capable receiving TCP **MUST** modify its ACK rate to one ACK every R full-sized received data segments from the sender, as long as packet reordering does not occur. When R is different from zero, the receiving TCP **MUST** ignore the N field of the TARR option.

A TARR-option-capable TCP that receives a TARR option with the Ignore Order (I) field set to True (see Section 4), **MUST NOT** send an ACK

after each reordered data segment. Instead, it MUST continue to send one ACK every R received data segments. Otherwise (i.e., Ignore Order = False), such a receiver will need to send an ACK after each reordered data segment received.

If a TARR-option-capable TCP receives a segment carrying the TARR option with R=0, the receiving TCP MUST send an ACK immediately, and it MUST also send an ACK immediately after each one of the N next consecutive segments to be received. N refers to the corresponding field in the TARR option of the received segment (see Section 4).

#### 4. Option Format

The TARR option presents two different formats that can be identified by the corresponding format length. For packets that have the SYN bit set, the TARR option has the format shown in Fig. 1.

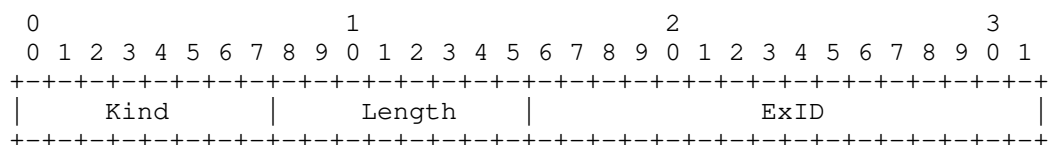


Figure 1: TCP ACK Rate Request option format for packets that have the SYN bit set.

Kind: The Kind field value is TBD.

Length: The Length field value is 4 bytes.

ExID: The experiment ID field size is 2 bytes, and its value is 0x00AC.

For packets that do not have the SYN bit set, the TARR option has the format and content shown in Fig. 2.

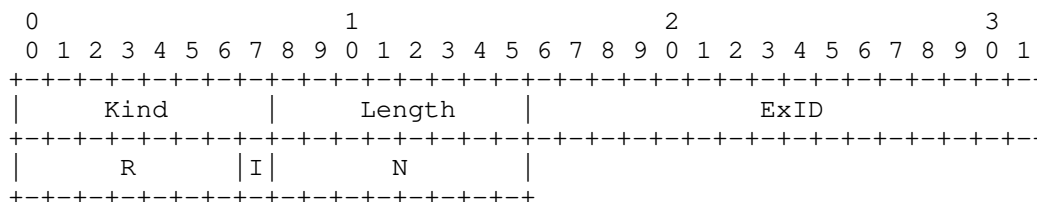


Figure 2: TCP ACK Rate Request option format.

Kind: The Kind field value is TBD.

Length: The Length field value is 6 bytes.

ExID: The experiment ID field size is 2 bytes, and its value is 0x00AC.

R: The size of this field is 7 bits. If all bits of this field are set to 0, the field indicates a request by the sender for the receiver to trigger one or more ACKs immediately. Otherwise, the field carries the binary encoding of the number of full-sized segments received after which the receiver is requested by the sender to send an ACK.

Ignore Order (I): The size of this field is 1 bit. This field either has the value 1 ("True") or 0 ("False"). When this field is set to True, the receiver MUST NOT send an ACK after each reordered data segment. Instead, it MUST continue to send one ACK every R received data segments.

N: The size of this field is 1 byte. When R=0, the N field indicates the number of subsequent consecutive data segments to be sent for which immediate ACKs are requested by the sender.

## 5. IANA Considerations

This document specifies a new TCP option (TCP ACK Rate Request) that uses the shared experimental options format [RFC6994], with ExID in network-standard byte order.

The authors plan to request the allocation of ExID value 0x00AC for the TCP option specified in this document.

## 6. Security Considerations

The TARR option opens the door to new security threats. This section discusses such new threats, and suggests mitigation techniques.

An attacker might be able to impersonate a legitimate sender, and forge an apparently valid packet intended for the receiver, in order to intentionally communicate a bad R value to the latter with the aim to damage communication or device performance. For example, in a small cwnd scenario, using a too high R value may lead to exacerbated RTT increase and throughput decrease. In other scenarios, a too low R value may contribute to depleting the energy of a battery-operated receiver at a faster rate or may lead to increased network packet load.

While Transport Layer Security (TLS) [RFC8446] is strongly recommended for securing TCP-based communication, TLS does not protect TCP headers, and thus cannot protect the TARR option fields carried by a segment. One approach to address the problem is using network-layer protection, such as Internet Protocol Security (IPsec) [RFC4301].

## 7. Acknowledgments

Bob Briscoe, Jonathan Morton, Richard Scheffenegger, Neal Cardwell, Michael Tuexen, Yuchung Cheng, Matt Mathis, Jana Iyengar, Gorrry Fairhurst, and Stuart Cheshire provided useful comments and input for this document. Jana Iyengar suggested including a field to allow a sender communicate its tolerance to reordering. Gorrry Fairhurst suggested adding a mechanism to request a number of consecutive immediate ACKs.

Carles Gomez has been funded in part by the Spanish Government through project PID2019-106808RA-I00, and by Secretaria d'Universitats i Recerca del Departament d'Empresa i Coneixement de la Generalitat de Catalunya 2017 through grant SGR 376.

## 8. References

### 8.1. Normative References

- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.

- [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>>.
- [RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, DOI 10.17487/RFC6994, August 2013, <<https://www.rfc-editor.org/info/rfc6994>>.

## 8.2. Informative References

- [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>>.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<https://www.rfc-editor.org/info/rfc4301>>.
- [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>>.
- [RFC8490] Bellis, R., Cheshire, S., Dickinson, J., Dickinson, S., Lemon, T., and T. Pusateri, "DNS Stateful Operations", RFC 8490, DOI 10.17487/RFC8490, March 2019, <<https://www.rfc-editor.org/info/rfc8490>>.

## Authors' Addresses

Carles Gomez  
UPC  
C/Esteve Terradas, 7  
Castelldefels 08860  
Spain  
  
Email: [carlesgo@entel.upc.edu](mailto:carlesgo@entel.upc.edu)

Jon Crowcroft  
University of Cambridge  
JJ Thomson Avenue  
Cambridge, CB3 0FD  
United Kingdom  
  
Email: [jon.crowcroft@cl.cam.ac.uk](mailto:jon.crowcroft@cl.cam.ac.uk)



TCPM Working Group  
Internet-Draft  
Intended status: Experimental  
Expires: 28 September 2022

C. Gomez  
UPC  
J. Crowcroft  
University of Cambridge  
March 2022

TCP ACK Rate Request Option  
draft-gomez-tcpm-ack-rate-request-04

Abstract

TCP Delayed Acknowledgments (ACKs) is a widely deployed mechanism that allows reducing protocol overhead in many scenarios. However, Delayed ACKs may also contribute to suboptimal performance. When a relatively large congestion window (cwnd) can be used, less frequent ACKs may be desirable. On the other hand, in relatively small cwnd scenarios, eliciting an immediate ACK may avoid unnecessary delays that may be incurred by the Delayed ACKs mechanism. This document specifies the TCP ACK Rate Request (TARR) option. This option allows a sender to request the ACK rate to be used by a receiver, and it also allows to request immediate ACKs from a receiver.

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 2 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 . . . . .	2
2. Conventions used in this document . . . . .	3
3. TCP ACK Rate Request Functionality . . . . .	4
3.1. Sender behavior . . . . .	4
3.2. Receiver behavior . . . . .	4
4. Option Format . . . . .	5
5. Changing the ACK rate during the lifetime of a TCP connection . . . . .	6
6. IANA Considerations . . . . .	7
7. Security Considerations . . . . .	7
8. Acknowledgments . . . . .	8
9. References . . . . .	8
9.1. Normative References . . . . .	8
9.2. Informative References . . . . .	9
Authors' Addresses . . . . .	9

## 1. Introduction

Delayed Acknowledgments (ACKs) were specified for TCP with the aim to reduce protocol overhead [RFC1122]. With Delayed ACKs, a TCP delays sending an ACK by up to 500 ms (often 200 ms, with lower values in recent implementations such as ~50 ms also reported), and typically sends an ACK for at least every second segment received in a stream of full-sized segments. This allows combining several segments into a single one (e.g. the application layer response to an application layer data message, and the corresponding ACK), and also saves up to one of every two ACKs, under many traffic patterns (e.g. bulk transfers). The "SHOULD" requirement level for implementing Delayed ACKs in RFC 1122, along with its expected benefits, has led to a widespread deployment of this mechanism.

However, there exist scenarios where Delayed ACKs contribute to suboptimal performance. We next roughly classify such scenarios into two main categories, in terms of the congestion window (cwnd) size and the Maximum Segment Size (MSS) that would be used therein: i) "large" cwnd scenarios (i.e.  $cwnd \gg MSS$ ), and ii) "small" cwnd scenarios (e.g. cwnd up to  $\sim MSS$ ).

In "large" cwnd scenarios, increasing the number of data segments after which a receiver transmits an ACK beyond the typical one (i.e. 2 when Delayed ACKs are used) may provide significant benefits. One example is mitigating performance limitations due to asymmetric path capacity (e.g. when the reverse path is significantly limited in comparison to the forward path) [RFC3449]. Another advantage is reducing the computational cost both at the sender and the receiver, and reducing network packet load, due to the lower number of ACKs involved.

In many "small" cwnd scenarios, a sender may want to request the receiver to acknowledge a data segment immediately (i.e. without the additional delay incurred by the Delayed ACKs mechanism). In high bit rate environments (e.g. data centers), a flow's fair share of the available Bandwidth Delay Product (BDP) may be in the order of one MSS, or even less. For an accordingly set cwnd value (e.g. cwnd up to MSS), Delayed ACKs would incur a delay that is several orders of magnitude greater than the RTT, severely degrading performance. Note that the Nagle algorithm may produce the same effect for some traffic patterns in the same type of environments [RFC8490]. In addition, when transactional data exchanges are performed over TCP, or when the cwnd size has been reduced, eliciting an immediate ACK from the receiver may avoid idle times and allow timely continuation of data transmission and/or cwnd growth, contributing to maintaining low latency.

Further "small" cwnd scenarios can be found in Internet of Things (IoT) environments. Many IoT devices exhibit significant memory constraints, such as only enough RAM for a send buffer size of 1 MSS. In that case, if the data segment does not elicit an application-layer response, the Delayed ACKs mechanism unnecessarily contributes a delay equal to the Delayed ACK timer to ACK transmission. The sender cannot transmit a new data segment until the ACK corresponding to the previous data segment is received and processed.

With the aim to provide a tool for performance improvement in both "large" and "small" cwnd scenarios, this document specifies the TCP ACK Rate request (TARR) option. This option allows a sender to request the ACK rate to be used by a receiver, and it also allows to request immediate ACKs from a receiver.

## 2. Conventions used in this document

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

### 3. TCP ACK Rate Request Functionality

A TCP endpoint announces that it supports the TARR option by including the TARR option format (with the appropriate Length value, see Section 4) in packets that have the SYN bit set.

Upon reception of a SYN segment carrying the TARR option, a TARR-option-capable endpoint **MUST** include the TARR option in the SYN-ACK segment sent in response.

The next two subsections define the sender and receiver behaviors for devices that support the TARR option, respectively.

#### 3.1. Sender behavior

A TCP sender **MUST NOT** include the TARR option in TCP segments to be sent if the TCP receiver does not support the TARR option.

A TCP sender **MAY** request a TARR-option-capable receiver to modify the ACK rate of the latter to one ACK every R data segments received from the sender. This request is performed by the sender by including the TARR option in the TCP header of a segment. The TARR option carries the R value requested by the sender (see section 4).

When a TCP sender needs a data segment to be acknowledged immediately by a TARR-option-capable receiving TCP, the sender includes the TARR option in the TCP header of the data segment, with a value of R equal to 1.

A TCP segment carrying retransmitted data is not required to include a TARR option.

#### 3.2. Receiver behavior

A receiving TCP conforming to this specification **MUST** process a TARR option present in a received segment.

A TARR-option-capable receiving TCP **SHOULD** modify its ACK rate to one ACK every R received data segments from the sender. If a TARR-option-capable TCP receives a segment carrying the TARR option with R=1, the receiving TCP **SHOULD** send an ACK immediately.

If packet reordering occurs, a TARR-option-capable receiver should send a duplicate ACK immediately when an out-of-order segment arrives [RFC5681]. After sending a duplicate ACK, the receiver **MAY** send the next non-duplicate ACK after R data segments received. Note also that the receiver might be unable to send ACKs at the requested rate (e.g., due to lack of resources); on the other hand, the receiver

might opt not to fulfill a request for security reasons (e.g., to avoid or mitigate an attack by which a large number of senders request disabling delayed ACKs simultaneously and send a large number of data segments to the receiver).

The request to modify the ACK rate of the receiver holds until the next segment carrying a TARR option is received.

#### 4. Option Format

The TARR option presents two different formats that can be identified by the corresponding format length. For packets that have the SYN bit set, the TARR option has the format shown in Fig. 1.

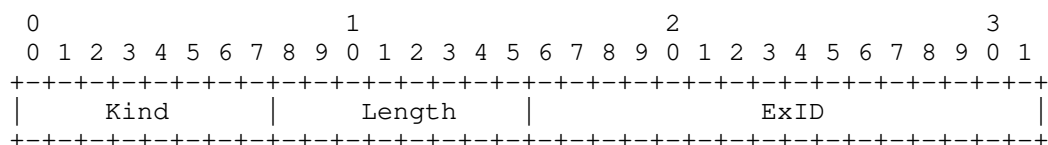


Figure 1: TCP ACK Rate Request option format for packets that have the SYN bit set.

Kind: The Kind field value is TBD.

Length: The Length field value is 4 bytes.

ExID: The experiment ID field size is 2 bytes, and its value is 0x00AC.

For packets that do not have the SYN bit set, the TARR option has the format and content shown in Fig. 2.

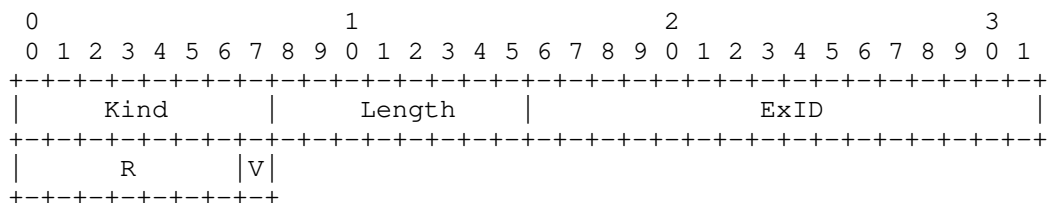


Figure 2: TCP ACK Rate Request option format.

Kind: The Kind field value is TBD.

Length: The Length field value is 5 bytes.

ExID: The experiment ID field size is 2 bytes, and its value is 0x00AC.

R: The size of this field is 7 bits. The field carries the ACK rate requested by the sender. The minimum value of R is 1.

Note: there are currently two options being considered regarding the semantics of the R field:

OPTION 1: the R field corresponds to the binary encoding of the requested ACK rate. The maximum value of R is 127. A receiver MUST ignore an R field with all bits set to zero. (TO-DO: if OPTION 1 is selected, see how to handle all bits of R being equal to zero.)

OPTION 2: the R field is composed of two subfields: the 5 leftmost bits represent a mantissa (m) and the 2 rightmost bits represent an exponent (e). The value of the requested ACK rate is obtained as  $R = (m+1) * 2^{(2*e)}$ . The maximum value of R is 2048.

ReserVed (V): The size of this field is 1 bit. This bit is reserved for future use.

## 5. Changing the ACK rate during the lifetime of a TCP connection

In some scenarios, setting the ACK rate once for the whole lifetime of a TCP connection may be suitable. However, there are also cases where it may be desirable to modify the ACK rate during the lifetime of a connection.

The ACK rate to be used may depend on the cwnd value used by the sender, which can change over the lifetime of a connection. cwnd will start at a low value and grow rapidly during the slow-start phase, then settle into a reasonably consistent range for the congestion-avoidance phase - assuming the underlying bandwidth-delay product (BDP) remains constant. Phenomena such as routing updates, link capacity changes or path load changes may modify the underlying BDP significantly; the cwnd should be expected to change accordingly, prompting the need for ACK rate updates.

TARR can also be used to suppress Delayed ACKs in order to allow measuring the RTT of each packet in specific intervals (e.g., during flow start-up), and allow a different ACK rate afterwards.

A Linux receiver has a heuristic to detect slow start and suppress Delayed ACKs just for that period. However, some slow start variants (e.g., HyStart, HyStart++, etc.) may alter the ending of slow start, thus confusing the heuristics of the receiver. To avoid slow start sender behavior ossification, an explicit signal such as TARR may be useful.

Another reason to modify the ACK rate might be reducing the ACK load. The sender may notice that the ACKs it receives cover more segments than the ACK rate requested, indicating that ACK decimation is occurring en route. The sender may then decide to reduce the ACK frequency to reduce receiver workload and network load up to the ACK decimation point.

Future TCP specifications may also permit Congestion Experienced (CE) marks to appear on pure ACKs [I-D.ietf-tcpm-generalized-ecn]. This might involve more frequent ACK rate updates (e.g., once an RTT), as the sender probes around an operating point.

## 6. IANA Considerations

This document specifies a new TCP option (TCP ACK Rate Request) that uses the shared experimental options format [RFC6994], with ExID in network-standard byte order.

The authors plan to request the allocation of ExID value 0x00AC for the TCP option specified in this document.

## 7. Security Considerations

The TARR option opens the door to new security threats. This section discusses such new threats, and suggests mitigation techniques.

An attacker might be able to impersonate a legitimate sender, and forge an apparently valid packet intended for the receiver. In such case, the attacker may mount a variety of harmful actions. By using TARR, the attacker may intentionally communicate a bad R value to the latter with the aim to damage communication or device performance. For example, in a small cwnd scenario, using a too high R value may lead to exacerbated RTT increase and throughput decrease. In other scenarios, a too low R value may contribute to depleting the energy of a battery-operated receiver at a faster rate or may lead to increased network packet load.

While Transport Layer Security (TLS) [RFC8446] is strongly recommended for securing TCP-based communication, TLS does not protect TCP headers, and thus cannot protect the TARR option fields carried by a segment. One approach to address the problem is using

network-layer protection, such as Internet Protocol Security (IPsec) [RFC4301]. Another solution is using the TCP Authentication Option (TCP-AO), which provides TCP segment integrity and protection against replay attacks [RFC5925].

While it is relatively hard for an off-path attacker to attack an unprotected TCP session, it is RECOMMENDED for a TARR receiver to use the guidance and attack mitigation given in [RFC5961]. The TARR option MUST be ignored on a packet that is deemed invalid.

A TARR receiver might opt not to fulfill a request to avoid or mitigate an attack by which a large number of senders request disabling delayed ACKs simultaneously and send a large number of data segments to the receiver (see Section 3.2).

## 8. Acknowledgments

Bob Briscoe, Jonathan Morton, Richard Scheffenegger, Neal Cardwell, Michael Tuexen, Yuchung Cheng, Matt Mathis, Jana Iyengar, Gorrry Fairhurst, Stuart Cheshire, Yoshifumi Nishida, Michael Scharf, Ian Swett, and Martin Duke provided useful comments and input for this document. Jana Iyengar suggested including a field to allow a sender communicate its tolerance to reordering. Jonathan Morton and Bob Briscoe provided the main input for Section 5.

Carles Gomez has been funded in part by the Spanish Government through project PID2019-106808RA-I00, and by Secretaria d'Universitats i Recerca del Departament d'Empresa i Coneixement de la Generalitat de Catalunya 2017 through grant SGR 376.

## 9. References

### 9.1. Normative References

- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.
- [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>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/info/rfc5681>>.



- [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <<https://www.rfc-editor.org/info/rfc5925>>.
- [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>>.
- [RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, DOI 10.17487/RFC6994, August 2013, <<https://www.rfc-editor.org/info/rfc6994>>.

## 9.2. Informative References

- [I-D.ietf-tcpm-generalized-ecn] Bagnulo, M. and B. Briscoe, "ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control Packets", Work in Progress, Internet-Draft, draft-ietf-tcpm-generalized-ecn-09, 31 January 2022, <<https://www.ietf.org/archive/id/draft-ietf-tcpm-generalized-ecn-09.txt>>.
- [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>>.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<https://www.rfc-editor.org/info/rfc4301>>.
- [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>>.
- [RFC8490] Bellis, R., Cheshire, S., Dickinson, J., Dickinson, S., Lemon, T., and T. Pusateri, "DNS Stateful Operations", RFC 8490, DOI 10.17487/RFC8490, March 2019, <<https://www.rfc-editor.org/info/rfc8490>>.

## Authors' Addresses

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

Email: carlesgo@entel.upc.edu

Jon Crowcroft  
University of Cambridge  
JJ Thomson Avenue  
Cambridge  
United Kingdom  
Email: jon.crowcroft@cl.cam.ac.uk

TCP Maintenance & Minor Extensions (tcpm)  
Internet-Draft  
Updates: 3168, 3449 (if approved)  
Intended status: Standards Track  
Expires: August 26, 2021

B. Briscoe  
Independent  
M. Kuehlewind  
Ericsson  
R. Scheffenegger  
NetApp  
February 22, 2021

More Accurate ECN Feedback in TCP  
draft-ietf-tcpm-accurate-ecn-14

Abstract

Explicit Congestion Notification (ECN) is a mechanism where network nodes can mark IP packets instead of dropping them to indicate incipient congestion to the end-points. Receivers with an ECN-capable transport protocol feed back this information to the sender. ECN is specified for TCP in such a way that only one feedback signal can be transmitted per Round-Trip Time (RTT). Recent new TCP mechanisms like Congestion Exposure (ConEx), Data Center TCP (DCTCP) or Low Latency Low Loss Scalable Throughput (L4S) need more accurate ECN feedback information whenever more than one marking is received in one RTT. This document specifies 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, that was previously used for the ECN-Nonce which has now been declared historic. It also overloads the two existing ECN flags in the TCP header. 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.

Status of This Memo

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

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

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

This Internet-Draft will expire on August 26, 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
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 . . . . .	10
3. AccECN Protocol Specification . . . . .	11
3.1. Negotiating to use AccECN . . . . .	11
3.1.1. Negotiation during the TCP handshake . . . . .	11
3.1.2. Backward Compatibility . . . . .	12
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 . . . . .	26
3.3. AccECN Compliance Requirements for TCP Proxies, Offload Engines and other Middleboxes . . . . .	35
3.3.1. Requirements for TCP Proxies . . . . .	35
3.3.2. Requirements for TCP Normalizers . . . . .	35
3.3.3. Requirements for TCP ACK Filtering . . . . .	35
3.3.4. Requirements for TCP Segmentation Offload . . . . .	36
4. Updates to RFC 3168 . . . . .	37

5.	Interaction with TCP Variants . . . . .	38
5.1.	Compatibility with SYN Cookies . . . . .	38
5.2.	Compatibility with TCP Experiments and Common TCP Options . . . . .	39
5.3.	Compatibility with Feedback Integrity Mechanisms . . . . .	39
6.	Protocol Properties . . . . .	40
7.	IANA Considerations . . . . .	43
8.	Security Considerations . . . . .	44
9.	Acknowledgements . . . . .	45
10.	Comments Solicited . . . . .	45
11.	References . . . . .	45
11.1.	Normative References . . . . .	45
11.2.	Informative References . . . . .	46
Appendix A.	Example Algorithms . . . . .	49
A.1.	Example Algorithm to Encode/Decode the AccECN Option . . . . .	49
A.2.	Example Algorithm for Safety Against Long Sequences of ACK Loss . . . . .	50
A.2.1.	Safety Algorithm without the AccECN Option . . . . .	50
A.2.2.	Safety Algorithm with the AccECN Option . . . . .	52
A.3.	Example Algorithm to Estimate Marked Bytes from Marked Packets . . . . .	54
A.4.	Example Algorithm to Beacon AccECN Options . . . . .	54
A.5.	Example Algorithm to Count Not-ECT Bytes . . . . .	55
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 . . . . .	57
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 to feed back the number of arriving CE marked packets. 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 (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 marked CE, ECT(1) and ECT(0) respectively. The fourth counts the number of packets arriving marked with a CE codepoint (including control packets without payload if they are CE-marked).

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

Each ACK carries the three least significant bits (LSBs) of the packet-based CE counter using the ECN bits in the TCP header, now renamed the Accurate ECN (ACE) field (see Figure 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 information about CE markings on both data and control packets. According to [RFC3168] the Data Sender is meant to set control packets to Not-ECT. However, mechanisms in certain private networks (e.g. data centres) set control packets to be ECN

capable because they are precisely the packets that performance depends on most.

For this reason, AccECN is designed to be a generic reflector of whatever ECN markings it sees, whether or not they are compliant with a current standard. Then as standards evolve, Data Senders can upgrade unilaterally without any need for receivers to upgrade too. It is also useful to be able to rely on generic reflection behaviour when senders need to test for unexpected interference with markings (for instance 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 ECN marking. 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 ECN field when it arrives at the receiver could still be useful, because middleboxes have been known to overwrite the ECN IP field as if it is still part of the old Type of Service (ToS) field [Mandalaril8]. If a TCP client has set the SYN to Not-ECT, but receives feedback that the ECN field on the SYN arrived with a different codepoint, it can detect such middlebox interference and send Not-ECT for the rest of the connection. Today, if a TCP server receives ECT or CE on a SYN, it cannot know whether it is invalid (or valid) because only the TCP client knows whether it originally marked the SYN as Not-ECT (or ECT). Therefore, prior to AccECN, the server's only safe course of action was to disable ECN for the connection. Instead, the AccECN protocol allows the server to feed back the received ECN field to the client, which then has all the information to decide whether the connection has to fall-back from supporting ECN (or not).

### 3. AccECN Protocol Specification

#### 3.1. Negotiating to use AccECN

##### 3.1.1. Negotiation during the TCP handshake

Given the ECN Nonce [RFC3540] has been reclassified as historic [RFC8311], the present specification re-allocates the TCP flag at bit 7 of the TCP header, which was previously called NS (Nonce Sum), as the AE (Accurate ECN) flag (see IANA Considerations in Section 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, without these rules, the two peers could end up using difference 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.

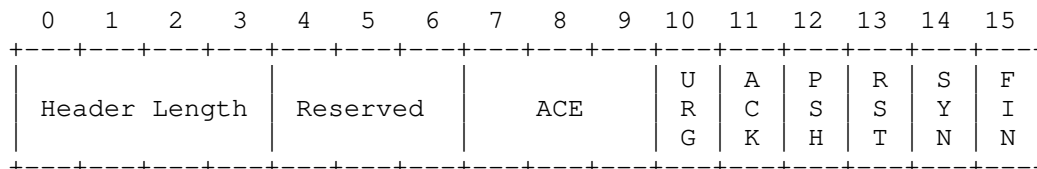


Figure 3: Definition of the ACE field within bytes 13 and 14 of the TCP Header (when AccECN has been negotiated and SYN=0).

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. The conditions for this to occur are quite unusual, but not impossible, 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. Nonetheless, once in the ESTABLISHED state, the server will consider the ACE field to be encoded as the normal ACE counter on all packets with SYN=0 (given it will be following the above rule in this bullet). The server MAY include a test to avoid this case.

#### 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 3WHS indicates the value of the IP/ECN field when the SYN/ACK arrived at the client. The server can compare this with how it originally set the IP/ECN field on the SYN/ACK. If this comparison implies an unsafe transition of the IP/ECN field, for the remainder of the connection the server MUST NOT send ECN-capable packets, but it MUST continue to 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 [Mandalaril18].

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 [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.

#### 3.2.2.5. 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

An AccECN Data Receiver:

- o SHOULD immediately send an ACK whenever a data packet marked CE arrives after the previous packet was not CE.
- o MUST immediately send an ACK once 'n' CE marks have arrived since the previous ACK, where 'n' SHOULD be 2 and MUST be in the range 2 to 6 inclusive.

These 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.

For the avoidance of doubt, the above change-triggered ACK mechanism is deliberately worded to solely apply to data packets, and to ignore the arrival of a control packet with no payload, because it is important that TCP does not acknowledge pure ACKs. The change-triggered ACK approach can lead to some additional ACKs but it feeds back the timing and the order in which ECN marks are received with minimal additional complexity. If only CE marks are infrequent, or

there are multiple marks in a row, the additional load will be low. Other marking patterns could increase the load significantly.

Even though the first bullet is stated as a "SHOULD", it is important for a transition to immediately trigger an ACK if at all possible, so that the Data Sender can rely on change-triggered ACKs to detect queue growth as soon as possible, e.g. at the start of a flow. This requirement can only be relaxed if certain offload hardware needed for high performance cannot support change-triggered ACKs (although high performance protocols such as DCTCP already successfully use change-triggered ACKs). 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.

#### 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'.

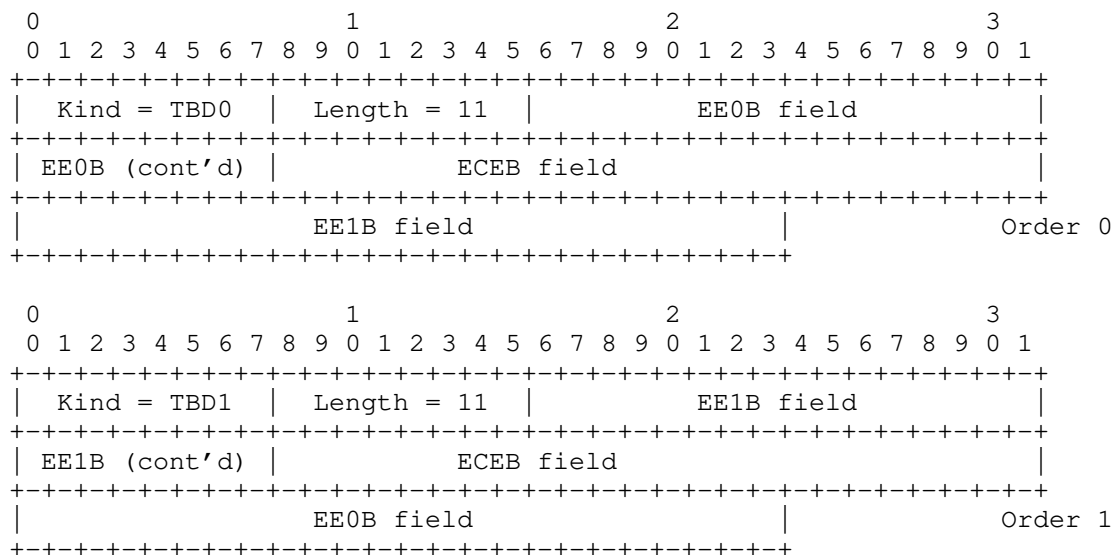


Figure 4: The AccECN TCP Option

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.5.

Whenever a Data Receiver sends an AccECN Option, the rules in Section 3.2.3.3 expect it to usually send a full-length option. To cope with option space limitations, it can omit unchanged fields from the tail of the option, as long as it preserves the order of the remaining fields and includes any field that has changed. The length field MUST indicate which fields are present as follows:

Length	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 **MUST** decode 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 locally held octet counters 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. (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 NOT include an AccECN Option in any of these three cases if it has cached knowledge that the packet would be likely to be blocked on the path to the other host if it included an AccECN Option.

#### 3.2.3.2.2. 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 the Data Receiver intends to use the AccECN TCP Option to provide feedback, the following rules determine when a Data Receiver in AccECN mode sends an ACK with the AccECN TCP Option, and which fields to include:

**Change-Triggered ACKs:** If an arriving packet increments a different byte counter to that incremented by the previous packet, the Data Receiver SHOULD immediately send an ACK with an AccECN Option, without waiting for the next delayed ACK (this is in addition to

the safety recommendation in Section 3.2.2.5 against ambiguity of the ACE field).

Even though this bullet is stated as a "SHOULD", it is important for a transition to immediately trigger an ACK if at all possible, as already argued when specifying change-triggered ACKs for the ACE.

Continual Repetition: Otherwise, if arriving packets continue to increment the same byte counter, the Data Receiver can include an AccECN Option on most or all (delayed) ACKs, but it does not have to.

- \* It SHOULD include a counter that has continued to increment on the next scheduled ACK following a change-triggered ACK;
- \* 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.

If the smallest allowed 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.

Necessary Option Length: It MAY exclude counter(s) that have not changed for the whole connection (but beacons still include all fields - see below). It SHOULD include counter(s) that have incremented at some time during the connection. It MUST include the counter(s) that have incremented since the previous AccECN Option and it MUST only truncate fields from the right-hand tail of the option to preserve the order of the remaining fields (see Section 3.2.3);

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

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

Even though first bullet is stated as a "SHOULD", it is important for a transition to immediately trigger an ACK if at all possible, so that the Data Sender can rely on change-triggered ACKs to detect queue growth as soon as possible, e.g. at the start of a flow. This requirement can only be relaxed if certain offload hardware needed for high performance cannot support change-triggered ACKs (although high performance protocols such as DCTCP already successfully use change-triggered ACKs). One possible experimental compromise would be for the receiver to heuristically detect whether the sender is in slow-start, then to implement change-triggered ACKs while the sender is in slow-start, and offload otherwise.

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

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

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

If a host has determined that segments with the AccECN Option always seem to be discarded somewhere along the path, it is no longer obliged to follow the above rules.

### 3.3. 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 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 and often outdated interpretation of the TCP specifications. To comply with the present AccECN specification, such a middlebox **MUST NOT** change the ACE field or the AccECN Option.

A middlebox claiming to be transparent at the transport layer **MUST** forward the AccECN TCP Option unaltered, whether or not the length value matches one of those specified in Section 3.2.3, and whether or not the initial values of the byte-counter fields are correct. This is because blocking apparently invalid values does not improve security (because AccECN hosts are required to ignore invalid values anyway), while it prevents the standardized set of values being extended in future (because outdated normalizers would block updated hosts from using the extended AccECN standard).

#### 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 DCTCP 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, DCTCP 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 and whether it needs to be a full-length option.

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.

## 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 a change to the Data Sender that is most useful when combined with AccECN. Without AccECN, the ConEx behaviour of a Data Sender would have to be more conservative than would be necessary if it had the accurate feedback of AccECN.

- o The TCP authentication option (TCP-AO [RFC5925]) can be used to detect any tampering with AccECN feedback between the Data Receiver and the Data Sender (whether malicious or accidental). The AccECN fields are immutable end-to-end, so they are amenable to TCP-AO protection, which covers TCP options by default. However, TCP-AO is often too brittle to use on many end-to-end paths, where middleboxes can make verification fail in their attempts to improve performance or security, e.g. by resegmentation or shifting the sequence space.

Originally the 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 3WHS so that it can fall back to operation without ECN and/or operation without the AccECN Option.

**Forward Compatibility:** The behaviour of endpoints and middleboxes is carefully defined for all reserved or currently unused codepoints in the scheme. 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 experiment. This bit was previously called the Nonce Sum (NS) flag [RFC3540], but RFC 3540 has been reclassified as historic [RFC8311]. The flag will now be defined as:

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 presented, 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 markings could be altered or suppressed, particularly because a misbehaving Data Receiver could increase its own throughput at the expense of others. AccECN is compatible with the three schemes known to assure the integrity of ECN feedback (see Section 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.

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. {ToDo: If necessary this 'forward compatibility' requirement can be capped or removed in a future revision, in order to narrow or close the covert channel. Comments are solicited on whether such a covert channel would be acceptable in TCP (given other TCP options already open up covert channels). And, if not, whether the channel should be narrowed or completely closed.}

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

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

## 9. Acknowledgements

We want to thank Koen De Schepper, Praveen Balasubramanian, Michael Welzl, Gorry Fairhurst, David Black, Spencer Dawkins, Michael Scharf, Michael Tuexen, Yuchung Cheng, Kenjiro Cho, Olivier Tilmans, Ilpo Jaervinen and Neal Cardwell 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-07 (work in progress), December 2020.
- [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-06 (work in progress), October 2020.
- [I-D.ietf-tsvwg-l4s-arch]  
Briscoe, B., Schepper, K., 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
```

#### 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 newlyAkedB, 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 ((newlyAkedB > 0) || (newlyAkedT > 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 row of ACKs goes missing that covers a stream of data long enough to contain 8 or more CE marks. We use the word 'missing' rather than 'lost', because some or all the missing ACKs might arrive eventually, but out of order. Even if some of the 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 newlyAkedB is the amount of data that an ACK acknowledges since the previous ACK, then the Data Sender could assume that this acknowledges newlyAkedPkt full-sized segments, where newlyAkedPkt = newlyAkedB/MSS. Then it could assume that the ACE field incremented by

```
dSafer.cep = newlyAkedPkt - ((newlyAkedPkt - d.cep) % DIVACE),
```

For example, imagine an ACK acknowledges newlyAkedPkt=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

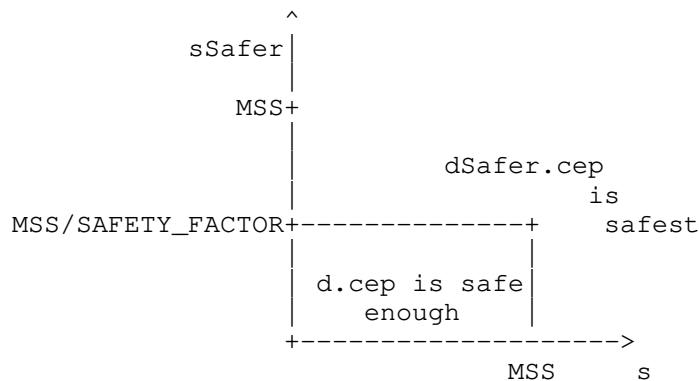
When the AccECN Option is available on the ACKs before and after the possible sequence of ACK losses, if the Data Sender only needs CE-marked bytes, it will have sufficient information in the AccECN Option without needing to process the ACE field. If for some reason it needs CE-marked packets, if dSafer.cep 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, dSafer.cep=15 and d.ceb=10200, then s=1457 and sSafer=680.

Therefore `d.ceb` 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.ceb` 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 Beacon AccECN Options

Section 3.2.3.3 requires a Data Receiver to beacon a full-length AccECN Option at least 3 times per RTT. This could be implemented by maintaining a variable to store the number of ACKs (pure and data



ACKs) since a full AccECN Option was last sent and another for the approximate number of ACKs sent in the last round trip time:

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

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

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

```
if ( time_now > time_last_option_sent + (RTT / BEACON_FREQ) )
    send_full_AccECN_Option()
```

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

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

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

For this approach to be precise, it has to be assumed that spurious (unnecessary) retransmissions do not lead to double counting. This assumption is currently correct, given that RFC 3168 requires that the Data Sender marks retransmitted segments as Not-ECT. However, the converse is not true; necessary 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 the experiment proves that a variant of AccECN is required, or if a completely different ECN feedback approach is needed:

Future AccECN variants: When the AccECN capability is negotiated during TCP's 3WHS, the rows in Table 2 tagged as 'Nonce' and 'Broken' in the column for the capability of node B are unused by any current protocol in the RFC series. These could be used by TCP servers in future to indicate a variant of the AccECN protocol. In recent measurement studies in which the response of large numbers of servers to an AccECN SYN has been tested, e.g. [Mandalaril18], 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.

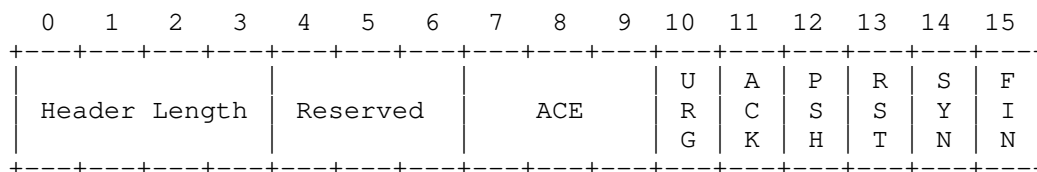


Figure 3: Definition of the ACE field within bytes 13 and 14 of the TCP Header (when AccECN has been negotiated and SYN=0).

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'.

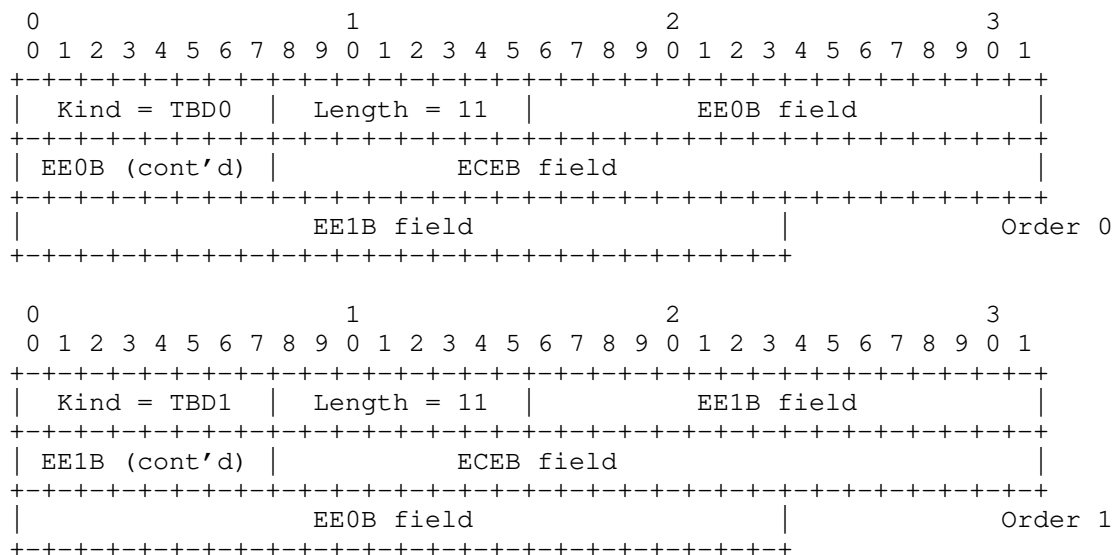


Figure 4: The AccECN TCP Option

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.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, Gorry Fairhurst, David Black, Spencer Dawkins, Michael Scharf, Michael Tuexen, Yuchung Cheng, Kenjiro Cho, Olivier Tilmans, Ilpo Jaervinen, Neal Cardwell, Yoshifumi Nishida, Martin Duke and Jonathan Morton for their input and discussion. The idea of using the three ECN-related TCP flags as one field for more accurate TCP-ECN feedback was first introduced in the re-ECN protocol that was the ancestor of ConEx.

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

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

## 10. Comments Solicited

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

## 11. References

### 11.1. Normative References

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

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

## 11.2. Informative References

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

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

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



## Appendix A. Example Algorithms

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

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

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

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

$$\text{DIVOPT} = 2^{24}$$

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

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

where `'%'` is the remainder operator.

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

```

if ((newlyAcedB > 0) || (newlyAcedT > 0)) {
    d.ceb = (ECEB + DIVOPT - (s.ceb % DIVOPT)) % DIVOPT
    s.ceb += d.ceb
}

```

For example, if s.ceb is 33,554,433 and ECEB is 1461 (both decimal), then

```

s.ceb % DIVOPT = 1
d.ceb = (1461 + 2^24 - 1) % 2^24
        = 1460
s.ceb = 33,554,433 + 1460
        = 33,555,893

```

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

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

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

Two variants of the algorithm are given: i) a more conservative variant for a Data Sender to use if it detects that the AccECN Option is not available (see Section 3.2.2.5 and Section 3.2.3.2); and ii) a less conservative variant that is feasible when complementary information is available from the AccECN Option.

##### A.2.1. Safety Algorithm without the AccECN Option

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

```
DIVACE = 2^3
```

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

ACE = r.cep % DIVACE.

If the Data Sender received an earlier value of the counter that had been delayed due to ACK reordering, it might incorrectly calculate that the ACE field had wrapped. Therefore, on the arrival of every ACK, the Data Sender ensures the ACK has not been superseded using the TCP acknowledgement number, any SACK options and timestamps (if available) to calculate newlyAckedB, as in Appendix A.1. If the ACK has not been superseded, the Data Sender calculates the minimum difference d.cep between the ACE field and its local s.cep counter, using modulo arithmetic as follows:

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

Section 3.2.2.5 expects the Data Sender to assume that the ACE field cycled if it is the safest likely case under prevailing conditions. The 3-bit ACE field in an arriving ACK could have cycled and become ambiguous to the Data Sender if a sequence of ACKs goes missing that covers a stream of data long enough to contain 8 or more CE marks. We use the word 'missing' rather than 'lost', because some or all the missing ACKs might arrive eventually, but out of order. Even if some of the missing ACKs were piggy-backed on data (i.e. not pure ACKs) retransmissions will not repair the lost AccECN information, because AccECN requires retransmissions to carry the latest AccECN counters, not the original ones.

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

```
dSafer.cep = newlyAckedPkt - ((newlyAckedPkt - d.cep) % DIVACE),
```

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

full-sized segments, then it would be necessary to assume that there could have been 10 CE marks (because  $10 - ((10-2) \% 8) = 10$ ).

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

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

Implementers could build in more heuristics to estimate prevailing average segment size and prevailing ECN marking. For instance, `newlyAkedPkt` in the above formula could be replaced with `newlyAkedPktHeur = newlyAkedPkt*p*MSS/s`, where `s` is the prevailing segment size and `p` is the prevailing ECN marking probability. However, ultimately, if TCP's ECN feedback becomes inaccurate it still has loss detection to fall back on. Therefore, it would seem safe to implement a simple algorithm, rather than a perfect one.

The simple algorithm for `dSafer.cep` above requires no monitoring of prevailing conditions and it would still be safe if, for example, segments were on average at least 5% of full-sized as long as ECN marking was 5% or less. Assuming it was used, the Data Sender would increment its packet counter as follows:

```
s.cep += dSafer.cep
```

If missing acknowledgement numbers arrive later (due to reordering), Section 3.2.2.5 says "the Data Sender MAY attempt to neutralize the effect of any action it took based on a conservative assumption that it later found to be incorrect". To do this, the Data Sender would have to store the values of all the relevant variables whenever it made assumptions, so that it could re-evaluate them later. Given this could become complex and it is not required, we do not attempt to provide an example of how to do this.

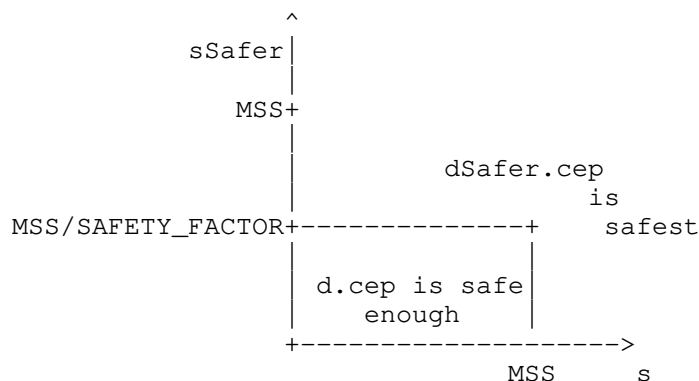
#### A.2.2. Safety Algorithm with the AccECN Option

When the AccECN Option is available on the ACKs before and after the possible sequence of ACK losses, if the Data Sender only needs CE-marked bytes, it will have sufficient information in the AccECN Option without needing to process the ACE field. If for some reason

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

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

The chart below shows when the above algorithm will consider `d.cep` can replace `dSafer.cep` as a safe enough estimate of the number of CE-marked packets:



The following examples give the reasoning behind the algorithm, assuming `MSS=1460 [B]`:

- o if `d.cep=0`, `dSafer.cep=8` and `d.ceb=1460`, then  $s=\text{infinity}$  and `sSafer=182.5`.  
Therefore even though the average size of 8 data segments is unlikely to have been as small as `MSS/8`, `d.cep` cannot have been correct, because it would imply an average segment size greater than the `MSS`.

- o if  $d_{cep}=2$ ,  $d_{safer_{cep}}=10$  and  $d_{ceb}=1460$ , then  $s=730$  and  $s_{safer}=146$ .  
Therefore  $d_{cep}$  is safe enough, because the average size of 10 data segments is unlikely to have been as small as  $MSS/10$ .
- o if  $d_{cep}=7$ ,  $d_{safer_{cep}}=15$  and  $d_{ceb}=10200$ , then  $s=1457$  and  $s_{safer}=680$ .  
Therefore  $d_{cep}$  is safe enough, because the average data segment size is more likely to have been just less than one MSS, rather than below  $MSS/2$ .

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

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

If the AccECN Option is not available, the Data Sender can only decode CE-marking from the ACE field in packets. Every time an ACK arrives, to convert this into an estimate of CE-marked bytes, it needs an average of the segment size,  $s_{ave}$ . Then it can add or subtract  $s_{ave}$  from the value of  $d_{ceb}$  as the value of  $d_{cep}$  increments or decrements. Some possible ways to calculate  $s_{ave}$  are outlined below. The precise details will depend on why an estimate of marked bytes is needed.

The implementation could keep a record of the byte numbers of all the boundaries between packets in flight (including control packets), and recalculate  $s_{ave}$  on every ACK. However it would be simpler to merely maintain a counter `packets_in_flight` for the number of packets in flight (including control packets), which is reset once per RTT. Either way, it would estimate  $s_{ave}$  as:

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

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

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

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

where  $a$  is the decay constant for the EWMA. However, then it is necessary to choose a good value for this constant, which ought to

depend on the number of packets in flight. Also the decay constant needs to be power of two to avoid floating point arithmetic.

#### A.4. Example Algorithm to Count Not-ECT Bytes

A Data Sender in AccECN mode can infer the amount of TCP payload data arriving at the receiver marked Not-ECT from the difference between the amount of newly ACKed data and the sum of the bytes with the other three markings, d.ceb, d.e0b and d.e1b.

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

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

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

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

AccECN uses a rather unorthodox approach to negotiate the highest version TCP ECN feedback scheme that both ends support, as justified below. It follows from the original TCP ECN capability negotiation [RFC3168], in which the client set the 2 least significant of the original reserved flags in the TCP header, and fell back to no ECN support if the server responded with the 2 flags cleared, which had previously been the default.

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

In order to be backward compatible with RFC 3168, AccECN continues this approach, using the 3rd least significant TCP header flag that had previously been allocated for the ECN nonce (now historic).

Then, whatever form of server an AccECN client encounters, the connection can fall back to the highest version of feedback protocol that both ends support, as explained in Section 3.1.

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

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

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

#### B.2. Four Codepoints in the SYN/ACK

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

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



on the SYN/ACK that had been reserved for future use by AccECN in earlier versions.

### B.3. Space for Future Evolution

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

**Future AccECN variants:** When the AccECN capability is negotiated during TCP's 3WHS, the rows in Table 2 tagged as 'Nonce' and 'Broken' in the column for the capability of node B are unused by any current protocol in the RFC series. These could be used by TCP servers in future to indicate a variant of the AccECN protocol. In recent measurement studies in which the response of large numbers of servers to an AccECN SYN has been tested, e.g. [Mandalaril8], a very small number of SYN/ACKs arrive with the pattern tagged as 'Nonce', and a small but more significant number arrive with the pattern tagged as 'Broken'. The 'Nonce' pattern could be a sign that a few servers have implemented the ECN Nonce [RFC3540], which has now been reclassified as historic [RFC8311], or it could be the random result of some unknown middlebox behaviour. The greater prevalence of the 'Broken' pattern suggests that some instances still exist of the broken code that reflects the reserved flags on the SYN.

The requirement not to reject unexpected initial values of the ACE counter (in the main TCP header) in the last para of Section 3.2.2.4 ensures that 3 unused codepoints on the ACK of the SYN/ACK, 6 unused values on the first SYN=0 data packet from the client and 7 unused values on the first SYN=0 data packet from the server could be used to declare future variants of the AccECN protocol. The word 'declare' is used rather than 'negotiate' because, at this late stage in the 3WHS, it would be too late for a negotiation between the endpoints to be completed. A similar requirement not to reject unexpected initial values in the TCP option (Section 3.2.3.2.4) is for the same purpose. If traversal of the TCP option were reliable, this would have enabled a far wider range of future variation of the whole AccECN protocol. Nonetheless, it could be used to reliably negotiate a wide range of variation in the semantics of the AccECN Option.

**Future non-AccECN variants:** Five codepoints out of the 8 possible in the 3 TCP header flags used by AccECN are unused on the initial SYN (in the order AE,CWR,ECE): 001, 010, 100, 101, 110. Section 3.1.3 ensures that the installed base of AccECN servers

will all assume these are equivalent to AccECN negotiation with 111 on the SYN. These codepoints would not allow fall-back to Classic ECN support for a server that did not understand them, but this approach ensures they are available in future, perhaps for uses other than ECN alongside the AccECN scheme. All possible combinations of SYN/ACK could be used in response except either 000 or reflection of the same values sent on the SYN.

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

#### Authors' Addresses

Bob Briscoe  
Independent  
UK

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

Mirja Kuehlewind  
Ericsson  
Germany

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

Richard Scheffenegger  
NetApp  
Vienna  
Austria

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

Network Working Group  
Internet-Draft  
Obsoletes: 5562 (if approved)  
Intended status: Experimental  
Expires: August 20, 2021

M. Bagnulo  
UC3M  
B. Briscoe  
Independent  
February 16, 2021

ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control  
Packets  
draft-ietf-tcpm-generalized-ecn-07

## Abstract

This document describes an experimental modification to ECN when used with TCP. It allows the use of ECN on the following TCP packets: SYNs, pure ACKs, Window probes, FINs, RSTs and retransmissions.

## Status of This Memo

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

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

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

This Internet-Draft will expire on August 20, 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.

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

## Table of Contents

1. Introduction . . . . .	3
1.1. Motivation . . . . .	4
1.2. Experiment Goals . . . . .	5
1.3. Document Structure . . . . .	6
2. Terminology . . . . .	6
3. Specification . . . . .	7
3.1. Network (e.g. Firewall) Behaviour . . . . .	7
3.2. Sender Behaviour . . . . .	8
3.2.1. SYN (Send) . . . . .	9
3.2.2. SYN-ACK (Send) . . . . .	13
3.2.3. Pure ACK (Send) . . . . .	14
3.2.4. Window Probe (Send) . . . . .	15
3.2.5. FIN (Send) . . . . .	16
3.2.6. RST (Send) . . . . .	16
3.2.7. Retransmissions (Send) . . . . .	17
3.2.8. General Fall-back for any Control Packet or Retransmission . . . . .	17
3.3. Receiver Behaviour . . . . .	18
3.3.1. Receiver Behaviour for Any TCP Control Packet or Retransmission . . . . .	18
3.3.2. SYN (Receive) . . . . .	18
3.3.3. Pure ACK (Receive) . . . . .	19
3.3.4. FIN (Receive) . . . . .	20
3.3.5. RST (Receive) . . . . .	20
3.3.6. Retransmissions (Receive) . . . . .	20
4. Rationale . . . . .	20
4.1. The Reliability Argument . . . . .	21
4.2. SYNs . . . . .	21
4.2.1. Argument 1a: Unrecognized CE on the SYN . . . . .	22
4.2.2. Argument 1b: ECT Considered Invalid on the SYN . . . . .	23
4.2.3. Caching Strategies for ECT on SYNs . . . . .	24
4.2.4. Argument 2: DoS Attacks . . . . .	27
4.3. SYN-ACKs . . . . .	28
4.3.1. Possibility of Unrecognized CE on the SYN-ACK . . . . .	28

4.3.2.	Response to Congestion on a SYN-ACK . . . . .	28
4.3.3.	Fall-Back if ECT SYN-ACK Fails . . . . .	29
4.4.	Pure ACKs . . . . .	30
4.4.1.	Mechanisms to Respond to CE-Marked Pure ACKs . . . . .	31
4.4.2.	Summary: Enabling ECN on Pure ACKs . . . . .	34
4.5.	Window Probes . . . . .	35
4.6.	FINs . . . . .	36
4.7.	RSTs . . . . .	36
4.8.	Retransmitted Packets. . . . .	37
4.9.	General Fall-back for any Control Packet . . . . .	38
5.	Interaction with popular variants or derivatives of TCP . . . . .	39
5.1.	IW10 . . . . .	39
5.2.	TFO . . . . .	40
5.3.	L4S . . . . .	41
5.4.	Other transport protocols . . . . .	41
6.	Security Considerations . . . . .	42
7.	IANA Considerations . . . . .	42
8.	Acknowledgments . . . . .	42
9.	References . . . . .	43
9.1.	Normative References . . . . .	43
9.2.	Informative References . . . . .	43
	Authors' Addresses . . . . .	47

## 1. Introduction

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

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

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

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

ECN++ is designed for compatibility with a number of latency improvements to TCP such as TCP Fast Open (TFO [RFC7413]), initial window of 10 SMSS (IW10 [RFC6928]) and Low latency Low Loss Scalable Transport (L4S [I-D.ietf-tsvwg-l4s-arch]), but they can all be implemented and deployed independently. [RFC8311] is a standards track procedural device that relaxes requirements in RFC 3168 and other standards track RFCs that would otherwise preclude the experimental modifications needed for ECN++ and other ECN experiments.

### 1.1. Motivation

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

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

it nearly impossible to establish a new connection in the presence of even mild traffic load.

Finally, there are ongoing experimental efforts to promote the adoption of a slightly modified variant of DCTCP (and similar congestion controls) over the Internet to achieve low latency, low loss and scalable throughput (L4S) for all communications [I-D.ietf-tsvwg-l4s-arch]. In such an approach, L4S packets identify themselves using an ECN codepoint [I-D.ietf-tsvwg-ecn-l4s-id]. With L4S, preventing TCP control packets from obtaining the benefits of ECN would not only expose them to the prevailing level of congestion loss, but it would also classify them into a different queue. Then only L4S data packets would be classified into the L4S queue that is expected to have lower latency, while the packets controlling and retransmitting these data packets would still get stuck behind the queue induced by non-L4S-enabled TCP traffic.

## 1.2. Experiment Goals

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

- o How SYNs, Window probes, pure ACKs, FINs, RSTs and retransmissions that carry the ECT(0), ECT(1) or CE codepoints are processed by the TCP endpoints and the network (including routers, firewalls and other middleboxes). In particular we would like to learn if these packets are frequently blocked or if these packets are usually forwarded and processed.
- o The scale of deployment of the different flavours of ECN, including [RFC3168], [RFC5562], [RFC3540] and [I-D.ietf-tcpm-accurate-ecn].
- o How much the performance of TCP communications is improved by allowing ECN marking of each packet type.
- o To identify any issues (including security issues) raised by enabling ECN marking of these packets.
- o To conduct the specific experiments identified in the text by the strings "EXPERIMENTATION NEEDED" or "MEASUREMENTS NEEDED".

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

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

### 1.3. Document Structure

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

## 2. Terminology

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

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

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

Window probe: Defined in [RFC0793], a window probe is a TCP segment with only one byte of data sent to learn if the receive window is still zero.

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

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

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



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

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

**ECT:** ECN-Capable Transport. One of the two codepoints ECT(0) or ECT(1) in the ECN field [RFC3168] of the IP header (v4 or v6). An ECN-capable sender sets one of these to indicate that both transport end-points support ECN. When this specification says the sender sets an ECT codepoint, by default it means ECT(0). Optionally, it could mean ECT(1), which is in the process of being redefined for use by L4S experiments [RFC8311] [I-D.ietf-tsvwg-ecn-l4s-id].

**Not-ECT:** The ECN codepoint set by senders that indicates that the transport is not ECN-capable.

**CE:** Congestion Experienced. The ECN codepoint that an intermediate node sets to indicate congestion [RFC3168]. A node sets an increasing proportion of ECT packets to CE as the level of congestion increases.

### 3. Specification

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

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

#### 3.1. Network (e.g. Firewall) Behaviour

Previously the specification of ECN for TCP [RFC3168] required the sender to set not-ECT on TCP control packets and retransmissions. Some readers of RFC 3168 might have erroneously interpreted this as a requirement for firewalls, intrusion detection systems, etc. to check and enforce this behaviour. Section 4.3 of [RFC8311] updates RFC 3168 to remove this ambiguity. It requires firewalls or any

intermediate nodes not to treat certain types of ECN-capable TCP segment differently (except potentially in one attack scenario). This is likely to only involve a firewall rule change in a fraction of cases (at most 0.4% of paths according to the tests reported in Section 4.2.2).

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

### 3.2. Sender Behaviour

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

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

Window probe and retransmission are abbreviated to W Probe and Re-XMT.

\* For a SYN, "negotiated" means "requested".

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

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

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

### 3.2.1. SYN (Send)

### 3.2.1.1. Setting ECT on the SYN

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

#### 3.2.1.1.1. ECN++ TCP Client also Supports AccECN

For the ECN++ experiment, if the SYN is requesting AccECN feedback, the TCP sender will also set ECT on the SYN. It can ignore the prohibition in section 6.1.1 of RFC 3168 against setting ECT on such a SYN, as per Section 4.3 of [RFC8311].

#### 3.2.1.1.2. ECN++ TCP Client does not Support AccECN

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

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

See Section 4.2 for discussion and rationale.

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

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

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

Until AccECN servers become widely deployed, a TCP initiator that sets ECT on a SYN (which typically implies the same SYN also requests AccECN, as above) SHOULD also maintain a cache entry per server to record servers that it is not worth sending an ECT SYN to, e.g. because they do not support AccECN and therefore have no logic for congestion markings on the SYN. Mobile hosts MAY maintain a cache

entry per access network to record 'non-ECT SYN' entries against proxies (see Section 4.2.3). This cache can be implemented as part of the shared state across multiple TCP connections, following [RFC2140].

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

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

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

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

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

#### 3.2.1.3. SYN Congestion Response

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

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

segment size). The rationale is the same as that for the response to CE on a SYN-ACK (Section 4.3.2).

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

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

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

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

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

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

An ECT SYN might be lost due to an over-zealous path element (or server) blocking ECT packets that do not conform to RFC 3168. Some evidence of this was found in a 2014 study [ecn-pam], but in a more recent study using 2017 data [Mandalari18] extensive measurements found no case where ECT on TCP control packets was treated any differently from ECT on TCP data packets. Loss is commonplace for numerous other reasons, e.g. congestion loss at a non-ECN queue on the forward or reverse path, transmission errors, etc. Alternatively, the cause of the loss might be the associated attempt to negotiate AccECN, or possibly other unrelated options on the SYN.

Therefore, if the timer expires after the TCP initiator has sent the first ECT SYN, it SHOULD make one more attempt to retransmit the SYN with ECT set (backing off the timer as usual). If the retransmission

timer expires again, it SHOULD retransmit the SYN with the not-ECT codepoint in the IP header, to expedite connection set-up. If other experimental fields or options were on the SYN, it will also be necessary to follow their specifications for fall-back too. It would make sense to coordinate all the strategies for fall-back in order to isolate the specific cause of the problem.

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

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

### 3.2.2. SYN-ACK (Send)

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

For the ECN++ experiment, the TCP implementation will set ECT on SYN-ACKs. It can ignore the requirement in section 6.1.1 of RFC 3168 to set not-ECT on a SYN-ACK, as per Section 4.3 of [RFC8311].

#### 3.2.2.2. SYN-ACK Congestion Response

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

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

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

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

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

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

The server MAY cache failed connection attempts, e.g. per client access network. A client-based alternative to caching at the server is given in Section 4.3.3. If the TCP server is caching failed connection attempts, it SHOULD NOT give up using ECT on the first SYN-ACK of subsequent connection attempts until it is clear that the blockage persistently and specifically affects ECT on SYN-ACKs. This is because loss is so commonplace for other reasons (see Section 3.2.1.4). If a cache is used, it SHOULD be arranged to expire so that the server will infrequently attempt to check whether the problem has been resolved.

### 3.2.3. Pure ACK (Send)

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

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

#### 3.2.3.1. Pure ACK without AccECN Feedback

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



### 3.2.3.2. Pure ACK with AccECN Feedback

For the ECN++ experiment, if AccECN has been successfully negotiated, either end of the connection will set ECT on Pure ACKs. They can ignore the requirement in section 6.1.4 of RFC 3168 to set not-ECT on a pure ACK, as per Section 4.3 of [RFC8311].

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

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

#### 3.2.3.2.1. Pure ACK Congestion Response

As explained above, this subsection only applies if AccECN has been successfully negotiated for the TCP connection.

A host that sets ECT on pure ACKs SHOULD respond to the congestion signal resulting from pure ACKs being marked with the CE codepoint. The specific response will need to be defined as an update to each congestion control specification. Possible responses to congestion feedback include reducing the congestion window (CWND) and/or regulating the pure ACK rate (see Section 4.4.1.1).

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

### 3.2.4. Window Probe (Send)

For the ECN++ experiment, the TCP sender will set ECT on window probes. It can ignore the prohibition in section 6.1.6 of RFC 3168 against setting ECT on a window probe, as per Section 4.3 of [RFC8311].

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

A receive window of zero indicates that the application is not consuming data fast enough and does not imply anything about network congestion. Once the receive window opens, the congestion window

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

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

### 3.2.5. FIN (Send)

A TCP implementation can set ECT on a FIN.

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

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

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

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

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

### 3.2.6. RST (Send)

A TCP implementation can set ECT on a RST.

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

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

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

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

### 3.2.7. Retransmissions (Send)

For the ECN++ experiment, the TCP sender will set ECT on retransmitted segments. It can ignore the prohibition in section 6.1.5 of RFC 3168 against setting ECT on retransmissions, as per Section 4.3 of [RFC8311].

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

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

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

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

Extensive measurements in fixed and mobile networks [Mandalaril18] have found no evidence of blockages due to ECT being set on any type of TCP control packet.

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

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

### 3.3. Receiver Behaviour

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

#### 3.3.1. Receiver Behaviour for Any TCP Control Packet or Retransmission

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

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

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

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

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

#### 3.3.2. SYN (Receive)

RFC 3168 negotiates the use of ECN for the connection end-to-end using the ECN flags in the TCP header. RFC 3168 originally said that "A host MUST NOT set ECT on SYN ... packets." but it was silent as to

what a TCP server ought to do if it receives a SYN packet with a non-zero IP/ECN field anyway.

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

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

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

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

### 3.3.3. Pure ACK (Receive)

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

- o Any TCP implementation SHOULD accept receipt of a pure ACK with a non-zero ECN field, despite current RFCs precluding the sending of such packets.
- o A TCP implementation taking part in the ECN++ experiment MUST accept receipt of a pure ACK with a non-zero ECN field.

The question of whether and how the receiver of pure ACKs is required to feed back any CE marks on them is outside the scope of the present specification because it is a matter for the relevant feedback specification ([RFC3168] or [I-D.ietf-tcpm-accurate-ecn]). AccECN feedback is required to count CE marking of any control packet including pure ACKs. Whereas RFC 3168 is silent on this point, so feedback of CE-markings might be implementation specific (see Section 4.4.1.1).

#### 3.3.4. FIN (Receive)

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

#### 3.3.5. RST (Receive)

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

#### 3.3.6. Retransmissions (Receive)

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

### 4. Rationale

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

#### 4.1. The Reliability Argument

Section 5.2 of RFC 3168 states:

"To ensure the reliable delivery of the congestion indication of the CE codepoint, an ECT codepoint MUST NOT be set in a packet unless the loss of that packet [at a subsequent node] in the network would be detected by the end nodes and interpreted as an indication of congestion."

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

The principle to determine whether a packet can be ECN-capable ought to be "do no extra harm", meaning that the reliability of a congestion signal's delivery ought to be no worse with ECN than without. In particular, setting the CE codepoint on the very same packet that would otherwise have been dropped fulfills this criterion, since either the packet is delivered and the CE signal is delivered to the endpoint, or the packet is dropped and the original congestion signal (packet loss) is delivered to the endpoint.

The concern about a CE marking being dropped at a subsequent node might be motivated by the idea that ECN-marking a packet at the first node does not remove the packet, so it could go on to worsen congestion at a subsequent node. However, it is not useful to reason about congestion by considering single packets. The departure rate from the first node will generally be the same (fully utilized) with or without ECN, so this argument does not apply.

#### 4.2. SYNs

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

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

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

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

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

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

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

1. An AccECN server uses the 3 'ECN' bits in the TCP header of the SYN-ACK to respond to the client. 4 of the possible 8 codepoints provide enough space for the server to feed back which of the 4 IP/ECN codepoints was on the incoming SYN (including CE of course).
2. If any of these 4 codepoints are in the SYN-ACK, it confirms that the server supports AccECN and, if another codepoint is returned, it confirms that the server doesn't support AccECN.

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

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



practical, such clients could reduce the initial window to say IW2 or IW3.

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

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

Given, until now, ECT-marked SYN packets have been prohibited, it cannot be assumed they will be accepted, by TCP middleboxes or servers.

##### 4.2.2.1. ECT on SYN Considered Invalid by Middleboxes

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

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

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

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

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

A study conducted in Nov 2017 [Kuehlewind18] found that, of the 82% of the Alexa top 50k web servers that supported ECN, 84% disabled ECN if the IP/ECN field on the SYN was ECT0, CE or either. Given most web servers use Linux, this behaviour can most likely be traced to a patch contributed in May 2012 that was first distributed in v3.5 of the Linux kernel [strict-ecn]. The comment says "RFC3168 : 6.1.1 SYN

packets must not have ECT/ECN bits set. If we receive a SYN packet with these bits set, it means a network is playing bad games with TOS bits. In order to avoid possible false congestion notifications, we disable TCP ECN negotiation." Of course, some of the 84% might be due to similar code in other OSs.

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

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

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

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

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

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

- (S1): Pessimistic ECT and cache successes: The initiator always requests AccECN, but by default without ECT on the SYN. Then it caches those servers that confirm that they support AccECN as 'ECT SYN OK'. On a subsequent connection to any server that supports AccECN, the initiator can then set ECT on the SYN. When connecting to other servers (non-ECN or classic

ECN) it will not set ECT on the SYN, so it will not fail the 'over-strict' ECN test.

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

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

A. No cache.

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

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

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

(S3): ECT by configuration: In a controlled environment, the administrator can make sure that servers support ECN-capable

SYN packets. Examples of controlled environments are single-tenant DCs, and possibly multi-tenant DCs if it is assumed that each tenant mostly communicates with its own VMs.

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

- o The "pessimistic ECT and cache successes" strategy (S1) suffers from exposing the initial SYN to the prevailing loss level, even if the server supports ECT on SYNs, but only on the first connection to each AccECN server. If AccECN becomes widely deployed on servers, SYNs to those AccECN servers that are less frequently used by the client and therefore don't fit in the cache will not benefit from ECN protection at all.
- o The "optimistic ECT without a cache" strategy (S2A) is the simplest. It would satisfy the goal of an implementer who is solely interested in low latency using AccECN and ECN++ and is not concerned about fall-back to Classic ECN.
- o The "optimistic ECT and cache failures" strategy (S2B) exploits ECT on SYNs from the very first attempt. But if the server turns out to be 'over-strict' it will disable ECN for the connection, but only for the first connection if it's one of the client's more popular servers that fits in the cache. If the server turns out not to support AccECN, the initiator has to conservatively limit its initial window, but again only for the first connection if it's one of the client's more popular servers (and anyway this rarely makes any difference when most client requests fit in a single packet).

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

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

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

#### 4.2.4. Argument 2: DoS Attacks

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

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

Mandating Not-ECT in an RFC does not stop attackers using ECT for flooding. Nonetheless, if a standard says SYNs are not meant to be ECT it would make it legitimate for firewalls to discard them. However this would negate the considerable benefit of ECT SYNs for compliant transports and seems unnecessary because RFC 3168 already provides the means to address this concern. In section 7, RFC 3168 says "During periods where ... the potential packet marking rate would be high, our recommendation is that routers drop packets rather than set the CE codepoint..." and this advice is repeated in [RFC7567] (section 4.2.1). This makes it harder for flooding packets to gain from ECT.

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

#### 4.3. SYN-ACKs

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

##### 4.3.1. Possibility of Unrecognized CE on the SYN-ACK

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

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

##### 4.3.2. Response to Congestion on a SYN-ACK

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

The present ECN++ experimental specification obsoletes RFC 5562 because it uses the ECN+ congestion response, not ECN+/TryOnce. First we argue against the rationale for ECN+/TryOnce given in sections 4.4 and 6.2 of [RFC5562]. It starts with a rather too literal interpretation of the requirement in RFC 3168 that says TCP's response to a single CE mark has to be "essentially the same as the congestion control response to a *single* dropped packet." TCP's response to a dropped initial (SYN or SYN-ACK) packet is to wait for the retransmission timer to expire (currently 1s). However, this

long delay assumes the worst case between two possible causes of the loss: a) heavy overload; or b) the normal capacity-seeking behaviour of other TCP flows. When the network is still delivering CE-marked packets, it implies that there is an AQM at the bottleneck and that it is not overloaded. This is because an AQM under overload will disable ECN (as recommended in section 7 of RFC 3168 and repeated in section 4.2.1 of RFC 7567). So scenario (a) can be ruled out. Therefore, TCP's response to a CE-marked SYN-ACK can be similar to its response to the loss of `_any_` packet, rather than backing off as if the special `_initial_` packet of a flow has been lost.

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

Therefore, we have to work out a reasoned argument for what to do. If an AQM is CE-marking packets, it implies there is already a queue and it is probably already somewhere around the AQM's operating point - it is unlikely to be well below and it might be well above. So, the more data packets that the client sends in its IW, the more likely at least one will be CE marked, leading it to exit slow-start early. On the other hand, it is highly unlikely that the SYN-ACK itself pushed the AQM into congestion, so it will be safe to introduce another single segment immediately (1 RTT after the SYN-ACK). Therefore, starting to probe for capacity with a slow start from an initial window of 1 segment seems appropriate to the circumstances. This is the approach adopted in Section 3.2.2.

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

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

An alternative to the server caching failed connection attempts would be for the server to rely on the client caching failed attempts (on the basis that the client would cache a failure whether ECT was blocked on the SYN or the SYN-ACK). This strategy cannot be used if the SYN does not request AccECN support. It works as follows: if the server receives a SYN that requests AccECN support but is set to not-ECT, it replies with a SYN-ACK also set to not-ECT. If a middlebox only blocks ECT on SYNs, not SYN-ACKs, this strategy might disable

ECN on a SYN-ACK when it did not need to, but at least it saves the server from maintaining a cache.

#### 4.4. Pure ACKs

Section 5.2 of RFC 3168 gives the following arguments for not allowing the ECT marking of pure ACKs (ACKs not piggy-backed on data):

"To ensure the reliable delivery of the congestion indication of the CE codepoint, an ECT codepoint MUST NOT be set in a packet unless the loss of that packet in the network would be detected by the end nodes and interpreted as an indication of congestion.

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

Later on, in section 6.1.4 it reads:

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

We next address each of the arguments presented above.

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

The second argument says that ECN ought not to be enabled unless there is a mechanism to respond to it. This argument actually comprises three sub-arguments:



Mechanism feasibility: If ECN is enabled on Pure ACKs, are there, or could there be, suitable mechanisms to detect, feed back and respond to ECN-marked Pure ACKs?

Do no extra harm: There has never been a mechanism to respond to loss of non-ECN Pure ACKs. So it seems that adding ECN without a response mechanism will do no extra harm to others, while improving a connection's own performance (because loss of an ACK holds back new data). However, if the end systems have no response mechanism, ECN Pure ACKs do slightly more harm than non-ECN, because the AQM doesn't immediately clear ECT packets from the queue until it reaches overload and disables ECN.

Standards policy: Even if there were no harm to others, does it set an undesirable precedent to allow a flow to use ECN to protect its Pure ACKs from loss, when there is no mechanism to respond to ECN-marking?

The last two arguments involve value judgements, but they both depend on the concrete technical question of mechanism feasibility, which will therefore be addressed first in Section 4.4.1 below. Then Section 4.4.2 draws conclusions by addressing the value judgements in the other two questions.

#### 4.4.1. Mechanisms to Respond to CE-Marked Pure ACKs

The question of whether the receiver of pure ACKs is required to detect and feed back any CE-marking is outside the scope of the present specification - it is a matter for the relevant feedback specification (classic ECN [RFC3168] and AccECN [I-D.ietf-tcpm-accurate-ecn]). The response to congestion feedback is also out of scope, because it would be defined in the base TCP congestion control specification [RFC5681] or its variants.

Nonetheless, in order to decide whether the present ECN++ experimental specification should require a host to set ECT on pure ACKs, we only need to know whether a response mechanism would be feasible - we do not have to standardize it. So the bullets below assess, for each type of feedback, whether the three stages of the congestion response mechanism could all work.

Detection: Can the receiver of a pure ACK detect a CE marking on it?:

- \* Classic feedback: RFC 3168 is silent on this point. The implementer of the receiver would not expect CE marks on pure ACKs, but the implementation might happen to check for CE marks

before it looks for the data. So detection will be implementation-dependent.

- \* AccECN feedback: the AccECN specification requires the receiver of any TCP packets to count any CE marks on them (whether or not it sends ECN-capable control packets itself).

Feedback: As a general rule, TCP does not ACK a pure ACK. However, even if the receiver of a CE-mark on a pure ACK does not feed it back immediately, it could still include it within subsequent feedback, for instance when it later sends a data segment (if it ever does):

- \* Classic feedback: RFC 3168 is silent on this point, so feedback of CE-markings might be implementation specific. If the receiver (of the pure ACKs) did generate feedback, it would set the echo congestion experienced (ECE) flag in the TCP header of subsequent packets in the round, as it would to feed back CE on data packets.
- \* AccECN feedback: the receiver continually feeds back a count of the number of CE-marked packets that it has received and, optionally, a count of CE-marked bytes. For either metric, AccECN takes into account all types of packets, including pure ACKs. CE-marked pure ACKs will solely increment the packet counter; not any byte counter, because by definition they contain no bytes of data.

Congestion response: In either case (classic or AccECN feedback), if the TCP sender does receive feedback about CE-markings on pure ACKs, it will be able to reduce the congestion window (cwnd) and/or the ACK rate.

Therefore a congestion response mechanism is clearly feasible if AccECN has been negotiated, but the position is unknown for the installed base of classic ECN feedback.

#### 4.4.1.1. Congestion Window Response to CE-Marked Pure ACKs

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

A CE-mark on a Pure ACK does not mean that only Pure ACKs are causing congestion. It only means that the marked Pure ACK is part of an aggregate that is collectively causing a bottleneck queue to randomly CE-mark a fraction of the packets. A CE-mark on a Pure ACK might be due to data packets in other flows through the same bottleneck, due

to data packets interspersed between Pure ACKs in the same half-connection, or just due to the rate of Pure ACKs alone. (RFC 3168 only considered the last possibility, which led to the argument that ECN-enabled Pure ACKs had to be deferred, because ACK congestion control was a research issue.)

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

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

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

Header bytes can be calculated by multiplying a packet count by a nominal header size, which is possible with AccECN feedback, because it gives a count of CE-marked packets (as well as CE-marked bytes). The above simple aggregate calculation caters for the full range of scenarios; from all Pure ACKs to just a few interspersed with data packets.

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

#### 4.4.1.2. ACK Rate Response to CE-Marked Pure ACKs

Reducing the congestion window will have no effect on the rate of pure ACKs. The worst case here is if the bottleneck is congested solely with pure ACKs, but it could also be problematic if a large fraction of the load was from unresponsive ACKs, leaving little or no capacity for the load from responsive data.

Since RFC 3168 was published, experimental Acknowledgement Congestion Control (AckCC) techniques have been documented in [RFC5690] (informational). So any pair of TCP end-points can choose to agree to regulate the delayed ACK ratio in response to lost or CE-marked pure ACKs. However, the protocol has a number of open issues concerning deployment (e.g. it requires support from both ends, it relies on two new TCP options, one of which is required on the SYN where option space is at a premium and, if either option is blocked by a middlebox, no fall-back behaviour is specified).

The new TCP options address two problems, namely that TCP had: i) no mechanism to allow ECT to be set on pure ACKs; and ii) no mechanism to feed back loss or CE-marking of pure ACKs. A combination of the present specification and AccECN addresses both these problems, at least for CE-marking. So it might now be possible to design an ECN-specific ACK congestion control scheme without the extra TCP options proposed in RFC 5690. However, such a mechanism is out of scope of the present document.

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

#### 4.4.2. Summary: Enabling ECN on Pure ACKs

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

In the case when Classic ECN has been negotiated, the argument for ECT on pure ACKs is less clear-cut. Some of the installed base of

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

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

#### 4.5. Window Probes

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

"If a window probe packet is dropped in the network, this loss is not detected by the receiver. Therefore, the TCP data sender MUST NOT set either an ECT codepoint or the CWR bit on window probe packets.

However, because window probes use exact sequence numbers, they cannot be easily spoofed in denial-of-service attacks. Therefore, if a window probe arrives with the CE codepoint set, then the receiver SHOULD respond to the ECN indications."

The reliability argument has already been addressed in Section 4.1.

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

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

#### 4.6. FINs

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

#### 4.7. RSTs

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

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

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

- o As explained above, a congestion response by the sender of a CE-marked RST message is not possible;
- o So the only reason for the sender setting ECT on a RST would be to improve the reliability of the message's delivery;
- o RST messages are used to both mount and mitigate attacks:
  - \* Spoofed RST messages are used by attackers to terminate ongoing connections, although the mitigations in RFC 5961 have considerably raised the bar against off-path RST attacks;
  - \* Legitimate RST messages allow endpoints to inform their peers to eliminate existing state that correspond to non existing connections, liberating resources e.g. in DoS attacks scenarios;

- o AQMs are advised to disable ECN marking during persistent overload, so:
  - \* it is harder for an attacker to exploit ECN to intensify an attack;
  - \* it is harder for a legitimate user to exploit ECN to more reliably mitigate an attack
- o Prohibiting ECT on a RST would deny the benefit of ECN to legitimate RST messages, but not to attackers who can disregard RFCs;
- o If ECT were prohibited on RSTs
  - \* it would be easy for security middleboxes to discard all ECN-capable RSTs;
  - \* However, unlike a SYN flood, it is already easy for a security middlebox (or host) to distinguish a RST flood from legitimate traffic [RFC5961], and even if a some legitimate RSTs are accidentally removed as well, legitimate connections still function.

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

#### 4.8. Retransmitted Packets.

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

The reliability argument has already been addressed in Section 4.1.

Protection against DoS attacks is not afforded by prohibiting ECT on retransmitted packets. An attacker can set CE on spoofed retransmissions whether or not it is prohibited by an RFC. Protection against the DoS attack described in section 6.1.5 of RFC 3168 is solely afforded by the requirement that "the TCP data receiver SHOULD ignore the CE codepoint on out-of-window packets". Therefore in Section 3.2.7 the sender is allowed to set ECT on retransmitted packets, in order to reduce the chance of them being

dropped. We also strengthen the receiver's requirement from "SHOULD ignore" to "MUST ignore". And we generalize the receiver's requirement to include failure of any validity check, not just out-of-window checks, in order to include the more stringent validity checks in RFC 5961 that have been developed since RFC 3168.

A consequence is that, for those retransmitted packets that arrive at the receiver after the original packet has been properly received (so-called spurious retransmissions), any CE marking will be ignored. There is no problem with that because the fact that the original packet has been delivered implies that the sender's original congestion response (when it deemed the packet lost and retransmitted it) was unnecessary.

Finally, the third argument is about over-reacting to congestion. The argument goes that, if a retransmitted packet is dropped, the sender will not detect it, so it will not react again to congestion (it would have reduced its congestion window already when it retransmitted the packet). Whereas, if retransmitted packets can be CE tagged instead of dropped, senders could potentially react more than once to congestion. However, we argue that it is legitimate to respond again to congestion if it still persists in subsequent round trip(s).

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

#### 4.9. General Fall-back for any Control Packet

Extensive experiments have found no evidence of any traversal problems with ECT on any TCP control packet [Mandalaril8]. Nonetheless, Sections 3.2.1.4 and 3.2.2.3 specify fall-back measures if ECT on the first packet of each half-connection (SYN or SYN-ACK) appears to be blocking progress. Here, the question of fall-back measures for ECT on other control packets is explored. It supports the advice given in Section 3.2.8; until there's evidence that something's broken, don't fix it.

If an implementation has had to disable ECT to ensure the first packet of a flow (SYN or SYN-ACK) gets through, the question arises whether it ought to disable ECT on all subsequent control packets within the same TCP connection. Without evidence of any such problems, this seems unnecessarily cautious. Particularly given it would be hard to detect loss of most other types of TCP control packets that are not ACK'd. And particularly given that unnecessarily removing ECT from other control packets could lead to performance problems, e.g. by directing them into another queue [I-D.ietf-tsvwg-ecn-l4s-id] or over a different path, because some



broken multipath equipment (erroneously) routes based on all 8 bits of the Diffserv field.

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

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

## 5. Interaction with popular variants or derivatives of TCP

The following subsections discuss any interactions between setting ECT on all packets and using the following popular variants of TCP: IW10 and TFO. It also briefly notes the possibility that the principles applied here should translate to protocols derived from TCP. This section is informative not normative, because no interactions have been identified that require any change to specifications. The subsection on IW10 discusses potential changes to specifications but recommends that no changes are needed.

The designs of the following TCP variants have also been assessed and found not to interact adversely with ECT on TCP control packets: SYN cookies (see Appendix A of [RFC4987] and section 3.1 of [RFC5562]), TCP Fast Open (TFO [RFC7413]) and L4S [I-D.ietf-tsvwg-l4s-arch].

### 5.1. IW10

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

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

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

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

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

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

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

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

## 5.2. TFO

TCP Fast Open (TFO [RFC7413]) is an experiment to remove the round trip delay of TCP's 3-way hand-shake (3WHS). A TFO initiator caches a cookie from a previous connection with a TFO-enabled server. Then, for subsequent connections to the same server, any data included on

the SYN can be passed directly to the server application, which can then return up to an initial window of response data on the SYN-ACK and on data segments straight after it, without waiting for the ACK that completes the 3WSHs.

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

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

### 5.3. L4S

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

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

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

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

### 5.4. Other transport protocols

Experience from experiments on adding ECN support to all TCP packets ought to be directly transferable between TCP and other transport protocols, like SCTP or QUIC.

Stream Control Transmission Protocol (SCTP [RFC4960]) is a standards track transport protocol derived from TCP. SCTP currently does not include ECN support, but Appendix A of RFC 4960 broadly describes how it would be supported and a (long-expired) draft on the addition of ECN to SCTP has been produced [I-D.stewart-tsvwg-sctpecn]. This draft avoided setting ECT on control packets and retransmissions, closely following the arguments in RFC 3168.

QUIC [I-D.ietf-quic-transport] is another standards track transport protocol offering similar services to TCP but intended to exploit some of the benefits of running over UDP. Building on the arguments in the current draft, a QUIC sender sets ECT(0) on all packets.

## 6. Security Considerations

Section 3.2.6 considers the question of whether ECT on RSTs will allow RST attacks to be intensified. There are several security arguments presented in RFC 3168 for preventing the ECN marking of TCP control packets and retransmitted segments. We believe all of them have been properly addressed in Section 4, particularly Section 4.2.4 and Section 4.8 on DoS attacks using spoofed ECT-marked SYNs and spoofed CE-marked retransmissions.

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

## 7. IANA Considerations

There are no IANA considerations in this memo.

## 8. Acknowledgments

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

The work of Marcelo Bagnulo has been performed in the framework of the H2020-ICT-2014-2 project 5G NORMA. His contribution reflects the consortium's view, but the consortium is not liable for any use that may be made of any of the information contained therein.

Bob Briscoe's contribution was partly funded by the Research Council of Norway through the TimeIn project, partly by CableLabs and partly

by the Comcast Innovation Fund. The views expressed here are solely those of the authors.

## 9. References

### 9.1. Normative References

- [I-D.ietf-tcpm-accurate-ecn]  
Briscoe, B., Kuehlewind, M., and R. Scheffenegger, "More Accurate ECN Feedback in TCP", draft-ietf-tcpm-accurate-ecn-13 (work in progress), November 2020.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC5961] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", RFC 5961, DOI 10.17487/RFC5961, August 2010, <<https://www.rfc-editor.org/info/rfc5961>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <<https://www.rfc-editor.org/info/rfc8311>>.

### 9.2. Informative References

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

[ecn-overload]

Steen, H., "Destruction Testing: Ultra-Low Delay using Dual Queue Coupled Active Queue Management", Masters Thesis, Uni Oslo , May 2017, <<https://www.duo.uio.no/bitstream/handle/10852/57424/thesis-henrste.pdf?sequence=1>>.

[ecn-pam] Trammell, B., Kuehlewind, M., Boppart, D., Learmonth, I., Fairhurst, G., and R. Scheffenegger, "Enabling Internet-Wide Deployment of Explicit Congestion Notification", Int'l Conf. on Passive and Active Network Measurement (PAM'15) pp193-205, 2015, <[https://link.springer.com/chapter/10.1007/978-3-319-15509-8\\_15](https://link.springer.com/chapter/10.1007/978-3-319-15509-8_15)>.

[ECN-PLUS]

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

[I-D.ietf-quic-transport]

Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-34 (work in progress), January 2021.

[I-D.ietf-tsvwg-ecn-l4s-id]

Schepper, K. and B. Briscoe, "Identifying Modified Explicit Congestion Notification (ECN) Semantics for Ultra-Low Queuing Delay (L4S)", draft-ietf-tsvwg-ecn-l4s-id-12 (work in progress), November 2020.

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

Briscoe, B., Schepper, K., 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.

[I-D.stewart-tsvwg-sctp-ecn]

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

[judd-nsdi]

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

[Kuehlewind18]

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

[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, <<https://ieeexplore.ieee.org/document/8316790>>.

[Manzoor17]

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

[PragueLinux]

Briscoe, B., De Schepper, K., Albisser, O., Misund, J., Tilmans, O., Kuehlewind, M., and A. Ahmed, "Implementing the 'TCP Prague' Requirements for Low Latency Low Loss Scalable Throughput (L4S)", Proc. Linux Netdev 0x13 , March 2019, <<https://www.netdevconf.org/0x13/session.html?talk-tcp-prague-l4s>>.

[relax-strict-ecn]

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

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.

[RFC2140] Touch, J., "TCP Control Block Interdependence", RFC 2140, DOI 10.17487/RFC2140, April 1997, <<https://www.rfc-editor.org/info/rfc2140>>.

[RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, DOI 10.17487/RFC3540, June 2003, <<https://www.rfc-editor.org/info/rfc3540>>.

- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.
- [RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <<https://www.rfc-editor.org/info/rfc4987>>.
- [RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", RFC 5562, DOI 10.17487/RFC5562, June 2009, <<https://www.rfc-editor.org/info/rfc5562>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/info/rfc5681>>.
- [RFC5690] Floyd, S., Arcia, A., Ros, D., and J. Iyengar, "Adding Acknowledgement Congestion Control to TCP", RFC 5690, DOI 10.17487/RFC5690, February 2010, <<https://www.rfc-editor.org/info/rfc5690>>.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <<https://www.rfc-editor.org/info/rfc6298>>.
- [RFC6928] Chu, J., Dukkkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <<https://www.rfc-editor.org/info/rfc6928>>.
- [RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <<https://www.rfc-editor.org/info/rfc7413>>.
- [RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <<https://www.rfc-editor.org/info/rfc7567>>.
- [RFC7661] Fairhurst, G., Sathiaselalan, A., and R. Secchi, "Updating TCP to Support Rate-Limited Traffic", RFC 7661, DOI 10.17487/RFC7661, October 2015, <<https://www.rfc-editor.org/info/rfc7661>>.



[RFC8257] Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <<https://www.rfc-editor.org/info/rfc8257>>.

[strict-ecn]

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

#### Authors' Addresses

Marcelo Bagnulo  
Universidad Carlos III de Madrid  
Av. Universidad 30  
Leganes, Madrid 28911  
SPAIN

Phone: 34 91 6249500  
Email: [marcelo@it.uc3m.es](mailto:marcelo@it.uc3m.es)  
URI: <http://www.it.uc3m.es>

Bob Briscoe  
Independent  
UK

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

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

M. Bagnulo  
UC3M  
B. Briscoe  
Independent  
31 January 2022

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

## Abstract

This document describes an experimental modification to ECN when used with TCP. It allows the use of ECN on the following TCP packets: SYNs, pure ACKs, Window probes, FINs, RSTs and retransmissions.

## Status of This Memo

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

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

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

This Internet-Draft will expire on 4 August 2022.

## Copyright Notice

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

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

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

## Table of Contents

1. Introduction . . . . .	3
1.1. Motivation . . . . .	4
1.2. Experiment Goals . . . . .	5
1.3. Document Structure . . . . .	6
2. Terminology . . . . .	6
3. Specification . . . . .	7
3.1. Network (e.g. Firewall) Behaviour . . . . .	8
3.2. Sender Behaviour . . . . .	8
3.2.1. SYN (Send) . . . . .	10
3.2.2. SYN-ACK (Send) . . . . .	13
3.2.3. Pure ACK (Send) . . . . .	14
3.2.4. Window Probe (Send) . . . . .	16
3.2.5. FIN (Send) . . . . .	16
3.2.6. RST (Send) . . . . .	17
3.2.7. Retransmissions (Send) . . . . .	17
3.2.8. General Fall-back for any Control Packet or Retransmission . . . . .	18
3.3. Receiver Behaviour . . . . .	18
3.3.1. Receiver Behaviour for Any TCP Control Packet or Retransmission . . . . .	18
3.3.2. SYN (Receive) . . . . .	19
3.3.3. Pure ACK (Receive) . . . . .	20
3.3.4. FIN (Receive) . . . . .	20
3.3.5. RST (Receive) . . . . .	20
3.3.6. Retransmissions (Receive) . . . . .	21
4. Rationale . . . . .	21
4.1. The Reliability Argument . . . . .	21
4.2. SYNs . . . . .	22
4.2.1. Argument 1a: Unrecognized CE on the SYN . . . . .	22
4.2.2. Argument 1b: ECT Considered Invalid on the SYN . . . . .	23
4.2.3. Caching Strategies for ECT on SYNs . . . . .	25
4.2.4. Argument 2: DoS Attacks . . . . .	27
4.3. SYN-ACKs . . . . .	28
4.3.1. Possibility of Unrecognized CE on the SYN-ACK . . . . .	28

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

## 1. Introduction

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

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

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

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

ECN++ is designed for compatibility with a number of latency improvements to TCP such as TCP Fast Open (TFO [RFC7413]), initial window of 10 SMSS (IW10 [RFC6928]) and Low latency Low Loss Scalable Transport (L4S [I-D.ietf-tsvwg-l4s-arch]), but they can all be implemented and deployed independently. [RFC8311] is a standards track procedural device that relaxes requirements in RFC 3168 and other standards track RFCs that would otherwise preclude the experimental modifications needed for ECN++ and other ECN experiments.

### 1.1. Motivation

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

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

Finally, there are ongoing experimental efforts to promote the adoption of a slightly modified variant of DCTCP (and similar congestion controls) over the Internet to achieve low latency, low loss and scalable throughput (L4S) for all communications [I-D.ietf-tsvwg-l4s-arch]. In such an approach, L4S packets identify themselves using an ECN codepoint [I-D.ietf-tsvwg-ecn-l4s-id]. With L4S, preventing TCP control packets from obtaining the benefits of ECN would not only expose them to the prevailing level of congestion loss, but it would also classify them into a different queue. Then only L4S data packets would be classified into the L4S queue that is expected to have lower latency, while the packets controlling and retransmitting these data packets would still get stuck behind the queue induced by non-L4S-enabled TCP traffic.

## 1.2. Experiment Goals

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

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

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

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

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

### 1.3. Document Structure

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

## 2. Terminology

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

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

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

Window probe: Defined in [RFC0793], a window probe is a TCP segment with only one byte of data sent to learn if the receive window is still zero.

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

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

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

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

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

ECT: ECN-Capable Transport. One of the two codepoints ECT(0) or ECT(1) in the ECN field [RFC3168] of the IP header (v4 or v6). An ECN-capable sender sets one of these to indicate that both transport end-points support ECN. When this specification says the sender sets an ECT codepoint, by default it means ECT(0). Optionally, it could mean ECT(1), which is in the process of being redefined for use by L4S experiments [RFC8311] [I-D.ietf-tsvwg-ecn-l4s-id].

Not-ECT: The ECN codepoint set by senders that indicates that the transport is not ECN-capable.

CE: Congestion Experienced. The ECN codepoint that an intermediate node sets to indicate congestion [RFC3168]. A node sets an increasing proportion of ECT packets to CE as the level of congestion increases.

### 3. Specification

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



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

### 3.1. Network (e.g. Firewall) Behaviour

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

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

### 3.2. Sender Behaviour

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

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

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

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

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

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

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

### 3.2.1. SYN (Send)

#### 3.2.1.1. Setting ECT on the SYN

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

##### 3.2.1.1.1. ECN++ TCP Client also Supports AccECN

For the ECN++ experiment, if the SYN is requesting AccECN feedback, the TCP sender will also set ECT on the SYN. It can ignore the prohibition in section 6.1.1 of RFC 3168 against setting ECT on such a SYN, as per Section 4.3 of [RFC8311].

##### 3.2.1.1.2. ECN++ TCP Client does not Support AccECN

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

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

See Section 4.2 for discussion and rationale.

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

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

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

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

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

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

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

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

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

### 3.2.1.3. SYN Congestion Response

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

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

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

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

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

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

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

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

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

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

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

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

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

### 3.2.2. SYN-ACK (Send)

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

For the ECN++ experiment, the TCP implementation will set ECT on SYN-ACKs. It can ignore the requirement in section 6.1.1 of RFC 3168 to set not-ECT on a SYN-ACK, as per Section 4.3 of [RFC8311].

#### 3.2.2.2. SYN-ACK Congestion Response

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

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

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

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

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

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

The server MAY cache failed connection attempts, e.g. per client access network. A client-based alternative to caching at the server is given in Section 4.3.3. If the TCP server is caching failed connection attempts, it SHOULD NOT give up using ECT on the first SYN-ACK of subsequent connection attempts until it is clear that the blockage persistently and specifically affects ECT on SYN-ACKs. This is because loss is so commonplace for other reasons (see Section 3.2.1.4). If a cache is used, it SHOULD be arranged to expire so that the server will infrequently attempt to check whether the problem has been resolved.

#### 3.2.3. Pure ACK (Send)

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

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

#### 3.2.3.1. Pure ACK without AccECN Feedback

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

#### 3.2.3.2. Pure ACK with AccECN Feedback

For the ECN++ experiment, if AccECN has been successfully negotiated, either end of the connection will set ECT on Pure ACKs. They can ignore the requirement in section 6.1.4 of RFC 3168 to set not-ECT on a pure ACK, as per Section 4.3 of [RFC8311].

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

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

##### 3.2.3.2.1. Pure ACK Congestion Response

As explained above, this subsection only applies if AccECN has been successfully negotiated for the TCP connection.

A host that sets ECT on pure ACKs SHOULD respond to the congestion signal resulting from pure ACKs being marked with the CE codepoint. The specific response will need to be defined as an update to each congestion control specification. Possible responses to congestion feedback include reducing the congestion window (CWND) and/or regulating the pure ACK rate (see Section 4.4.1.1).

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



#### 3.2.4. Window Probe (Send)

For the ECN++ experiment, the TCP sender will set ECT on window probes. It can ignore the prohibition in section 6.1.6 of RFC 3168 against setting ECT on a window probe, as per Section 4.3 of [RFC8311].

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

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

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

#### 3.2.5. FIN (Send)

A TCP implementation can set ECT on a FIN.

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

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

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

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

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

#### 3.2.6. RST (Send)

A TCP implementation can set ECT on a RST.

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

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

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

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

#### 3.2.7. Retransmissions (Send)

For the ECN++ experiment, the TCP sender will set ECT on retransmitted segments. It can ignore the prohibition in section 6.1.5 of RFC 3168 against setting ECT on retransmissions, as per Section 4.3 of [RFC8311].

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

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

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

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

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

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

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

### 3.3. Receiver Behaviour

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

#### 3.3.1. Receiver Behaviour for Any TCP Control Packet or Retransmission

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

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

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

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

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

### 3.3.2. SYN (Receive)

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

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

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

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

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

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

### 3.3.3. Pure ACK (Receive)

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

- \* Any TCP implementation SHOULD accept receipt of a pure ACK with a non-zero ECN field, despite current RFCs precluding the sending of such packets.
- \* A TCP implementation taking part in the ECN++ experiment MUST accept receipt of a pure ACK with a non-zero ECN field.

The question of whether and how the receiver of pure ACKs is required to feed back any CE marks on them is outside the scope of the present specification because it is a matter for the relevant feedback specification ([RFC3168] or [I-D.ietf-tcpm-accurate-ecn]). AccECN feedback is required to count CE marking of any control packet including pure ACKs. Whereas RFC 3168 is silent on this point, so feedback of CE-markings might be implementation specific (see Section 4.4.1.1).

### 3.3.4. FIN (Receive)

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

### 3.3.5. RST (Receive)

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

### 3.3.6. Retransmissions (Receive)

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

## 4. Rationale

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

### 4.1. The Reliability Argument

Section 5.2 of RFC 3168 states:

"To ensure the reliable delivery of the congestion indication of the CE codepoint, an ECT codepoint MUST NOT be set in a packet unless the loss of that packet [at a subsequent node] in the network would be detected by the end nodes and interpreted as an indication of congestion."

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

The principle to determine whether a packet can be ECN-capable ought to be "do no extra harm", meaning that the reliability of a congestion signal's delivery ought to be no worse with ECN than without. In particular, setting the CE codepoint on the very same packet that would otherwise have been dropped fulfills this criterion, since either the packet is delivered and the CE signal is delivered to the endpoint, or the packet is dropped and the original congestion signal (packet loss) is delivered to the endpoint.

The concern about a CE marking being dropped at a subsequent node might be motivated by the idea that ECN-marking a packet at the first node does not remove the packet, so it could go on to worsen congestion at a subsequent node. However, it is not useful to reason about congestion by considering single packets. The departure rate from the first node will generally be the same (fully utilized) with or without ECN, so this argument does not apply.

#### 4.2. SYNs

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

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

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

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

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

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

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

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

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

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

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

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

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

Given, until now, ECT-marked SYN packets have been prohibited, it cannot be assumed they will be accepted, by TCP middleboxes or servers.

##### 4.2.2.1. ECT on SYN Considered Invalid by Middleboxes

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

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

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



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

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

A study conducted in Nov 2017 [Kuehlewindl8] found that, of the 82% of the Alexa top 50k web servers that supported ECN, 84% disabled ECN if the IP/ECN field on the SYN was ECT0, CE or either. Given most web servers use Linux, this behaviour can most likely be traced to a patch contributed in May 2012 that was first distributed in v3.5 of the Linux kernel [strict-ecn]. The comment says "RFC3168 : 6.1.1 SYN packets must not have ECT/ECN bits set. If we receive a SYN packet with these bits set, it means a network is playing bad games with TOS bits. In order to avoid possible false congestion notifications, we disable TCP ECN negotiation." Of course, some of the 84% might be due to similar code in other OSs.

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

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

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

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

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

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

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

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

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

a. No cache.

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

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

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

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

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

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

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

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

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

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

#### 4.2.4. Argument 2: DoS Attacks

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

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

Mandating Not-ECT in an RFC does not stop attackers using ECT for flooding. Nonetheless, if a standard says SYNs are not meant to be ECT it would make it legitimate for firewalls to discard them. However this would negate the considerable benefit of ECT SYNs for compliant transports and seems unnecessary because RFC 3168 already provides the means to address this concern. In section 7, RFC 3168 says "During periods where ... the potential packet marking rate would be high, our recommendation is that routers drop packets rather than set the CE codepoint..." and this advice is repeated in [RFC7567] (section 4.2.1). This makes it harder for flooding packets to gain from ECT.

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

#### 4.3. SYN-ACKs

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

##### 4.3.1. Possibility of Unrecognized CE on the SYN-ACK

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

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

#### 4.3.2. Response to Congestion on a SYN-ACK

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

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

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

Therefore, we have to work out a reasoned argument for what to do. If an AQM is CE-marking packets, it implies there is already a queue and it is probably already somewhere around the AQM's operating point - it is unlikely to be well below and it might be well above. So, the more data packets that the client sends in its IW, the more likely at least one will be CE marked, leading it to exit slow-start early. On the other hand, it is highly unlikely that the SYN-ACK itself pushed the AQM into congestion, so it will be safe to introduce another single segment immediately (1 RTT after the SYN-ACK). Therefore, starting to probe for capacity with a slow start from an initial window of 1 segment seems appropriate to the circumstances. This is the approach adopted in Section 3.2.2.

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

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

An alternative to the server caching failed connection attempts would be for the server to rely on the client caching failed attempts (on the basis that the client would cache a failure whether ECT was blocked on the SYN or the SYN-ACK). This strategy cannot be used if the SYN does not request AccECN support. It works as follows: if the server receives a SYN that requests AccECN support but is set to not-ECT, it replies with a SYN-ACK also set to not-ECT. If a middlebox only blocks ECT on SYNs, not SYN-ACKs, this strategy might disable ECN on a SYN-ACK when it did not need to, but at least it saves the server from maintaining a cache.

#### 4.4. Pure ACKs

Section 5.2 of RFC 3168 gives the following arguments for not allowing the ECT marking of pure ACKs (ACKs not piggy-backed on data):

"To ensure the reliable delivery of the congestion indication of the CE codepoint, an ECT codepoint MUST NOT be set in a packet unless the loss of that packet in the network would be detected by the end nodes and interpreted as an indication of congestion.

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

Later on, in section 6.1.4 it reads:

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

We next address each of the arguments presented above.

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

The second argument says that ECN ought not to be enabled unless there is a mechanism to respond to it. This argument actually comprises three sub-arguments:

Mechanism feasibility: If ECN is enabled on Pure ACKs, are there, or could there be, suitable mechanisms to detect, feed back and respond to ECN-marked Pure ACKs?

Do no extra harm: There has never been a mechanism to respond to loss of non-ECN Pure ACKs. So it seems that adding ECN without a response mechanism will do no extra harm to others, while improving a connection's own performance (because loss of an ACK



holds back new data). However, if the end systems have no response mechanism, ECN Pure ACKs do slightly more harm than non-ECN, because the AQM doesn't immediately clear ECT packets from the queue until it reaches overload and disables ECN.

Standards policy: Even if there were no harm to others, does it set an undesirable precedent to allow a flow to use ECN to protect its Pure ACKs from loss, when there is no mechanism to respond to ECN-marking?

The last two arguments involve value judgements, but they both depend on the concrete technical question of mechanism feasibility, which will therefore be addressed first in Section 4.4.1 below. Then Section 4.4.2 draws conclusions by addressing the value judgements in the other two questions.

#### 4.4.1. Mechanisms to Respond to CE-Marked Pure ACKs

The question of whether the receiver of pure ACKs is required to detect and feed back any CE-marking is outside the scope of the present specification - it is a matter for the relevant feedback specification (classic ECN [RFC3168] and AccECN [I-D.ietf-tcpm-accurate-ecn]). The response to congestion feedback is also out of scope, because it would be defined in the base TCP congestion control specification [RFC5681] or its variants.

Nonetheless, in order to decide whether the present ECN++ experimental specification should require a host to set ECT on pure ACKs, we only need to know whether a response mechanism would be feasible - we do not have to standardize it. So the bullets below assess, for each type of feedback, whether the three stages of the congestion response mechanism could all work.

Detection: Can the receiver of a pure ACK detect a CE marking on it?:

- \* Classic feedback: RFC 3168 is silent on this point. The implementer of the receiver would not expect CE marks on pure ACKs, but the implementation might happen to check for CE marks before it looks for the data. So detection will be implementation-dependent.
- \* AccECN feedback: the AccECN specification requires the receiver of any TCP packets to count any CE marks on them (whether or not it sends ECN-capable control packets itself).

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

even if the receiver of a CE-mark on a pure ACK does not feed it back immediately, it could still include it within subsequent feedback, for instance when it later sends a data segment (if it ever does):

- \* Classic feedback: RFC 3168 is silent on this point, so feedback of CE-markings might be implementation specific. If the receiver (of the pure ACKs) did generate feedback, it would set the echo congestion experienced (ECE) flag in the TCP header of subsequent packets in the round, as it would to feed back CE on data packets.
- \* AccECN feedback: the receiver continually feeds back a count of the number of CE-marked packets that it has received and, optionally, a count of CE-marked bytes. For either metric, AccECN takes into account all types of packets, including pure ACKs. CE-marked pure ACKs will solely increment the packet counter; not any byte counter, because by definition they contain no bytes of data.

Congestion response: In either case (classic or AccECN feedback), if the TCP sender does receive feedback about CE-markings on pure ACKs, it will be able to reduce the congestion window (cwnd) and/or the ACK rate.

Therefore a congestion response mechanism is clearly feasible if AccECN has been negotiated, but the position is unknown for the installed base of classic ECN feedback.

#### 4.4.1.1. Congestion Window Response to CE-Marked Pure ACKs

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

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

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

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

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

Header bytes can be calculated by multiplying a packet count by a nominal header size, which is possible with AccECN feedback, because it gives a count of CE-marked packets (as well as CE-marked bytes). The above simple aggregate calculation caters for the full range of scenarios; from all Pure ACKs to just a few interspersed with data packets.

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

#### 4.4.1.2. ACK Rate Response to CE-Marked Pure ACKs

Reducing the congestion window will have no effect on the rate of pure ACKs. The worst case here is if the bottleneck is congested solely with pure ACKs, but it could also be problematic if a large fraction of the load was from unresponsive ACKs, leaving little or no capacity for the load from responsive data.

Since RFC 3168 was published, experimental Acknowledgement Congestion Control (AckCC) techniques have been documented in [RFC5690] (informational). So any pair of TCP end-points can choose to agree to regulate the delayed ACK ratio in response to lost or CE-marked pure ACKs. However, the protocol has a number of open issues concerning deployment (e.g. it requires support from both ends, it relies on two new TCP options, one of which is required on the SYN where option space is at a premium and, if either option is blocked by a middlebox, no fall-back behaviour is specified).

The new TCP options address two problems, namely that TCP had: i) no mechanism to allow ECT to be set on pure ACKs; and ii) no mechanism to feed back loss or CE-marking of pure ACKs. A combination of the present specification and AccECN addresses both these problems, at least for CE-marking. So it might now be possible to design an ECN-specific ACK congestion control scheme without the extra TCP options proposed in RFC 5690. However, such a mechanism is out of scope of the present document.

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

#### 4.4.2. Summary: Enabling ECN on Pure ACKs

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

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

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

#### 4.5. Window Probes

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

"If a window probe packet is dropped in the network, this loss is not detected by the receiver. Therefore, the TCP data sender **MUST NOT** set either an ECT codepoint or the CWR bit on window probe packets.

However, because window probes use exact sequence numbers, they cannot be easily spoofed in denial-of-service attacks. Therefore, if a window probe arrives with the CE codepoint set, then the receiver **SHOULD** respond to the ECN indications."

The reliability argument has already been addressed in Section 4.1.

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

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

#### 4.6. FINs

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

#### 4.7. RSTs

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

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

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

- \* As explained above, a congestion response by the sender of a CE-marked RST message is not possible;
- \* So the only reason for the sender setting ECT on a RST would be to improve the reliability of the message's delivery;
- \* RST messages are used to both mount and mitigate attacks:
  - Spoofed RST messages are used by attackers to terminate ongoing connections, although the mitigations in RFC 5961 have considerably raised the bar against off-path RST attacks;
  - Legitimate RST messages allow endpoints to inform their peers to eliminate existing state that correspond to non existing connections, liberating resources e.g. in DoS attacks scenarios;

- \* AQMs are advised to disable ECN marking during persistent overload, so:
  - it is harder for an attacker to exploit ECN to intensify an attack;
  - it is harder for a legitimate user to exploit ECN to more reliably mitigate an attack
- \* Prohibiting ECT on a RST would deny the benefit of ECN to legitimate RST messages, but not to attackers who can disregard RFCs;
- \* If ECT were prohibited on RSTs
  - it would be easy for security middleboxes to discard all ECN-capable RSTs;
  - However, unlike a SYN flood, it is already easy for a security middlebox (or host) to distinguish a RST flood from legitimate traffic [RFC5961], and even if a some legitimate RSTs are accidentally removed as well, legitimate connections still function.

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

#### 4.8. Retransmitted Packets.

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

The reliability argument has already been addressed in Section 4.1.

Protection against DoS attacks is not afforded by prohibiting ECT on retransmitted packets. An attacker can set CE on spoofed retransmissions whether or not it is prohibited by an RFC. Protection against the DoS attack described in section 6.1.5 of RFC 3168 is solely afforded by the requirement that "the TCP data receiver SHOULD ignore the CE codepoint on out-of-window packets". Therefore in Section 3.2.7 the sender is allowed to set ECT on retransmitted packets, in order to reduce the chance of them being

dropped. We also strengthen the receiver's requirement from "SHOULD ignore" to "MUST ignore". And we generalize the receiver's requirement to include failure of any validity check, not just out-of-window checks, in order to include the more stringent validity checks in RFC 5961 that have been developed since RFC 3168.

A consequence is that, for those retransmitted packets that arrive at the receiver after the original packet has been properly received (so-called spurious retransmissions), any CE marking will be ignored. There is no problem with that because the fact that the original packet has been delivered implies that the sender's original congestion response (when it deemed the packet lost and retransmitted it) was unnecessary.

Finally, the third argument is about over-reacting to congestion. The argument goes that, if a retransmitted packet is dropped, the sender will not detect it, so it will not react again to congestion (it would have reduced its congestion window already when it retransmitted the packet). Whereas, if retransmitted packets can be CE tagged instead of dropped, senders could potentially react more than once to congestion. However, we argue that it is legitimate to respond again to congestion if it still persists in subsequent round trip(s).

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

#### 4.9. General Fall-back for any Control Packet

Extensive experiments have found no evidence of any traversal problems with ECT on any TCP control packet [Mandalaril8]. Nonetheless, Sections 3.2.1.4 and 3.2.2.3 specify fall-back measures if ECT on the first packet of each half-connection (SYN or SYN-ACK) appears to be blocking progress. Here, the question of fall-back measures for ECT on other control packets is explored. It supports the advice given in Section 3.2.8; until there's evidence that something's broken, don't fix it.



If an implementation has had to disable ECT to ensure the first packet of a flow (SYN or SYN-ACK) gets through, the question arises whether it ought to disable ECT on all subsequent control packets within the same TCP connection. Without evidence of any such problems, this seems unnecessarily cautious. Particularly given it would be hard to detect loss of most other types of TCP control packets that are not ACK'd. And particularly given that unnecessarily removing ECT from other control packets could lead to performance problems, e.g. by directing them into another queue [I-D.ietf-tsvwg-ecn-l4s-id] or over a different path, because some broken multipath equipment (erroneously) routes based on all 8 bits of the Diffserv field.

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

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

## 5. Interaction with popular variants or derivatives of TCP

The following subsections discuss any interactions between setting ECT on all packets and using the following popular variants of TCP: IW10 and TFO. It also briefly notes the possibility that the principles applied here should translate to protocols derived from TCP. This section is informative not normative, because no interactions have been identified that require any change to specifications. The subsection on IW10 discusses potential changes to specifications but recommends that no changes are needed.

The designs of the following TCP variants have also been assessed and found not to interact adversely with ECT on TCP control packets: SYN cookies (see Appendix A of [RFC4987] and section 3.1 of [RFC5562]), TCP Fast Open (TFO [RFC7413]) and L4S [I-D.ietf-tsvwg-l4s-arch].

## 5.1. IW10

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

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

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

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

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

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

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

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

## 5.2. TFO

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

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

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

## 5.3. L4S

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

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

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

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

#### 5.4. Other transport protocols

Experience from experiments on adding ECN support to all TCP packets ought to be directly transferable between TCP and other transport protocols, like SCTP or QUIC.

Stream Control Transmission Protocol (SCTP [RFC4960]) is a standards track transport protocol derived from TCP. SCTP currently does not include ECN support, but Appendix A of RFC 4960 broadly describes how it would be supported and a (long-expired) draft on the addition of ECN to SCTP has been produced [I-D.stewart-tsvwg-sctpecn]. This draft avoided setting ECT on control packets and retransmissions, closely following the arguments in RFC 3168.

QUIC [RFC9000] is another standards track transport protocol offering similar services to TCP but intended to exploit some of the benefits of running over UDP. Building on the arguments in the current draft, a QUIC sender sets ECT(0) on all packets.

#### 6. Security Considerations

Section 3.2.6 considers the question of whether ECT on RSTs will allow RST attacks to be intensified. There are several security arguments presented in RFC 3168 for preventing the ECN marking of TCP control packets and retransmitted segments. We believe all of them have been properly addressed in Section 4, particularly Section 4.2.4 and Section 4.8 on DoS attacks using spoofed ECT-marked SYNs and spoofed CE-marked retransmissions.

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

#### 7. IANA Considerations

There are no IANA considerations in this memo.

## 8. Acknowledgments

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

The work of Marcelo Bagnulo has been performed in the framework of the H2020-ICT-2014-2 project 5G NORMA. His contribution reflects the consortium's view, but the consortium is not liable for any use that may be made of any of the information contained therein.

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

## 9. References

### 9.1. Normative References

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

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

## 9.2. Informative References

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

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

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



## [Kuehlewind18]

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

## [strict-ecn]

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

## [relax-strict-ecn]

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

## [ecn-overload]

Steen, H., "Destruction Testing: Ultra-Low Delay using Dual Queue Coupled Active Queue Management", Masters Thesis, Uni Oslo , May 2017, <<https://www.duo.uio.no/bitstream/handle/10852/57424/thesis-henrste.pdf?sequence=1>>.

## [PragueLinux]

Briscoe, B., De Schepper, K., Albisser, O., Misund, J., Tilmans, O., Kühlewind, M., and A.S. Ahmed, "Implementing the 'TCP Prague' Requirements for Low Latency Low Loss Scalable Throughput (L4S)", Proc. Linux Netdev 0x13 , March 2019, <<https://www.netdevconf.org/0x13/session.html?talk-tcp-prague-l4s>>.

## [DOCSIS3.1]

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

## Authors' Addresses

Marcelo Bagnulo  
Universidad Carlos III de Madrid  
Av. Universidad 30  
28911 Leganes Madrid  
Spain

Phone: 34 91 6249500

Email: [marcelo@it.uc3m.es](mailto:marcelo@it.uc3m.es)  
URI: <http://www.it.uc3m.es>

Bob Briscoe  
Independent  
United Kingdom

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

TCPM  
Internet-Draft  
Intended status: Standards Track  
Expires: May 20, 2021

M. Scharf  
Hochschule Esslingen  
V. Murgai  
Samsung  
M. Jethanandani  
Kloud Services  
November 16, 2020

YANG Model for Transmission Control Protocol (TCP) Configuration  
draft-ietf-tcpm-yang-tcp-01

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 May 20, 2021.

Copyright Notice

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

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

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

## Table of Contents

1. Introduction . . . . .	2
2. Requirements Language . . . . .	3
2.1. Note to RFC Editor . . . . .	3
3. Model Overview . . . . .	4
3.1. Modeling Scope . . . . .	4
3.2. Model Design . . . . .	5
3.3. Tree Diagram . . . . .	6
4. TCP YANG Model . . . . .	6
5. IANA Considerations . . . . .	13
5.1. The IETF XML Registry . . . . .	13
5.2. The YANG Module Names Registry . . . . .	13
6. Security Considerations . . . . .	14
7. References . . . . .	15
7.1. Normative References . . . . .	15
7.2. Informative References . . . . .	16
Appendix A. Acknowledgements . . . . .	18
Appendix B. Changes compared to previous versions . . . . .	18
Appendix C. Examples . . . . .	19
C.1. Keepalive Configuration . . . . .	19
C.2. TCP-AO Configuration . . . . .	20
Appendix D. Complete Tree Diagram . . . . .	21
Authors' Addresses . . . . .	22

## 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:

- o TCP header attributes are modeled in other models, such as YANG Data Model for Network Access Control Lists (ACLs) [RFC8519] and Distributed Denial-of-Service Open Thread Signaling (DOTS) Data Channel Specification [RFC8783].
- o TCP-related configuration of a NAT (e.g., NAT44, NAT64, Destination NAT, ...) is defined in A YANG Module for Network Address Translation (NAT) and Network Prefix Translation (NPT) [RFC8512] and A YANG Data Model for Dual-Stack Lite (DS-Lite) [RFC8513].

## 2. Requirements Language

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

### 2.1. Note to RFC Editor

This document uses several placeholder values throughout the document. Please replace them as follows and remove this note before publication.

RFC XXXX, where XXXX is the number assigned to this document at the time of publication.

2020-11-16 with the actual date of the publication of this document.

### 3. Model Overview

#### 3.1. Modeling Scope

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

- o Global configuration: Many TCP implementations have configuration parameters that affect all TCP connections. Typical examples include enabling or disabling optional protocol features.
- o Interface configuration: It can be useful to use different TCP parameters on different interfaces, e.g., different device ports or IP interfaces. In that case, TCP parameters can be part of the interface configuration. Typical examples are the Maximum Segment Size (MSS) or configuration related to hardware offloading.
- o Connection parameters: Many implementations have means to influence the behavior of each TCP connection, e.g., on the programming interface used by applications. A typical example are socket options in the socket API, such as disabling the Nagle algorithm by TCP\_NODELAY. If an application uses such an interface, it is possible that the configuration of the application or application protocol includes TCP-related parameters. An example is the BGP YANG Model for Service Provider Networks [I-D.ietf-idr-bgp-model].
- o Policies: Setting of TCP parameters can also be part of system policies, templates, or profiles. An example would be the preferences defined in An Abstract Application Layer Interface to Transport Services [I-D.ietf-taps-interface].

As a result, there is no ground truth for setting certain TCP parameters, and traditionally different TCP implementations have used different modeling approaches. For instance, one implementation may define a given configuration parameter globally, while another one uses per-interface settings, and both approaches work well for the corresponding use cases. Also, different systems may use different default values. In addition, TCP can be implemented in different

ways and design choices by the protocol engine often affect configuration options.

Nonetheless, a number of TCP stack parameters require configuration by YANG models. This document therefore defines a minimal YANG model with fundamental parameters directly following from TCP standards.

An important use case is the TCP configuration on network elements such as routers, which often use YANG data models. The model therefore specifies TCP parameters that are important on such TCP stacks.

A typical example is the support of TCP-AO [RFC5925]. TCP-AO is increasingly supported on routers to secure routing protocols such as BGP. In that case, TCP-AO configuration is required on routers. 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.

- o Statistics: Counters for the number of active/passive opens, sent and received segments, errors, and possibly other detailed debugging information
- o TCP connection table: Access to status information for all TCP connections

This allows implementations of TCP MIB [RFC4022] to migrate to the YANG model defined in this memo.

### 3.2. Model Design

The YANG model defined in this document includes definitions from the YANG Groupings for TCP Clients and TCP Servers [I-D.ietf-netconf-tcp-client-server]. Similar to that model, this specification defines YANG groupings. This allows reuse of these groupings in different YANG data models. It is intended that these groupings will be used either standalone or for TCP-based protocols as part of a stack of protocol-specific configuration models. An example could be the BGP YANG Model for Service Provider Networks [I-D.ietf-idr-bgp-model].

### 3.3. Tree Diagram

This section provides a abridged tree diagram for the YANG module defined in this document. Annotations used in the diagram are defined in YANG Tree Diagrams [RFC8340].

```
module: ietf-tcp
  +--rw tcp!
    +--rw connections
    |   ...
    +--rw server {server}?
    |   ...
    +--rw client {client}?
    |   ...
    +--ro statistics {statistics}?
    |   ...
```

### 4. TCP YANG Model

<CODE BEGINS> file "ietf-tcp@2020-11-16.yang"

```
module ietf-tcp {
  yang-version "1.1";
  namespace "urn:ietf:params:xml:ns:yang:ietf-tcp";
  prefix "tcp";

  import ietf-yang-types {
    prefix "yang";
    reference
      "RFC 6991: Common YANG Data Types.";
  }
  import ietf-tcp-client {
    prefix "tcpc";
  }
  import ietf-tcp-server {
    prefix "tcps";
  }
  import ietf-tcp-common {
    prefix "tcpcmn";
  }
  import ietf-inet-types {
    prefix "inet";
  }

  organization
    "IETF TCPM Working Group";

  contact
```



"WG Web: <<http://tools.ietf.org/wg/tcpm>>  
WG List: <[tcpm@ietf.org](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 "2020-11-16" {  
  description  
    "Initial Version";  
  reference  
    "RFC XXXX, TCP Configuration.";  
}  
  
// Features  
feature server {  
  description  
    "TCP Server configuration supported.";  
}  
  
feature client {  
  description  
    "TCP Client configuration supported.";
```

```
}

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

// TCP-AO Groupings

grouping ao {
  leaf enable-ao {
    type boolean;
    default "false";
    description
      "Enable support of TCP-Authentication Option (TCP-AO).";
  }

  leaf send-id {
    type uint8 {
      range "0..255";
    }
    must "../enable-ao = 'true'";
    description
      "The SendID is inserted as the KeyID of the TCP-AO option
      of outgoing segments.";
    reference
      "RFC 5925: The TCP Authentication Option.";
  }

  leaf recv-id {
    type uint8 {
      range "0..255";
    }
    must "../enable-ao = 'true'";
    description
      "The RecvID is matched against the TCP-AO KeyID of incoming
      segments.";
    reference
      "RFC 5925: The TCP Authentication Option.";
  }

  leaf include-tcp-options {
    type boolean;
    must "../enable-ao = 'true'";
    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
  "Connection related parameters.";
}
description
  "A container of all TCP connections.";
```

```
container server {
  if-feature server;
  uses tcps:tcp-server-grouping;
  description
    "Definitions of TCP server configuration.";
}

container client {
  if-feature client;
  uses tcpc:tcp-client-grouping;
  description
    "Definitions of TCP client configuration.";
}

container statistics {
  if-feature statistics;
  config false;

  leaf active-opens {
    type yang:counter32;
    description
      "The number of times that TCP connections have made a direct
      transition to the SYN-SENT state from the CLOSED state.";
  }

  leaf passive-opens {
    type yang:counter32;
    description
      "The number of times TCP connections have made a direct
      transition to the SYN-RCVD state from the LISTEN state.";
  }

  leaf attempt-fails {
    type yang:counter32;
    description
      "The number of times that TCP connections have made a direct
      transition to the CLOSED state from either the SYN-SENT
      state or the SYN-RCVD state, plus the number of times that
      TCP connections have made a direct transition to the
      LISTEN state from the SYN-RCVD state.";
  }

  leaf establish-resets {
    type yang:counter32;
    description
      "The number of times that TCP connections have made a direct
      transition to the CLOSED state from either the ESTABLISHED
      state or the CLOSE-WAIT state.";
```

```
}

leaf currently-established {
  type yang:gauge32;
  description
    "The number of TCP connections for which the current state
    is either ESTABLISHED or CLOSE-WAIT.";
}

leaf in-segments {
  type yang:counter64;
  description
    "The total number of segments received, including those
    received in error. This count includes segments received
    on currently established connections.";
}

leaf out-segments {
  type yang:counter64;
  description
    "The total number of segments sent, including those on
    current connections but excluding those containing only
    retransmitted octets.";
}

leaf retransmitted-segments {
  type yang:counter32;
  description
    "The total number of segments retransmitted; that is, the
    number of TCP segments transmitted containing one or more
    previously transmitted octets.";
}

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

leaf out-resets {
  type yang:counter32;
  description
    "The number of TCP segments sent containing the RST flag.";
}

action reset {
  description
```

```
    "Reset statistics action command.";
  input {
    leaf reset-at {
      type yang:date-and-time;
      description
        "Time when the reset action needs to be
        executed.";
    }
  }
  output {
    leaf reset-finished-at {
      type yang:date-and-time;
      description
        "Time when the reset action command completed.";
    }
  }
}
description
  "Statistics across all connections.";
}
```

<CODE ENDS>

## 5. IANA Considerations

### 5.1. The IETF XML Registry

This document registers two URIs in the "ns" subregistry of the IETF XML Registry [RFC3688]. Following the format in IETF XML Registry [RFC3688], the following registrations are requested:

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

### 5.2. The YANG Module Names Registry

This document registers a YANG modules in the YANG Module Names registry YANG - A Data Modeling Language [RFC6020]. Following the format in YANG - A Data Modeling Language [RFC6020], the following registrations are requested:

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

## 6. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) described in Using the NETCONF protocol over SSH [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

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:

- o Server configuration. Unrestricted access to all the nodes under server configuration, e.g. local-address or keepalive idle-timer, can cause connections to the server to fail or to timeout prematurely.
- o Client configuration. Similar to server configuration, unrestricted access to the nodes under client configuration can cause connections from the client to fail, or connections to the server to be redirected, and in case of keepalive, cause connections to timeout prematurely etc.

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:

- o Unrestricted access to connection information of the client or server can be used by a malicious user to launch an attack, e.g. MITM.
- o 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:

- o 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", draft-ietf-netconf-tcp-client-server-08 (work in progress), August 2020.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, DOI 10.17487/RFC2385, August 1998, <<https://www.rfc-editor.org/info/rfc2385>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <<https://www.rfc-editor.org/info/rfc5925>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.

- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [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", draft-ietf-idr-bgp-model-09 (work in progress), June 2020.

[I-D.ietf-taps-interface]

Trammell, B., Welzl, M., Enghardt, T., Fairhurst, G., Kuehlewind, M., Perkins, C., Tiesel, P., Wood, C., and T. Pauly, "An Abstract Application Layer Interface to Transport Services", draft-ietf-taps-interface-10 (work in progress), November 2020.

[I-D.touch-tcpm-ao-test-vectors]

Touch, J. and J. Kuusisaari, "TCP-AO Test Vectors", draft-touch-tcpm-ao-test-vectors-01 (work in progress), August 2020.

[RFC4022] Raghunarayan, R., Ed., "Management Information Base for the Transmission Control Protocol (TCP)", RFC 4022, DOI 10.17487/RFC4022, March 2005, <<https://www.rfc-editor.org/info/rfc4022>>.

[RFC4898] Mathis, M., Heffner, J., and R. Raghunarayan, "TCP Extended Statistics MIB", RFC 4898, DOI 10.17487/RFC4898, May 2007, <<https://www.rfc-editor.org/info/rfc4898>>.

[RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIPv2) 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

- o Removed congestion control
- o Removed global stack parameters

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

- o Updated TCP-AO grouping
- o Added congestion control

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

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

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

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

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

- o Editorial improvements

## Appendix C. Examples

## C.1. Keepalive Configuration

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

[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).
-->
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <tcp
    xmlns="urn:ietf:params:xml:ns:yang:ietf-tcp">
    <connections>
      <connection>
        <local-address>192.168.1.1</local-address>
        <remote-address>192.168.1.2</remote-address>
        <local-port>1025</local-port>
        <remote-port>80</remote-port>
        <common>
          <keepalives>
            <idle-time>5</idle-time>
            <max-probes>5</max-probes>
            <probe-interval>10</probe-interval>
          </keepalives>
        </common>
      </connection>
    </connections>
  <!--
    It is not clear why a server and client configuration is
    needed here even as they under a feature statement and
    therefore are required only if the feature is declared.
    Adding it so that yanglint allows this validation to run.
  -->
  <server>
    <local-address>192.168.1.1</local-address>
  </server>
  <client>
    <remote-address>192.168.1.2</remote-address>
  </client>
</tcp>
</config>
```

## 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.
-->
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <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."</descriptio\
n>
      <key>
        <key-id>61</key-id>
        <crypto-algorithm>hmac-sha-1-12</crypto-algorithm>
        <key-string>
          <hexadecimal-string>01:23:a5:93:b9:db:70:62:9b:be:2c:a6:77:cd:fd:e\
a:6f:e0:ac:ad</hexadecimal-string>
        </key-string>
      </key>
    </key-chain>
  </key-chains>
  <tcp
    xmlns="urn:ietf:params:xml:ns:yang:ietf-tcp">
    <!--
      The example in draft-touch-tcpm-ao-test-vectors uses
      IP addresses that are not valid for examples. Changing
      them here to valid addresses.
    -->
    <server>
      <local-address>192.168.1.1</local-address>
    </server>
    <client>
      <remote-address>192.168.1.2</remote-address>
    </client>
  </tcp>
</config>
```

## 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 recv-id?     uint8
              +--rw include-tcp-options? boolean
              +--rw accept-key-mismatch? boolean
            +--:(md5)
              +--rw enable-md5?   boolean
      +--rw server {server}?
        +--rw local-address      inet:ip-address
        +--rw local-port?        inet:port-number
        +--rw keepalives!
          +--rw idle-time        uint16
          +--rw max-probes       uint16
          +--rw probe-interval   uint16
      +--rw client {client}?
        +--rw remote-address     inet:host
        +--rw remote-port?       inet:port-number
        +--rw local-address?     inet:ip-address
        +--rw local-port?        inet:port-number
        +--rw keepalives!
          +--rw idle-time        uint16
          +--rw max-probes       uint16
          +--rw probe-interval   uint16
      +--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  
Esslingen 73732  
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



TCP Maintenance and Minor Extensions  
Internet-Draft  
Intended status: Informational  
Expires: 10 August 2021

J. Kang, Ed.  
Q. Liang  
Huawei  
6 February 2021

Subtype Capability Exchange During MPTCP Handshake  
draft-kang-tcpm-subtype-capability-exchange-00

Abstract

Multipath TCP provides the ability to simultaneously use multiple paths between peers. MPTCP protocol defines seven subtypes in MPTCP v0 [RFC6824] and ten subtypes in MPTCP v1 [RFC8684] to differentiate message types and implement some additional functions during a session.

This draft proposes an enhancement to support Subtype Capability Exchange during MPTCP connection establishment in order to improve elastic scalability of MPTCP protocol. It includes: 1) requirements for which this kind of capability exchange during handshake is important for a MPTCP session; 2) a typical flow for Subtype Capability Exchange between peers; 3) a feasible solution on protocol design is suggested.

Status of This Memo

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

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

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

This Internet-Draft will expire on 10 August 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 . . . . .	2
1.1. Requirements Language . . . . .	2
1.2. Background . . . . .	2
2. One Typical Flow . . . . .	3
3. Protocol Implementation . . . . .	5
3.1. Carrying Subtype Capabilities in MP_CAPABLE Option . . . . .	5
4. Security Considerations . . . . .	6
5. IANA Considerations . . . . .	6
6. References . . . . .	6
6.1. Normative References . . . . .	6
6.2. Informative References . . . . .	7
Authors' Addresses . . . . .	7

## 1. Introduction

### 1.1. Requirements Language

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

### 1.2. Background

Table 1 lists all subtypes that have been specified in current MPTCP versions. Besides version negotiation, MPTCP peers can not interact with each other on the granularity of subtype capability. This feature may cause inflexible protocol extension. For example, if a new message type A is added in future extension, a higher version should be released to import it and a new subtype may need to be allocated. Another case is that if a sender does not know the subtypes supported by a receiver in a MPTCP session, as a result, invalid data packets may be sent from the sender during data transmission and the receiver will discard it which causes system overhead on receiver side.

Value	Symbol	Name	MPTCPv0	MPTCPv1
0x0	MP_CAPABLE	Multipath Capable	Supported	Supported
0x1	MP_JOIN	Join Connection	Supported	Supported
0x2	DSS	Data Sequence Signal (Data ACK and Data Sequence Mapping)	Supported	Supported
0x3	ADD_ADDR	Add Address	Supported	Supported
0x4	REMOVE_ADDR	Remove Address	Supported	Supported
0x5	MP_PRIO	Change Subflow Priority	Supported	Supported
0x6	MP_FAIL	Fallback	Supported	Supported
0x7	MP_FASTCLOSE	Fast Close	Supported	Supported
0x8	MP_TCP_RST	Subflow Reset	/	Supported
0xf	MP_EXPERIMENTAL	Reserved for Private Use	/	Supported

Table 1: Overview MPTCP Subtypes

This document suggests a new function of Subtype Capability Exchange during MPTCP handshake in the scenario that MPTCP peers in a session support same MPTCP protocol version but with different subtype sets.

## 2. One Typical Flow

Figure 1 illustrates a typical flow for this Subtype Capability Exchange during MPTCP connection setup. The field of Subtype Capability is used to indicate whether these subtypes are supported by the sender, for example, Host A Subtype Capabilities indicates the status of the subtypes on Host A and Host B Subtype Capabilities indicates that on Host B. Through the transmission of this

information between both parties, a sender can determine whether a message can be properly processed by its receiver and only send the message that can be supported by the receiver during data transmission.

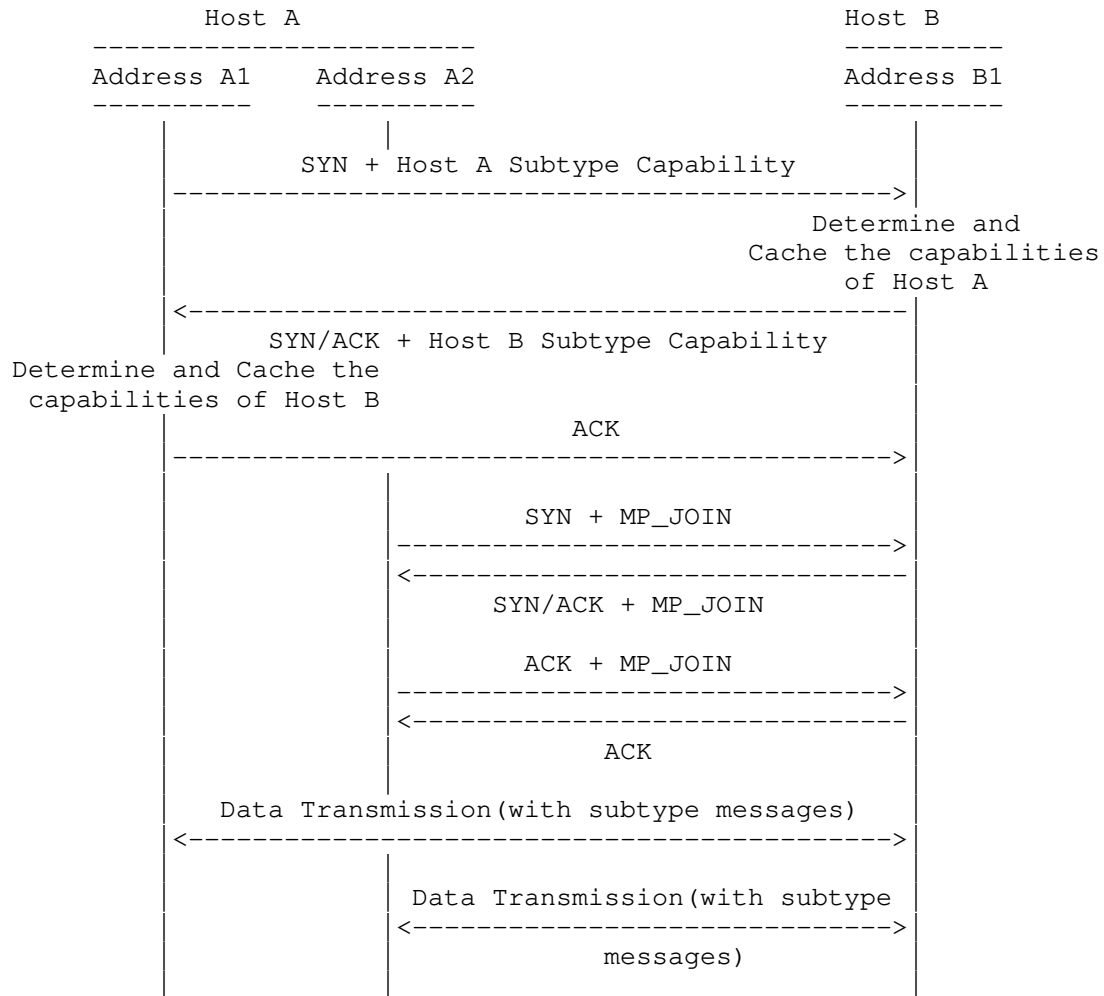


Figure 1: MPTCP Subtype Capability Exchange

In practice, another possible implementation is as follows: after receiving the subtype capability information sent by Host A, Host B determines the common subtype sets supported by both parties, and returns this common subtype sets in the reponse. Host A caches this common subtype sets locally. In data transmission phase, Host A sends the specified subtype messages to Host B that are included in the common subtype sets. As an alternative solution, its protocol design on MPTCP will be considered and updated in later versions.

### 3. Protocol Implementation

This document describes one solution on the modifications to MPTCP protocol to support this mechanism. In this solution, MP\_CAPABLE option is used and extended to add bits to carry subtype capabilities information. There should be other possible solutions that can be defined in subsequent discussions.

#### 3.1. Carrying Subtype Capabilities in MP\_CAPABLE Option

In Figure 2, a 32-bit "OptionSupported" is added to MP\_CAPABLE option to indicate whether the subtypes are supported by the sender.

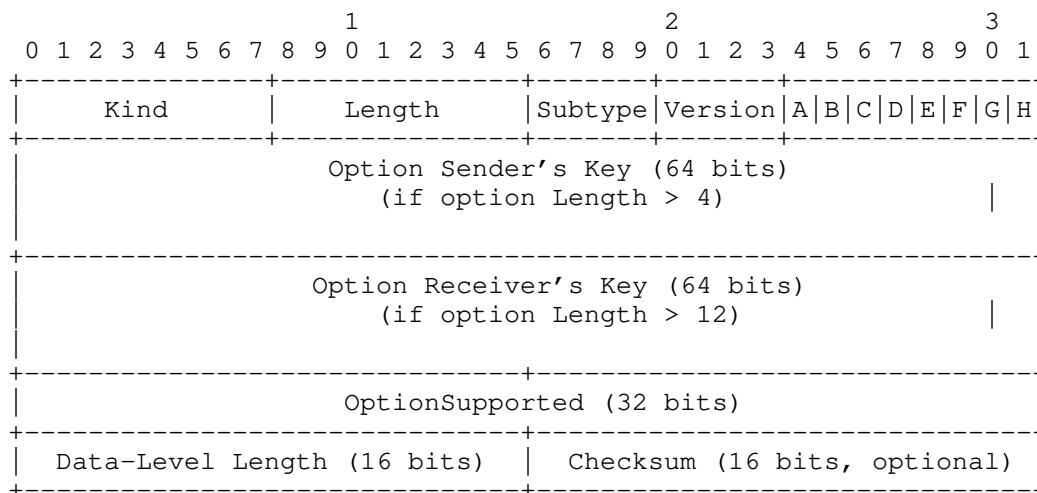


Figure 2: OptionSupported Format

For MPTCP v1, ten subtypes has been defined and applied in practice. So the first 10-bits in OptionSupported field is used for indicating whether these subtypes is supported by sender. The order is listed below:

0: MP\_CAPABLE

- 1: MP\_JOIN
- 2: DSS
- 3: ADD\_ADDR
- 4: REMOVE\_ADDR
- 5: MP\_PRIO
- 6: MP\_FAIL
- 7: MP\_FASTCLOSE
- 9: MP\_TCPRST
- 10: MP\_EXPERIMENTAL
- 11~31: Reserved for Future Use

Two values, that is 0 and 1, can be set to these bits in OptionSupported field. The value of 0 indicates that the sender does not support this subtype. The value of 1 indicates that the sender supports this subtype.

#### 4. Security Considerations

To be added.

#### 5. IANA Considerations

To be added.

#### 6. References

##### 6.1. 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>>.

- [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>>.
- [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>>.

## 6.2. Informative References

- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, DOI 10.17487/RFC2629, June 1999, <<https://www.rfc-editor.org/info/rfc2629>>.

### Authors' Addresses

Jiao Kang (editor)  
Huawei  
D2-03, Huawei Industrial Base  
Shenzhen  
China

Email: kangjiao@huawei.com

Qiandeng Liang  
Huawei  
No. 207, Jiufeng 3rd Road, East Lake High-tech Development Zone  
Wuhan  
China

Email: liangqiandeng@huawei.com

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: August 6, 2021

Y. Nishida  
Amazon Web Services  
February 2, 2021

Aggregated Option for SYN Option Space Extension  
draft-nishida-tcpm-agg-syn-ext-00

Abstract

TCP option space is scarce resource as its max length is limited to 40 bytes. This limitation becomes more significant in SYN segments as all options used in a connection should be exchanged during SYN negotiations. This document proposes a new SYN option negotiation scheme that provide a feature to compress TCP options in SYN segments and provide more option space. The proposed scheme does not update the format of TCP header nor transmit any additional SYN or SYN-like segments so that it has lower risks for middlebox interventions. In addition, by combining another proposal for option space extension, it is possible to provide further option space.

Status of This Memo

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

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

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

This Internet-Draft will expire on August 6, 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 . . . . .	2
2. Conventions and Terminology . . . . .	4
3. Basic Design . . . . .	4
3.1. Aggregated Option . . . . .	4
3.2. Additional Information Exchange Using 3rd ACK Segments .	6
3.3. Examples of Option Exchanges with Aggregated Options . .	7
4. Option Bits Registration . . . . .	9
5. Discussions . . . . .	9
5.1. Incremental Deployments for Aggregated Option . . . . .	10
5.2. Middlebox Considerations . . . . .	10
5.3. Which Options are Aggregatable . . . . .	10
5.4. Further Extensions . . . . .	11
6. Security Considerations . . . . .	11
7. IANA Considerations . . . . .	11
7.1. Aggregated Option . . . . .	11
7.2. Option Bits Registry for Aggregated Option . . . . .	12
8. References . . . . .	12
8.1. Normative References . . . . .	12
8.2. Informative References . . . . .	12
Author's Address . . . . .	14

## 1. Introduction

TCP option space is scarce resource as its max length is limited to 40 bytes. This limitation is a nominal issue especially for SYN segment. This is because although a TCP endpoint can send only one SYN segment to the peer, SYN segments need to contain all options expected to be used for the connection. As a result, the current SYN option space tends to be congested. Many TCP connections use MSS [RFC0793], Timestamp and Window Scale [RFC7323], SACK Permitted options [RFC2018] which already consume 19 bytes (10 + 4 + 3 + 2). In addition to them, if a connection wants to use Multipath TCP [RFC8684], it requires additional 4-12 bytes for MP\_CAPABLE or 12-16 bytes for MP\_JOIN option. Similarly, TCP Fast Open [RFC7413] and TCP AO [RFC5925] require additional 6-18 bytes and 16 bytes respectively. Moreover, Experimental Option Format defined in [RFC6994] consumes more option space as it requires additional 16 bits or 32 bits EXID field than the standard option format. Hence, if an endpoint is willing to activate some of extra features in addition to common options, 40 bytes space may not be sufficient. If SYN segment cannot

accommodate all required options, it means endpoints need to give up using some features. This issue affects the extensibility of TCP.

There have been various proposals in order to extend option space in SYN Segments [I-D.eddy-tcp-loo] [I-D.yourtchenko-tcp-loic] [I-D.touch-tcpm-tcp-syn-ext-opt] [I-D.briscoe-tcpm-inner-space] [I-D.allman-tcpv2-hack]. These proposals have adopted one or both of the following two types of approach.

- o Extending TCP header in SYN segment: This approach tries to accommodate more options in a SYN segment by using payload. (E.g override Data Offset field in TCP header)
- o Using Multiple SYN or SYN-like segment: This approach tries to use multiple SYN segment or additional segment that can be treated as a SYN segment. (E.g sending another SYN with wrong checksum or from different source port)

However, these approach tend to require a certain complexity and have potential risks for middlebox interventions. In this document, in order to reduce the risk for middlebox intervention we propose to extend option space in SYN segments through the following approach which utilizes "3rd segments". In this document, we define "3rd segment" as the segments sent from initiator that contains acknowledge for the SYN ACK segment from responder.

- o Aggregated Option Format: we propose a new option format that can aggregate multiple TCP options into a single option format so that it can create more space than conventional option formats.
- o Using 3rd segments for supplemental negotiation: we propose to use 3rd segment and its acknowledge for additional feature information exchanges.

One important characteristic of this approach is that it does not require any changes in SYN header format nor using multiple SYN or SYN-like segments. In stead, it utilizes 3rd segments which should looks legitimate segments from middleboxes' points of view. MPTCP [RFC8684] specifications already requires the use of 3rd segment for further information exchanges. Hence, in a sense, the proposal in this document may look using the scheme in MPTCP for more generic purposes. Given that the maturity and the deployment status of MPTCP implementations, we believe that this approach has lower risks for middlebox interventions without introducing lots of complexity in TCP architecture.

## 2. Conventions and 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. Basic Design

The proposed scheme in this document has the following two design criteria.

1. All feature negotiations are archived during a single SYN exchange. No extra SYN or SYN-like segments are utilized.
2. No change is required for TCP header format. Use current option space in TCP header and not use payload for option space.

These requirements ensure that the scheme can have simple architecture and can traverse middleboxes without problems. In order to fulfill this requirements, For this purpose, we propose a new option format and a scheme for additional information exchange as follows.

### 3.1. Aggregated Option

This aggregated option can be used only to indicate that an endpoint want to enable the specified features in the option. This option uses one bit to express one TCP option. This is because this option is used only to indicate that an endpoint wants to use specified features. The receiver of the option can only specify whether it agrees to use the requested features or not. The format of aggregated option format is shown in Figure 1. The option can contains 0-3 Aggregated Blocks which have 1 byte length. The length of the option format is varied by the number of Aggregated blocks in the option.

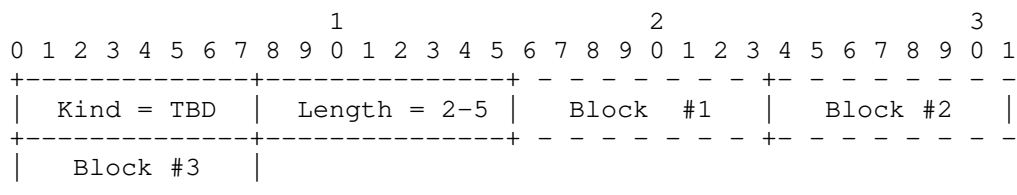


Figure 1: Aggregated Option format

Figure 2 shows the format of Aggregated block. Aggregated block has 1 byte length which consists of 2 bits "Group ID" (GID) field and 6 bits "Option Bits" field. The options supported by Aggregated Option is split into 4 groups, such as an option defined in a standard RFC belongs to group 1 or group 2, an option in an experimental RFC belongs to group 3, etc. (Note: how to allocate option bits will be discussed later) Group ID field is used to identify the group identifier of the option bits in the Aggregated Option and each single bit in Option Bits field represent a option in the group.

If all options that an endpoint wants to aggregate belong to one group, the aggregated option will need to contain just one Aggregated Block. However, if the option will need to contain multiple blocks when an endpoint wants to use options in multiple groups.

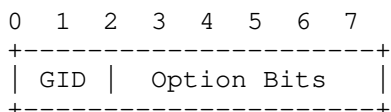


Figure 2: Aggregated Block format

GID field in Aggregated Block indicate the group ID that option bits in the block belongs to. Aggregated Blocks are appeared in SYN and SYN ACK segments, however, different mappings between GID value in the option and Group ID are used for these two segments. This is because some implementations may copy back unknown options in their response segments. These mappings is used not to be confused by such cases. Table Table 1 shows the mapping between GID value in the option and Group ID. For example, GID value 1 in SYN represents group 2, while GID value 1 in ACK segment for SYN ACK represents group 1.

GID value in SYN	GID value in SYN ACK	Group ID Description
0	1	group 1
1	2	group 2
2	3	group 3
3	0	Extensions

Table 1: Mapping between GID value and Group ID

The allocation of the bit in Option Bits field in each group will be managed by the registry provided by IANA. Since an aggregated block has 6 bits field to indicate options, one group can have 6 options at most. As shown in Table 1, we currently propose to allocate 3 GID for options and 1 GID is reserved for future extensions. Consequently, the maximum number of options that can be aggregated with Aggregated Option will be limited to 18. We believe this is sufficient number for the time being based on the current usage of option code points. In addition, Section 5.4 describes some possibilities to overcome the limitation in case we need to support more options to be supported.

Aggregated Option that contains Aggregated Blocks MUST be only used in SYN segments. When an endpoint received SYN segments with Aggregated Option, it checks Aggregated Blocks in the option. If the option does not contain Aggregated Blocks, the segment MUST be discarded. If it contains Aggregated Blocks, the options specified in the blocks MUST be processed as well as options in original formats. When a responder sends back SYN ACK to the initiator, it uses original formats of the options or Aggregated options depends on remaining options space or other criteria.

### 3.2. Additional Information Exchange Using 3rd ACK Segments

Aggregated Option is used to negotiate the use of features that an endpoint would like to activate. However, it does not provide a way to negotiate additional information. This suffices for options that do not require additional parameters such as SACK-Permit option. However, if options need additional parameter exchanges, these parameters will need to be sent through 3rd segment. When options are sent in the 3rd segments, original formats of the options are used. In this case, in order to solicit acknowledgement for this segments, an initiator MUST include 2 bytes length Aggregated Option to 3rd segment with the format shown in Figure 3.

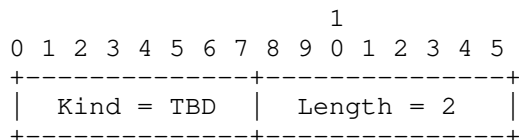


Figure 3: 2-Byte Aggregated Option format

In addition, it MUST start retransmission timer even if it is a pure ACK and MUST retransmit the same segment when the timer is expired. Note that the 3rd segment does not have to be a pure ACK segment. If an endpoint has data to be sent, it can send it with the segment.

When a responder receives 2 bytes length Aggregated Option in the 3rd segment, it MUST send back Aggregated Option to acknowledge the arrival of the option. If responder does not need to receive acknowledgement from initiator, it MUST send back a segment with Aggregated Option with the format shown in Figure 4 to indicate the end of option exchanges. If responder need to send additional options that requires acknowledgement, it MUST send 2-bytes Aggregated Option shown in Figure 3 and MUST start retransmission timer for retransmission. An endpoint that receives Fin Aggregated Option MUST not use Aggregated Options in subsequent segments. Similarly, an endpoint that sends Fin Aggregated Option MUST not use Aggregated Options once the segment is acknowledged.

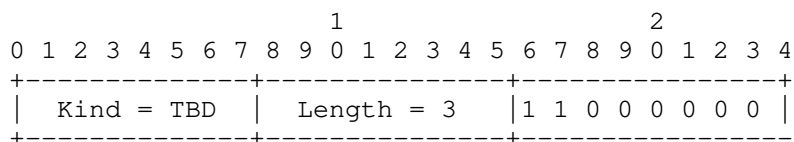


Figure 4: Fin Aggregated Option format

When an endpoint sends a segment with 2 byte Aggregated Option, it MUST retransmit the segment until it receives an ACK for the segment with Aggregated Option (2 byte or Fin Aggregated Option). When it receives an ACK for the segment without Aggregated Option, it MUST ignore and discard the ACK.

### 3.3. Examples of Option Exchanges with Aggregated Options

We show 3 patterns of option negotiations with Aggregated Option in the following examples: Figure Figure 5 , Figure Figure 6 and Figure Figure 7. In these examples, we use hypothetical option A and B. Option A is an option that does not require additional information such as SACK Permit option. While Option B are an option that needs to exchanges additional parameters. "Non-Agg options" in the examples represent the options that are not aggregated, which are sent with their original formats.

As shown in Figure 5, as option A does not require additional information, exchanging SYN and SYN ACK segments with Aggregated Options are enough for feature negotiation. In this case, initiator sends back 3rd segment with Fin Aggregated Option only to indicate the end of option negotiation.

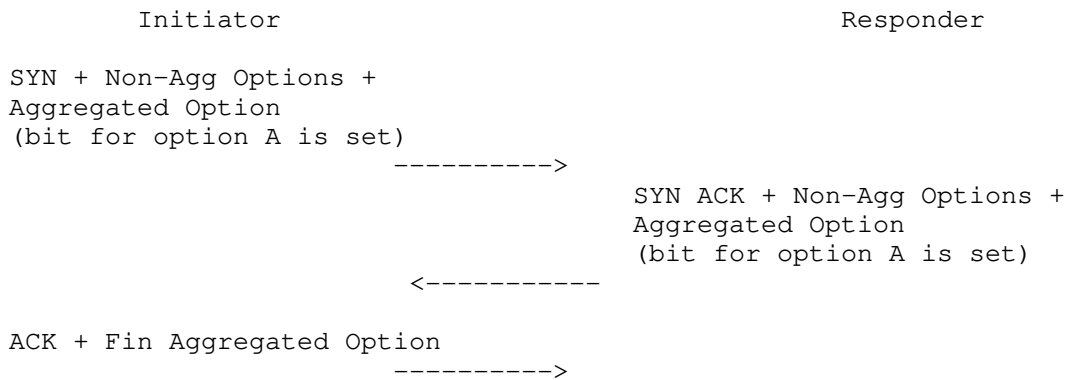


Figure 5: Example of Aggregated Option #1

Figure 6 is an example that uses option B which require additional information exchange. In this case, After exchanging Aggregated Options during SYN exchanges, initiator sends 3rd ACK segment with 2-Byte Aggregated Options in addition to the original form of option B. Similarly, responder sends back the ACK for the segment with 2-Bytes Aggregated Options in addition to the original form of Option B. After that, initiator sends an ACK with Fin Aggregated Option only to indicate the end of option negotiation.

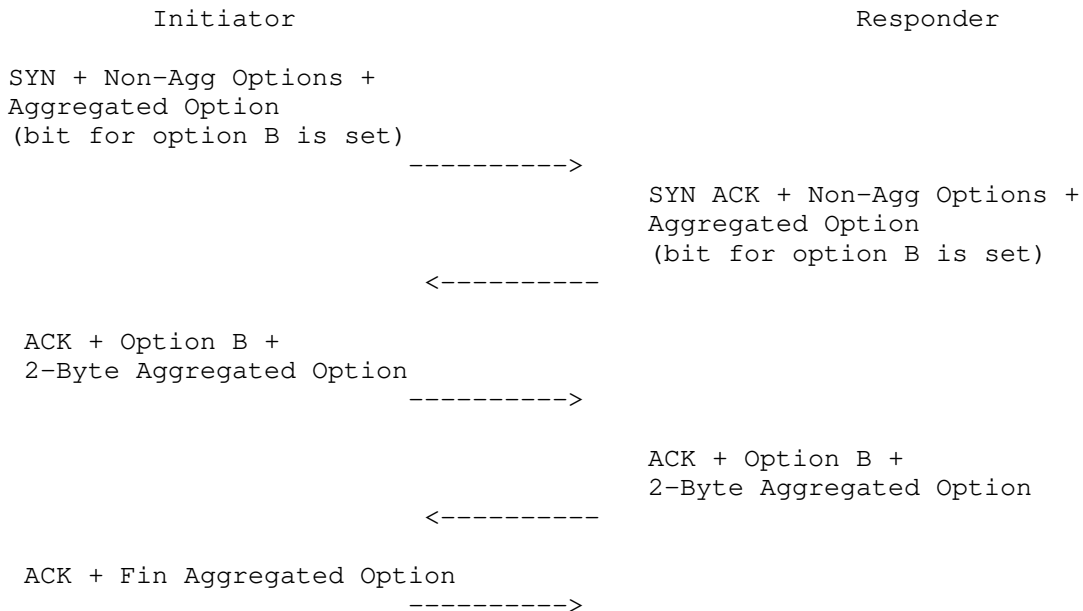


Figure 6: Example of Aggregated Option #2

Figure 7 also uses option B as Figure 6, however, initiator only sends two segments for option negotiations. In this case, when responder sends SYN-ACK, it sends back the segment with original form of Option B. In this case, when responder sends back the ACK for 3rd segment, it sends the segment with Fin Aggregated Option to indicate the end of option negotiation. In this case, while sender utilizes extra option space with Aggregated Option in 3rd segment, responder uses option space only in SYN ACK segment. This use case can be useful when only option space in SYN segment needs to be extended.

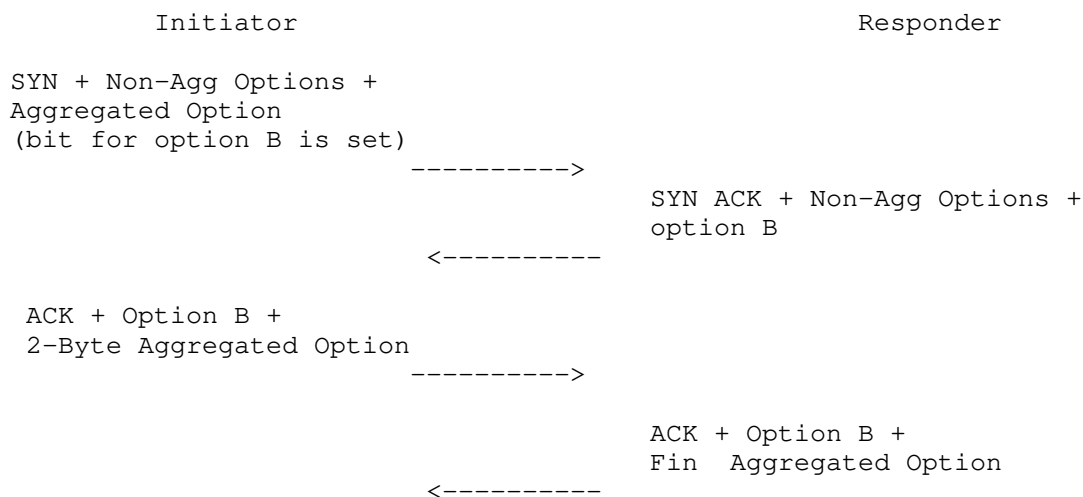


Figure 7: Example of Aggregated Option #3

#### 4. Option Bits Registration

As described in Section 3, Aggregated Option has 18 Option bits space and each bit represents a single option ID. The allocation of the Option Bits in Aggregated Option is maintained by IANA [IANA]. If a new option can be aggregatable, one can request Option Bit in addition to the current procedure, requesting TCP Option Kind Number in [TCPPParameters] . If a option already has assigned TCP Option Kind Number, one can only request Option bit which will represent the option kind.

#### 5. Discussions



### 5.1. Incremental Deployments for Aggregated Option

An endpoint MAY send the original form of an option and Aggregated Option for the option in SYN segments if option space is available. This may look redundant, but it can be useful when an endpoint sends a SYN segment to a new destination. In this case, if the peers support the option and Aggregated Option, they can send back Aggregated Option in addition to the response for normal option format. The endpoint cache this information and when it opens another connection to the the same destination with a short interval, it can send only Aggregated Option this time. If an endpoint did not receive SYN ACK when it sent Aggregated Option to a new destination, the retransmitted SYN segment SHOULD NOT include Aggregated Option.

### 5.2. Middlebox Considerations

As Aggregated Option will be a new option, there is a potential risk that the option will be removed from the segment or the segment that carries the option will be discarded by the middleboxes. While we believe the risk for this risk is low from the past studies [HONDA11], if endpoints try to aggregate well-known TCP options such as SACK-Permit, MSS, TimeStamp to new destinations, it is recommended to follow incremental deployment approach described in the previous section. On the other hand, the risk for the removal of Aggregated Option and new options can be considered mostly in the same level. In this case, endpoints MAY send only the aggregated option to new destinations.

If a middlebox strips Aggregated options in SYN segments but keeps other options unchanged, the proposed scheme can behave safely. When Aggregated Option in SYN segment from initiator is removed on outgoing path, both endpoint will not activate any features in the Aggregated Option. When Aggregated Option is removed on return path, there will be a situation where the responder enables the features in Aggregated Option while initiator disables the features. However, the responder still can aware that the Aggregated Option is removed on the return path when it receives the 3rd segments without Aggregate Option. In this case, the responder can decide whether it can continue the connection or reset it based on the features in the Aggregated Option.

### 5.3. Which Options are Aggregatable

While we think most of options can be aggregated by using Aggregated Option, there will be some limitations. One example would be TCP AO [RFC5925] as it has to carry security information in SYN segments. If an option must carry additional information for the feature like TCP AO, we cannot aggregate the option. TCP Fast Open [RFC7413]

would be another example as it needs to carry cached cookie information in SYN segments. Sending cookie info in the 3rd segments is not desirable from the objective of TCP Fast Open. However, when initiator sends Fast Open Cookie Request, the request can be aggregated as it does not carry additional information. Hence, we still believe that it will be useful to aggregate TCP Fast Open option in some cases.

#### 5.4. Further Extensions

The proposed approach can aggregate 18 options with Aggregated Options. We believe this is enough number for the time being based on the current usages of TCP options. It is also possible to accommodate more options by using Group id 4 which is currently reserved for future extensions.

Aggregated Option can extend available space for TCP options in SYN segments by utilizing 3rd segments and its acknowledgement. However, there might be a situation where this is still not enough to accommodate all TCP options that an endpoint would like to activate. In this case, one possible approach is to utilize TCP Extended Data Offset Option (EDO) [I-D.ietf-tcpm-tcp-edo]. As EDO supported Option does not carry any information, it can be aggregated in Aggregated Option. Once both endpoints agree to use the feature by SYN exchange, an endpoint can start using EDO to extend option space in the 3rd segments and its acknowledgement. This approach will create more space for options in SYN segments although further discussions will be required for more detailed design.

#### 6. Security Considerations

We believe Aggregated Option maintains the same level of security as other TCP options does, but further discussions will be required.

#### 7. IANA Considerations

This document requests new TCP option codepoint. In addition, this document requires new registry for the option. They are described in the following subsections.

##### 7.1. Aggregated Option

This document requests to add new option: Aggregated Option to the TCP option space registry which points to this document as follows:

Kind	Length	Meaning	Reference
TBD	N	Aggregated Option	This Document

Table 2: Aggregated Option Format

## 7.2. Option Bits Registry for Aggregated Option

This document also requests to create a "Aggregated Option Identifiers" registry in IANA registries. The registry maintains 24 records which are mapped to the TCP Option Kind Number Records in [TCPParameters]. The 24 records are divided into 4 groups so that each group contains 6 records.

## 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>>.

### 8.2. Informative References

[HONDA11] Honda, M., "Is it still possible to extend TCP?", IMC 2011, November 2011.

[I-D.allman-tcpv2-hack]  
Allman, M., "TCPv2: Don't Fence Me In", draft-allman-tcpv2-hack-00 (work in progress), May 2006.

[I-D.briscoe-tcpm-inner-space]  
Briscoe, B., "Inner Space for TCP Options", draft-briscoe-tcpm-inner-space-01 (work in progress), October 2014.

[I-D.eddy-tcp-loo]  
Eddy, W. and A. Langley, "Extending the Space Available for TCP Options", draft-eddy-tcp-loo-04 (work in progress), July 2008.

[I-D.ietf-tcpm-tcp-edo]  
Touch, J. and W. Eddy, "TCP Extended Data Offset Option", draft-ietf-tcpm-tcp-edo-10 (work in progress), July 2018.

- [I-D.touch-tcpm-tcp-syn-ext-opt]  
Touch, J. and T. Faber, "TCP SYN Extended Option Space Using an Out-of-Band Segment", draft-touch-tcpm-tcp-syn-ext-opt-09 (work in progress), July 2018.
- [I-D.yourtchenko-tcp-loic]  
Yourtchenko, A., "Introducing TCP Long Options by Invalid Checksum", draft-yourtchenko-tcp-loic-00 (work in progress), April 2011.
- [IANA] "IANA Home Page", <<https://www.iana.org/>>.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [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>>.
- [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>>.
- [RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, DOI 10.17487/RFC6994, August 2013, <<https://www.rfc-editor.org/info/rfc6994>>.
- [RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, DOI 10.17487/RFC7323, September 2014, <<https://www.rfc-editor.org/info/rfc7323>>.
- [RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <<https://www.rfc-editor.org/info/rfc7413>>.
- [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>>.
- [TCPPParameters]  
"Transmission Control Protocol (TCP) Parameters", <<https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml#tcp-parameters-1>>.

Author's Address

Yoshifumi Nishida  
Amazon Web Services  
410 Terry Avenue North  
Seattle, WA 98109  
USA

Email: nishida@wide.ad.jp

TCPM  
Internet Draft  
Intended status: Informational  
Expires: June 2021

J. Touch  
Independent consultant  
J. Kuusisaari  
Infinera  
December 23, 2020

## TCP-AO Test Vectors

draft-touch-tcpm-ao-test-vectors-02.txt

### Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, except to format it for publication as an RFC or to translate it into languages other than English.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>

This Internet-Draft will expire on June 23, 2021.

### Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://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.

## Abstract

This document provides test vectors to validate implementations of the two mandatory authentication algorithms specified for the TCP Authentication Option over both IPv4 and IPv6. This includes validation of the key derivation function (KDF) based on a set of test connection parameters as well as validation of the message authentication code (MAC). Vectors are provided for both currently required pairs of KDF and MAC algorithms: one based on SHA-1 and the other on AES-128. The vectors also validate both whole TCP segments as well as segments whose options are excluded for NAT traversal.

## Table of Contents

1. Introduction.....	3
2. Conventions used in this document.....	4
3. Input Test Vectors.....	4
3.1. TCP Connection Parameters.....	4
3.1.1. TCP-AO parameters.....	4
3.1.2. Active (client) side parameters.....	4
3.1.3. Passive (server) side parameters.....	5
3.1.4. Other IP fields and options.....	5
3.1.5. Other TCP fields and options.....	5
4. IPv4 SHA-1 Output Test Vectors.....	5
4.1. SHA-1 MAC (default - covers TCP options).....	6
4.1.1. Send (client) SYN (covers options).....	6
4.1.2. Receive (server) SYN-ACK (covers options).....	6
4.1.3. Send (client) non-SYN (covers options).....	7
4.1.4. Receive (server) non-SYN (covers options).....	7
4.2. SHA-1 MAC (omits TCP options).....	8
4.2.1. Send (client) SYN (omits options).....	8
4.2.2. Receive (server) SYN-ACK (omits options).....	8
4.2.3. Send (client) non-SYN (omits options).....	9
4.2.4. Receive (server) non-SYN (omits options).....	9
5. IPv4 AES-128 Output Test Vectors.....	10
5.1. AES MAC (default - covers TCP options).....	10
5.1.1. Send (client) SYN (covers options).....	10
5.1.2. Receive (server) SYN-ACK (covers options).....	11
5.1.3. Send (client) non-SYN (covers options).....	11
5.1.4. Receive (server) non-SYN (covers options).....	12

5.2. AES MAC (omits TCP options).....	12
5.2.1. Send (client) SYN (omits options).....	12
5.2.2. Receive (server) SYN-ACK (omits options).....	13
5.2.3. Send (client) non-SYN (omits options).....	13
5.2.4. Receive (server) non-SYN (omits options).....	14
6. IPv6 SHA-1 Output Test Vectors.....	14
6.1. SHA-1 MAC (default - covers TCP options).....	14
6.1.1. Send (client) SYN (covers options).....	14
6.1.2. Receive (server) SYN-ACK (covers options).....	15
6.1.3. Send (client) non-SYN (covers options).....	15
6.1.4. Receive (server) non-SYN (covers options).....	16
6.2. SHA-1 MAC (omits TCP options).....	17
6.2.1. Send (client) SYN (omits options).....	17
6.2.2. Receive (server) SYN-ACK (omits options).....	17
6.2.3. Send (client) non-SYN (omits options).....	18
6.2.4. Receive (server) non-SYN (omits options).....	18
7. IPv6 AES-128 Output Test Vectors.....	19
7.1. AES MAC (default - covers TCP options).....	19
7.1.1. Send (client) SYN (covers options).....	19
7.1.2. Receive (server) SYN-ACK (covers options).....	20
7.1.3. Send (client) non-SYN (covers options).....	20
7.1.4. Receive (server) non-SYN (covers options).....	21
7.2. AES MAC (omits TCP options).....	21
7.2.1. Send (client) SYN (omits options).....	21
7.2.2. Receive (server) SYN-ACK (omits options).....	22
7.2.3. Send (client) non-SYN (omits options).....	22
7.2.4. Receive (server) non-SYN (omits options).....	23
8. Observed Implementation Errors.....	23
8.1. Algorithm issues.....	23
8.2. Algorithm parameters.....	24
8.3. String handling issues.....	24
8.4. Header coverage issues.....	24
9. Security Considerations.....	24
10. IANA Considerations.....	25
11. References.....	25
11.1. Normative References.....	25
11.2. Informative References.....	25
12. Acknowledgments.....	26

## 1. Introduction

This document provides test vectors to validate the correct implementation of the TCP Authentication Option (TCP-AO) [RFC5925]. It includes the specification of all endpoint parameters to generate the variety of TCP segments covered by different keys and MAC coverage, i.e., both the default case and the variant where TCP



options are ignored. It also includes both default key derivation functions (KDFs) and MAC generation algorithms [RFC5926].

The experimental extension to support NAT traversal is not included in the provided test vectors [RFC6978].

This document provides test vectors from an implementation.

## 2. Conventions used in this document

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. Input Test Vectors

### 3.1. TCP Connection Parameters

The following parameters are used throughout this suite of test vectors. The terms 'active' and 'passive' are used as defined for TCP [RFC793].

#### 3.1.1. TCP-AO parameters

The following values are used for all exchanges. This suite does not test key switchover. The KeyIDs are as indicated for TCP-AO [RFC5925]. The Master Key is used to derive the traffic keys [RFC5926].

Active (client) side KeyID: 61 (3D)

Passive (server) side KeyID: 84 (54)

Master\_key: "testvector" (length = 10 bytes)

#### 3.1.2. Active (client) side parameters

The following endpoint parameters are used on the active side of the TCP connection, i.e., the side that initiates the TCP SYN.

For IPv4: 10.11.12.13

For IPv6: FD00::1

TCP port: (varies)

### 3.1.3. Passive (server) side parameters

The following endpoint parameters are used for the passive side of the TCP connection, i.e., the side that responds with a TCP SYN-ACK.

For IPv4: 172.27.28.29

For IPv6: FD00::2

TCP port = 179 (BGP)

### 3.1.4. Other IP fields and options

No IP options are used in these test vectors.

All IPv4 packets use the following other parameters [RFC791]: DSCP = 111000 (CS7) as is typical for BGP, ECN = 00, set DF, and clear MF.

IPv4 uses a TTL of 255; IPv6 uses a hopcount of 64.

All IPv6 use the following other parameters [RFC8200]: (TBD).

### 3.1.5. Other TCP fields and options

The SYN and SYN-ACK segments include MSS [RFC793], NOP, WindowScale [RFC7323], SACK Permitted [RFC2018], TimeStamp [RFC7323], and TCP-AO [RFC5925], in that order.

All other example segments include NOP, NOP, TimeStamp, and TCP-AO, in that order.

All segment URG pointers are zero [RFC793]. All segments with data set the PSH flag [RFC793].

## 4. IPv4 SHA-1 Output Test Vectors

SHA-1 is computed as specified for TCP-AO [RFC5926].

## 4.1. SHA-1 MAC (default - covers TCP options)

## 4.1.1. Send (client) SYN (covers options)

Send\_SYN\_traffic\_key:

```
6d 63 ef 1b 02 fe 15 09 d4 b1 40 27 07 fd 7b 04
16 ab b7 4f
```

IPv4/TCP:

```
45 e0 00 4c dd 0f 40 00 ff 06 bf 6b 0a 0b 0c 0d
ac 1b 1c 1d e9 d7 00 b3 fb fb ab 5a 00 00 00 00
e0 02 ff ff ca c4 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a 00 15 5a b7 00 00 00 00 1d 10 3d 54
2e e4 37 c6 f8 ed e6 d7 c4 d6 02 e7
```

MAC:

```
2e e4 37 c6 f8 ed e6 d7 c4 d6 02 e7
```

## 4.1.2. Receive (server) SYN-ACK (covers options)

Receive\_SYN\_traffic\_key:

```
d9 e2 17 e4 83 4a 80 ca 2f 3f d8 de 2e 41 b8 e6
79 7f ea 96
```

IPv4/TCP:

```
45 e0 00 4c 65 06 40 00 ff 06 37 75 ac 1b 1c 1d
0a 0b 0c 0d 00 b3 e9 d7 11 c1 42 61 fb fb ab 5b
e0 12 ff ff 37 76 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a 84 a5 0b eb 00 15 5a b7 1d 10 54 3d
ee ab 0f e2 4c 30 10 81 51 16 b3 be
```

MAC:

```
ee ab 0f e2 4c 30 10 81 51 16 b3 be
```

## 4.1.3. Send (client) non-SYN (covers options)

Send\_other\_traffic\_key:

```
d2 e5 9c 65 ff c7 b1 a3 93 47 65 64 63 b7 0e dc
24 a1 3d 71
```

IPv4/TCP:

```
45 e0 00 87 36 a1 40 00 ff 06 65 9f 0a 0b 0c 0d
ac 1b 1c 1d e9 d7 00 b3 fb fb ab 5b 11 c1 42 62
c0 18 01 04 a1 62 00 00 01 01 08 0a 00 15 5a c1
84 a5 0b eb 1d 10 3d 54 70 64 cf 99 8c c6 c3 15
c2 c2 e2 bf ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da bf 00 b4 0a 0b 0c 0d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da bf 02 08 40
06 00 64 00 01 01 00
```

MAC:

```
70 64 cf 99 8c c6 c3 15 c2 c2 e2 bf
```

## 4.1.4. Receive (server) non-SYN (covers options)

Receive\_other\_traffic\_key:

```
d9 e2 17 e4 83 4a 80 ca 2f 3f d8 de 2e 41 b8 e6
79 7f ea 96
```

IPv4/TCP:

```
45 e0 00 87 1f a9 40 00 ff 06 7c 97 ac 1b 1c 1d
0a 0b 0c 0d 00 b3 e9 d7 11 c1 42 62 fb fb ab 9e
c0 18 01 00 40 0c 00 00 01 01 08 0a 84 a5 0b f5
00 15 5a c1 1d 10 54 3d a6 3f 0e cb bb 2e 63 5c
95 4d ea c7 ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da c0 00 b4 ac 1b 1c 1d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da c0 02 08 40
06 00 64 00 01 01 00
```

MAC:

a6 3f 0e cb bb 2e 63 5c 95 4d ea c7

#### 4.2. SHA-1 MAC (omits TCP options)

##### 4.2.1. Send (client) SYN (omits options)

Send\_SYN\_traffic\_key:

30 ea a1 56 0c f0 be 57 da b5 c0 45 22 9f b1 0a  
42 3c d7 ea

IPv4/TCP:

45 e0 00 4c 53 99 40 00 ff 06 48 e2 0a 0b 0c 0d  
ac 1b 1c 1d ff 12 00 b3 cb 0e fb ee 00 00 00 00  
e0 02 ff ff 54 1f 00 00 02 04 05 b4 01 03 03 08  
04 02 08 0a 00 02 4c ce 00 00 00 00 1d 10 3d 54  
80 af 3c fe b8 53 68 93 7b 8f 9e c2

MAC:

80 af 3c fe b8 53 68 93 7b 8f 9e c2

##### 4.2.2. Receive (server) SYN-ACK (omits options)

Receive\_SYN\_traffic\_key:

b5 b2 89 6b b3 66 4e 81 76 b0 ed c6 e7 99 52 41a  
01 a8 30 7f

IPv4/TCP:

45 e0 00 4c 32 84 40 00 ff 06 69 f7 ac 1b 1c 1d  
0a 0b 0c 0d 00 b3 ff 12 ac d5 b5 e1 cb 0e fb ef  
e0 12 ff ff 38 8e 00 00 02 04 05 b4 01 03 03 08  
04 02 08 0a 57 67 72 f3 00 02 4c ce 1d 10 54 3d  
09 30 6f 9a ce a6 3a 8c 68 cb 9a 70

MAC:

09 30 6f 9a ce a6 3a 8c 68 cb 9a 70

#### 4.2.3. Send (client) non-SYN (omits options)

Send\_other\_traffic\_key:

f3 db 17 93 d7 91 0e cd 80 6c 34 f1 55 ea 1f 00  
34 59 53 e3

IPv4/TCP:

45 e0 00 87 a8 f5 40 00 ff 06 f3 4a 0a 0b 0c 0d  
ac 1b 1c 1d ff 12 00 b3 cb 0e fb ef ac d5 b5 e2  
c0 18 01 04 6c 45 00 00 01 01 08 0a 00 02 4c ce  
57 67 72 f3 1d 10 3d 54 71 06 08 cc 69 6c 03 a2  
71 c9 3a a5 ff ff ff ff ff ff ff ff ff ff  
ff ff ff ff 00 43 01 04 da bf 00 b4 0a 0b 0c 0d  
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02  
00 02 02 42 00 02 06 41 04 00 00 da bf 02 08 40  
06 00 64 00 01 01 00

MAC:

71 06 08 cc 69 6c 03 a2 71 c9 3a a5

#### 4.2.4. Receive (server) non-SYN (omits options)

Receive\_other\_traffic\_key:

b5 b2 89 6b b3 66 4e 81 76 b0 ed c6 e7 99 52 41  
01 a8 30 7f

## IPv4/TCP:

```

45 e0 00 87 54 37 40 00 ff 06 48 09 ac 1b 1c 1d
0a 0b 0c 0d 00 b3 ff 12 ac d5 b5 e2 cb 0e fc 32
c0 18 01 00 46 b6 00 00 01 01 08 0a 57 67 72 f3
00 02 4c ce 1d 10 54 3d 97 76 6e 48 ac 26 2d e9
ae 61 b4 f9 ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da c0 00 b4 ac 1b 1c 1d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da c0 02 08 40
06 00 64 00 01 01 00

```

## MAC:

```

97 76 6e 48 ac 26 2d e9 ae 61 b4 f9

```

## 5. IPv4 AES-128 Output Test Vectors

AES-128 is computed as required by TCP-AO [RFC5926].

## 5.1. AES MAC (default - covers TCP options)

## 5.1.1. Send (client) SYN (covers options)

## Send\_SYN\_traffic\_key:

```

f5 b8 b3 d5 f3 4f db b6 eb 8d 4a b9 66 0e 60 e3

```

## IP/TCP:

```

45 e0 00 4c 7b 9f 40 00 ff 06 20 dc 0a 0b 0c 0d
ac 1b 1c 1d c4 fa 00 b3 78 7a 1d df 00 00 00 00
e0 02 ff ff 5a 0f 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a 00 01 7e d0 00 00 00 00 1d 10 3d 54
e4 77 e9 9c 80 40 76 54 98 e5 50 91

```

## MAC:

```

e4 77 e9 9c 80 40 76 54 98 e5 50 91

```

## 5.1.2. Receive (server) SYN-ACK (covers options)

Receive\_SYN\_traffic\_key:

4b c7 57 1a 48 6f 32 64 bb d8 88 47 40 66 b4 b1

IPv4/TCP:

```
45 e0 00 4c 4b ad 40 00 ff 06 50 ce ac 1b 1c 1d
0a 0b 0c 0d 00 b3 c4 fa fa dd 6d e9 78 7a 1d e0
e0 12 ff ff f3 f2 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a 93 f4 e9 e8 00 01 7e d0 1d 10 54 3d
d6 ad a7 bc 4c dd 53 6d 17 69 db 5f
```

MAC:

d6 ad a7 bc 4c dd 53 6d 17 69 db 5f

## 5.1.3. Send (client) non-SYN (covers options)

Send\_other\_traffic\_key:

8c 8a e0 e8 37 1e c5 cb b9 7e a7 9d 90 41 83 91

IPv4/TCP:

```
45 e0 00 87 fb 4f 40 00 ff 06 a0 f0 0a 0b 0c 0d
ac 1b 1c 1d c4 fa 00 b3 78 7a 1d e0 fa dd 6d ea
c0 18 01 04 95 05 00 00 01 01 08 0a 00 01 7e d0
93 f4 e9 e8 1d 10 3d 54 77 41 27 42 fa 4d c4 33
ef f0 97 3e ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da bf 00 b4 0a 0b 0c 0d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da bf 02 08 40
06 00 64 00 01 01 00
```

MAC:

77 41 27 42 fa 4d c4 33 ef f0 97 3e



## 5.1.4. Receive (server) non-SYN (covers options)

Receive\_other\_traffic\_key:

4b c7 57 1a 48 6f 32 64 bb d8 88 47 40 66 b4 b1

IPv4/TCP:

```
45 e0 00 87 b9 14 40 00 ff 06 e3 2b ac 1b 1c 1d
0a 0b 0c 0d 00 b3 c4 fa fa dd 6d ea 78 7a 1e 23
c0 18 01 00 e7 db 00 00 01 01 08 0a 93 f4 e9 e8
00 01 7e d0 1d 10 54 3d f6 d9 65 a7 83 82 a7 48
45 f7 2d ac ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da c0 00 b4 ac 1b 1c 1d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da c0 02 08 40
06 00 64 00 01 01 00
```

MAC:

f6 d9 65 a7 83 82 a7 48 45 f7 2d ac

## 5.2. AES MAC (omits TCP options)

## 5.2.1. Send (client) SYN (omits options)

Send\_SYN\_traffic\_key:

2c db ae 13 92 c4 94 49 fa 92 c4 50 97 35 d5 0e

IPv4/TCP:

```
45 e0 00 4c f2 2e 40 00 ff 06 aa 4c 0a 0b 0c 0d
ac 1b 1c 1d da 1c 00 b3 38 9b ed 71 00 00 00 00
e0 02 ff ff 70 bf 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a 00 01 85 e1 00 00 00 00 1d 10 3d 54
c4 4e 60 cb 31 f7 c0 b1 de 3d 27 49
```

MAC:

c4 4e 60 cb 31 f7 c0 b1 de 3d 27 49

## 5.2.2. Receive (server) SYN-ACK (omits options)

Receive\_SYN\_traffic\_key:

3c e6 7a 55 18 69 50 6b 63 47 b6 33 c5 0a 62 4a

IPv4/TCP:

```
45 e0 00 4c 6c c0 40 00 ff 06 2f bb ac 1b 1c 1d
0a 0b 0c 0d 00 b3 da 1c d3 84 4a 6f 38 9b ed 72
e0 12 ff ff e4 45 00 00 02 04 05 b4 01 03 03 08
04 02 08 0a ce 45 98 38 00 01 85 e1 1d 10 54 3d
3a 6a bb 20 7e 49 b1 be 71 36 db 90
```

MAC:

3a 6a bb 20 7e 49 b1 be 71 36 db 90

## 5.2.3. Send (client) non-SYN (omits options)

Send\_other\_traffic\_key:

03 5b c4 00 a3 41 ff e5 95 f5 9f 58 00 50 06 ca

IPv4/TCP:

```
45 e0 00 87 ee 91 40 00 ff 06 ad ae 0a 0b 0c 0d
ac 1b 1c 1d da 1c 00 b3 38 9b ed 72 d3 84 4a 70
c0 18 01 04 88 51 00 00 01 01 08 0a 00 01 85 e1
ce 45 98 38 1d 10 3d 54 75 85 e9 e9 d5 c3 ec 85
7b 96 f8 37 ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da bf 00 b4 0a 0b 0c 0d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da bf 02 08 40
06 00 64 00 01 01 00
```

MAC:

75 85 e9 e9 d5 c3 ec 85 7b 96 f8 37

## 5.2.4. Receive (server) non-SYN (omits options)

Receive\_other\_traffic\_key:

3c e6 7a 55 18 69 50 6b 63 47 b6 33 c5 0a 62 4a

IPv4/TCP:

```

45 e0 00 87 6a 21 40 00 ff 06 32 1f ac 1b 1c 1d
0a 0b 0c 0d 00 b3 da 1c d3 84 4a 70 38 9b ed 72
c0 18 01 00 04 49 00 00 01 01 08 0a ce 45 98 38
00 01 85 e1 1d 10 54 3d 5c 04 0f d9 23 33 04 76
5c 09 82 f4 ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff 00 43 01 04 da c0 00 b4 ac 1b 1c 1d
26 02 06 01 04 00 01 00 01 02 02 80 00 02 02 02
00 02 02 42 00 02 06 41 04 00 00 da c0 02 08 40
06 00 64 00 01 01 00

```

MAC:

5c 04 0f d9 23 33 04 76 5c 09 82 f4

## 6. IPv6 SHA-1 Output Test Vectors

SHA-1 is computed as specified for TCP-AO [RFC5926].

## 6.1. SHA-1 MAC (default - covers TCP options)

## 6.1.1. Send (client) SYN (covers options)

Send\_SYN\_traffic\_key:

```

62 5e c0 9d 57 58 36 ed c9 b6 42 84 18 bb f0 69
89 a3 61 bb

```

IPv6/TCP:

```

6e 08 91 dc 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00
00 00 00 00 00 00 00 00 02 f7 e4 00 b3 17 6a 83 3f
00 00 00 00 e0 02 ff ff 47 21 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a 00 41 d0 87 00 00 00 00
1d 10 3d 54 90 33 ec 3d 73 34 b6 4c 5e dd 03 9f

```

MAC:

90 33 ec 3d 73 34 b6 4c 5e dd 03 9f

#### 6.1.2. Receive (server) SYN-ACK (covers options)

Receive\_SYN\_traffic\_key:

e4 a3 7a da 2a 0a fc a8 71 14 34 91 3f e1 38 c7  
71 eb cb 4a

IPv6/TCP:

6e 01 00 9e 00 38 06 40 fd 00 00 00 00 00 00 00  
00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00 00  
00 00 00 00 00 00 00 01 00 b3 f7 e4 3f 51 99 4b  
17 6a 83 40 e0 12 ff ff bf ec 00 00 02 04 05 a0  
01 03 03 08 04 02 08 0a bd 33 12 9b 00 41 d0 87  
1d 10 54 3d f1 cb a3 46 c3 52 61 63 f7 1f 1f 55

MAC:

f1 cb a3 46 c3 52 61 63 f7 1f 1f 55

#### 6.1.3. Send (client) non-SYN (covers options)

Send\_other\_traffic\_key:

1e d8 29 75 f4 ea 44 4c 61 58 0c 5b d9 0d bd 61  
bb c9 1b 7e

## IPv6/TCP:

```

6e 08 91 dc 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00
00 00 00 00 00 00 00 00 02 f7 e4 00 b3 17 6a 83 40
3f 51 99 4c c0 18 01 00 32 9c 00 00 01 01 08 0a
00 41 d0 91 bd 33 12 9b 1d 10 3d 54 bf 08 05 fe
b4 ac 7b 16 3d 6f cd f2 ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 79 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00

```

## MAC:

```

bf 08 05 fe b4 ac 7b 16 3d 6f cd f2

```

## 6.1.4. Receive (server) non-SYN (covers options)

## Receive\_other\_traffic\_key:

```

e4 a3 7a da 2a 0a fc a8 71 14 34 91 3f e1 38 c7
71 eb cb 4a

```

## IPv6/TCP:

```

6e 01 00 9e 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00
00 00 00 00 00 00 00 00 01 00 b3 f7 e4 3f 51 99 4c
17 6a 83 83 c0 18 01 00 ee 6e 00 00 01 01 08 0a
bd 33 12 a5 00 41 d0 91 1d 10 54 3d 6c 48 12 5c
11 33 5b ab 9a 07 a7 97 ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 7a 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00

```

## MAC:

```

6c 48 12 5c 11 33 5b ab 9a 07 a7 97

```

## 6.2. SHA-1 MAC (omits TCP options)

## 6.2.1. Send (client) SYN (omits options)

Send\_SYN\_traffic\_key:

```
31 a3 fa f6 9e ff ae 52 93 1b 7f 84 54 67 31 5c
27 0a 4e dc
```

IPv6/TCP:

```
6e 07 8f cd 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 c6 cd 00 b3 02 0c 1e 69
00 00 00 00 e0 02 ff ff a4 1a 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a 00 9d b9 5b 00 00 00 00
1d 10 3d 54 88 56 98 b0 53 0e d4 d5 a1 5f 83 46
```

MAC:

```
88 56 98 b0 53 0e d4 d5 a1 5f 83 46
```

## 6.2.2. Receive (server) SYN-ACK (omits options)

Receive\_SYN\_traffic\_key:

```
40 51 08 94 7f 99 65 75 e7 bd bc 26 d4 02 16 a2
c7 fa 91 bd
```

IPv6/TCP:

```
6e 0a 7e 1f 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 b3 c6 cd eb a3 73 4d
02 0c 1e 6a e0 12 ff ff 77 4d 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a 5e c9 9b 70 00 9d b9 5b
1d 10 54 3d 3c 54 6b ad 97 43 f1 2d f8 b8 01 0d
```

MAC:

```
3c 54 6b ad 97 43 f1 2d f8 b8 01 0d
```

## 6.2.3. Send (client) non-SYN (omits options)

Send\_other\_traffic\_key:

```
b3 4e ed 6a 93 96 a6 69 f1 c4 f4 f5 76 18 f3 65
6f 52 c7 ab
```

IPv6/TCP:

```
6e 07 8f cd 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 02 c6 cd 00 b3 02 0c 1e 6a
eb a3 73 4e c0 18 01 00 83 e6 00 00 01 01 08 0a
00 9d b9 65 5e c9 9b 70 1d 10 3d 54 48 bd 09 3b
19 24 e0 01 19 2f 5b f0 ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 79 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00
```

MAC:

```
48 bd 09 3b 19 24 e0 01 19 2f 5b f0
```

## 6.2.4. Receive (server) non-SYN (omits options)

Receive\_other\_traffic\_key:

```
40 51 08 94 7f 99 65 75 e7 bd bc 26 d4 02 16 a2
c7 fa 91 bd
```

## IPv6/TCP:

```

6e 0a 7e 1f 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00
00 00 00 00 00 00 00 00 01 00 b3 c6 cd eb a3 73 4e
02 0c 1e ad c0 18 01 00 71 6a 00 00 01 01 08 0a
5e c9 9b 7a 00 9d b9 65 1d 10 54 3d 55 9a 81 94
45 b4 fd e9 8d 9e 13 17 ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 7a 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00

```

## MAC:

```
55 9a 81 94 45 b4 fd e9 8d 9e 13 17
```

## 7. IPv6 AES-128 Output Test Vectors

AES-128 is computed as required by TCP-AO [RFC5926].

## 7.1. AES MAC (default - covers TCP options)

## 7.1.1. Send (client) SYN (covers options)

## Send\_SYN\_traffic\_key:

```
fa 5a 21 08 88 2d 39 d0 c7 19 29 17 5a b1 b7 b8
```

## IP/TCP:

```

6e 04 a7 06 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00
00 00 00 00 00 00 00 00 02 f8 5a 00 b3 19 3c cc ec
00 00 00 00 e0 02 ff ff de 5d 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a 13 e4 ab 99 00 00 00 00
1d 10 3d 54 59 b5 88 10 74 81 ac 6d c3 92 70 40

```

## MAC:

```
59 b5 88 10 74 81 ac 6d c3 92 70 40
```



## 7.1.2. Receive (server) SYN-ACK (covers options)

Receive\_SYN\_traffic\_key:

cf 1b 1e 22 5e 06 a6 36 16 76 4a 06 7b 46 f4 b1

IPv6/TCP:

```
6e 06 15 20 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00
00 00 00 00 00 00 00 00 01 00 b3 f8 5a a6 74 4e cb
19 3c cc ed e0 12 ff ff ea bb 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a 71 da ab c8 13 e4 ab 99
1d 10 54 3d dc 28 43 a8 4e 78 a6 bc fd c5 ed 80
```

MAC:

dc 28 43 a8 4e 78 a6 bc fd c5 ed 80

## 7.1.3. Send (client) non-SYN (covers options)

Send\_other\_traffic\_key:

61 74 c3 55 7a be d2 75 74 db a3 71 85 f0 03 00

IPv6/TCP:

```
6e 04 a7 06 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00
00 00 00 00 00 00 00 00 02 f8 5a 00 b3 19 3c cc ed
a6 74 4e cc c0 18 01 00 32 80 00 00 01 01 08 0a
13 e4 ab a3 71 da ab c8 1d 10 3d 54 7b 6a 45 5c
0d 4f 5f 01 83 5b aa b3 ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 79 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00
```

MAC:

7b 6a 45 5c 0d 4f 5f 01 83 5b aa b3

## 7.1.4. Receive (server) non-SYN (covers options)

Receive\_other\_traffic\_key:

cf 1b 1e 22 5e 06 a6 36 16 76 4a 06 7b 46 f4 b1

IPv6/TCP:

```
6e 06 15 20 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00
00 00 00 00 00 00 00 00 01 00 b3 f8 5a a6 74 4e cc
19 3c cd 30 c0 18 01 00 52 f4 00 00 01 01 08 0a
71 da ab d3 13 e4 ab a3 1d 10 54 3d c1 06 9b 7d
fd 3d 69 3a 6d f3 f2 89 ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 7a 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00
```

MAC:

c1 06 9b 7d fd 3d 69 3a 6d f3 f2 89

## 7.2. AES MAC (omits TCP options)

## 7.2.1. Send (client) SYN (omits options)

Send\_SYN\_traffic\_key:

a9 4f 51 12 63 e4 09 3d 35 dd 81 8c 13 bb bf 53

IPv6/TCP:

```
6e 09 3d 76 00 38 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00
00 00 00 00 00 00 00 00 02 f2 88 00 b3 b0 1d a7 4a
00 00 00 00 e0 02 ff ff 75 ff 00 00 02 04 05 a0
01 03 03 08 04 02 08 0a 14 27 5b 3b 00 00 00 00
1d 10 3d 54 3d 45 b4 34 2d e8 bb 15 30 84 78 98
```

MAC:

3d 45 b4 34 2d e8 bb 15 30 84 78 98

#### 7.2.2. Receive (server) SYN-ACK (omits options)

Receive\_SYN\_traffic\_key:

92 de a5 bb c7 8b 1d 9f 5b 29 52 e9 cd 30 64 2a

IPv6/TCP:

6e 0c 60 0a 00 38 06 40 fd 00 00 00 00 00 00 00  
00 00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00  
00 00 00 00 00 00 00 00 01 00 b3 f2 88 a6 24 61 45  
b0 1d a7 4b e0 12 ff ff a7 0c 00 00 02 04 05 a0  
01 03 03 08 04 02 08 0a 17 82 24 5b 14 27 5b 3b  
1d 10 54 3d 1d 01 f6 c8 7c 6f 93 ac ff a9 d4 b5

MAC:

1d 01 f6 c8 7c 6f 93 ac ff a9 d4 b5

#### 7.2.3. Send (client) non-SYN (omits options)

Send\_other\_traffic\_key:

4f b2 08 6e 40 2c 67 90 79 ed 65 d4 bf 97 69 3d

IPv6/TCP:

6e 09 3d 76 00 73 06 40 fd 00 00 00 00 00 00 00  
00 00 00 00 00 00 00 00 01 fd 00 00 00 00 00 00  
00 00 00 00 00 00 00 00 02 f2 88 00 b3 b0 1d a7 4b  
a6 24 61 46 c0 18 01 00 c3 6d 00 00 01 01 08 0a  
14 27 5b 4f 17 82 24 5b 1d 10 3d 54 29 0c f4 14  
cc b4 7a 33 32 76 e7 f8 ff ff ff ff ff ff ff  
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4  
01 01 01 79 26 02 06 01 04 00 01 00 01 02 02 80  
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd  
e8 02 08 40 06 00 64 00 01 01 00

MAC:

29 0c f4 14 cc b4 7a 33 32 76 e7 f8

#### 7.2.4. Receive (server) non-SYN (omits options)

Receive\_other\_traffic\_key:

92 de a5 bb c7 8b 1d 9f 5b 29 52 e9 cd 30 64 2a

IPv6/TCP:

```
6e 0c 60 0a 00 73 06 40 fd 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 02 fd 00 00 00 00 00 00
00 00 00 00 00 00 00 00 01 00 b3 f2 88 a6 24 61 46
b0 1d a7 8e c0 18 01 00 34 51 00 00 01 01 08 0a
17 82 24 65 14 27 5b 4f 1d 10 54 3d 99 51 5f fc
d5 40 34 99 f6 19 fd 1b ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff 00 43 01 04 fd e8 00 b4
01 01 01 7a 26 02 06 01 04 00 01 00 01 02 02 80
00 02 02 02 00 02 02 42 00 02 06 41 04 00 00 fd
e8 02 08 40 06 00 64 00 01 01 00
```

MAC:

99 51 5f fc d5 40 34 99 f6 19 fd 1b

## 8. Observed Implementation Errors

The following is a partial list of implementation errors that this set of test vectors is intended to validate.

### 8.1. Algorithm issues

- o Underlying implementation of HMAC SHA1 or AES128 CMAC does not pass their corresponding test vectors [RFC2202] [RFC4493]
- o SNE algorithm does not consider corner cases (the pseudocode in [RFC5925] was not intended as complete, as discussed in [To20])

## 8.2. Algorithm parameters

- o KDF context length is incorrect, e.g. it does not include TCP header length + payload length (it should, per 5.2 of TCP-AO [RFC5925])
- o KDF calculation does not start from counter  $i = 1$  (it should, per Sec. 3.1.1 of the TCP-AO crypto algorithms [RFC5926])
- o KDF calculation does not include output length in bits, contained in two bytes in network byte order (it should, per Sec. 3.1.1 of the TCP-AO crypto algorithms [RFC5926])
- o KDF uses keys generated from current TCP segment sequence numbers (KDF should use only local and remote ISNs or zero, as indicated in Sec. 5.2 of TCP-AO [RFC5925])

## 8.3. String handling issues

The strings indicated in TCP-AO and its algorithms are indicated as a sequence of bytes of known length. In some implementations, string lengths are indicated by a terminal value (e.g., zero in C). This terminal value is not included as part of the string for calculations.

- o Password includes the last zero-byte (it should not)
- o Label "TCP-AO" includes the last zero byte (it should not)

## 8.4. Header coverage issues

- o TCP checksum and/or MAC is not zeroed properly before calculation (both should be)
- o TCP header is not included to the MAC calculation (it should be)
- o TCP options are not included to the MAC calculation by default (there is a separate parameter in the master key tuple to ignore options; this document provides test vectors for both options-included and options-excluded cases)

## 9. Security Considerations

This document is intended to assist in the validation of implementations of TCP-AO, to further enable its more widespread use as a security mechanism to authenticate not only TCP payload contents but the TCP headers and protocol.

The master\_key of "testvector" used here for test vector generation SHOULD NOT be used operationally.

## 10. IANA Considerations

This document contains no IANA issues. This section should be removed upon publication as an RFC.

## 11. References

### 11.1. Normative References

- [RFC791] Postel, J., "Internet Protocol," RFC 791, Sept. 1981.
- [RFC793] Postel, J., "Transmission Control Protocol," RFC 793, September 1981.
- [RFC2018] Mathis, M., J. Mahdavi, S. Floyd, A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018, Oct. 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997.
- [RFC5925] Touch, J., A. Mankin, R. Bonica, "The TCP Authentication Option," RFC 5925, June 2010.
- [RFC5926] Lebovitz, G., and E. Rescorla, "Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)," RFC 5925, June 2010.
- [RFC6978] Touch, J., "A TCP Authentication Option Extension for NAT Traversal," RFC 6978, July 2013.
- [RFC7323] Borman, D., B. Braden, V. Jacobson, R. Scheffenegger, Ed., "TCP Extensions for High Performance," RFC 7323, Sept. 2014.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words," RFC 2119, May 2017.
- [RFC8200] Deering, S., R. Hinden, "Internet Protocol Version 6 (IPv6) Specification," RFC 8200, Jul. 2017.

### 11.2. Informative References

- [RFC2202] Cheng, P., and R. Glenn, "Test Cases for HMAC-MD5 and HMAC-SHA-1," RFC 2202, Sept. 1997.

[RFC4493] Song, JH, R. Poovendran, J. Lee, T. Iwata, "The AES-CMAC Algorithm," RFC 4493, June 2006.

[To20] Touch, J., "Sequence Number Extension for Windowed Protocols," draft-tsvwg-touch-sne, Jun. 2020.

## 12. Acknowledgments

(TBD)

This document was prepared using 2-Word-v2.0.template.dot.

## Authors' Addresses

Joe Touch  
Manhattan Beach, CA 90266 USA  
Phone: +1 (310) 560-0334  
Email: touch@strayalpha.com

Juhamatti Kuusisaari  
Infinera Corporation  
Sinimaentie 6c  
FI-02630 Espoo, Finland  
Email: jkuusisaari@infinera.com

