Stream:	Internet Engineering Task Force (IETF)		
RFC:	9768		
Updates:	3168		
Category:	Standards Track		
Published:	August 2025		
ISSN:	2070-1721		
Authors:	B. Briscoe	M. Kühlewind	R. Scheffenegger
	Independent	*Ericsson*	*NetApp*

RFC 9768
More Accurate Explicit Congestion Notification (AccECN) Feedback in TCP

Abstract

Explicit Congestion Notification (ECN) is a mechanism by which network nodes can mark IP packets instead of dropping them to indicate incipient congestion to the endpoints. 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). More recently defined TCP mechanisms like Congestion Exposure (ConEx), Data Center TCP (DCTCP), or Low Latency, Low Loss, and Scalable Throughput (L4S) need more Accurate ECN (AccECN) feedback information whenever more than one marking is received in one RTT. This document updates the original ECN specification defined in RFC 3168 by specifying a scheme that provides 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 additionally exploited to feed back the IP ECN field received during the TCP connection establishment. Supplementary feedback information can optionally be provided in two new TCP Option alternatives, which are never used on the TCP SYN. The document also specifies the treatment of this updated TCP wire protocol by middleboxes.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9768.

Copyright Notice

Copyright (c) 2025 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 4
   1.1. Document Roadmap 6
   1.2. Goals 6
   1.3. Terminology 6
   1.4. Recap of Existing ECN Feedback in IP/TCP 7
2. AccECN Protocol Overview and Rationale 8
   2.1. Capability Negotiation 9
   2.2. Feedback Mechanism 10
   2.3. Delayed ACKs and Resilience Against ACK Loss 10
   2.4. Feedback Metrics 11
   2.5. Generic (Mechanistic) Reflector 11
3. AccECN Protocol Specification 12
   3.1. Negotiating to Use AccECN 12
      3.1.1. Negotiation During the TCP Three-Way Handshake 12

3.1.2. Backward Compatibility 13
3.1.3. Forward Compatibility 15
3.1.4. Multiple SYNs or SYN/ACKs 15
3.1.4.1. Retransmitted SYNs 15
3.1.4.2. Retransmitted SYN/ACKs 17
3.1.5. Implications of AccECN Mode 17
3.2. AccECN Feedback 20
3.2.1. Initialization of Feedback Counters 21
3.2.2. The ACE Field 21
3.2.2.1. ACE Field on the ACK of the SYN/ACK 22
3.2.2.2. Encoding and Decoding Feedback in the ACE Field 24
3.2.2.3. Testing for Mangling of the IP/ECN Field 25
3.2.2.4. Testing for Zeroing of the ACE Field 27
3.2.2.5. Safety Against Ambiguity of the ACE Field 28
3.2.3. The AccECN Option 30
3.2.3.1. Encoding and Decoding Feedback in the AccECN Option Fields 32
3.2.3.2. Path Traversal of the AccECN Option 32
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 and Large Receive Offload 38
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 41
5.3. Compatibility with Feedback Integrity Mechanisms 42
6. Summary: Protocol Properties 43

7. IANA Considerations 44
8. Security and Privacy Considerations 45
9. References 46
9.1. Normative References 46
9.2. Informative References 47
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 51
A.2.2. Safety Algorithm with the AccECN Option 53
A.3. Example Algorithm to Estimate Marked Bytes from Marked Packets 54
A.4. Example Algorithm to Count Not-ECT Bytes 54
Appendix B. Rationale for Usage of TCP Header Flags 55
B.1. Three TCP Header Flags in the SYN-SYN/ACK Handshake 55
B.2. Four Codepoints in the SYN/ACK 56
B.3. Space for Future Evolution 56
Acknowledgements 57
Authors' Addresses 58

1. Introduction

Explicit Congestion Notification (ECN) [RFC3168] is a mechanism by which network nodes can mark IP packets instead of dropping them to indicate incipient congestion to the endpoints. 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). This is sufficient for congestion control schemes like Reno [RFC6582] and CUBIC [RFC9438], as those schemes reduce their congestion window by a fixed factor if congestion occurs within an RTT independent of the number of received congestion markings. More recently defined mechanisms like Congestion Exposure (ConEx [RFC7713]), DCTCP [RFC8257], and L4S [RFC9330] 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, while still also supporting the pre-existing TCP congestion controllers that

use just one feedback signal per round. Congestion control is the term the IETF uses to describe data rate management. It is the algorithm that a sender uses to optimize its sending rate so that it transmits data as fast as the network can carry it, but no faster. A fuller description 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 is 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 and its negotiation remain unchanged by this specification. In particular, the definition of ECN at the IP layer is unaffected. Section 4 details the aspects of RFC 3168 that are updated by this document.

This document uses the term "Classic ECN feedback" when it needs to distinguish the TCP/ECN feedback scheme defined in [RFC3168] from the AccECN TCP feedback scheme. AccECN is intended to offer 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 hosts using Classic TCP/ECN feedback. So its applicability is intended to include the public Internet as well as private IP networks such as data centres (and even any non-IP networks over which TCP is used), whether or not any nodes on the path support ECN, of whatever flavour.

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 3-bit counter field for feeding back the number of packets marked as congestion experienced (CE). 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 or CUBIC congestion control [RFC5681] [RFC9438] to respond to the existence of at least one congestion notification within a round trip. Or, unlike Reno or CUBIC, 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 the original ECN specification adopted in 2001 [RFC3168], as updated by the more relaxed rules introduced in 2018 to allow ECN experiments [RFC8311], namely: a TCP-based Low Latency Low Loss Scalable (L4S) congestion control [RFC9330]; or Alternative Backoff with ECN (ABE) [RFC8511].

Section 5.2 explains how AccECN is compatible with current commonly used TCP Options, and a number of current experimental modifications to TCP, as well as SYN cookies.

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 3.3 collects requirements for proxies, offload engines, and other middleboxes. Section 4 clarifies which aspects of RFC 3168 are updated by AccECN. Section 5 assesses the interaction of AccECN with commonly used variants of TCP, whether they are standardized or not. Then Section 6 summarizes the features and properties of AccECN.

Section 7 summarizes the protocol fields and numbers that IANA assigned, 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 needs 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 considers the properties of AccECN against these requirements and discusses the trade-offs.

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 acts as a generic (mechanistic) reflector of congestion information with the aim that new sender behaviours can be deployed unilaterally (see Section 2.5) in the future.

1.3. Terminology

AccECN: The more Accurate ECN feedback scheme is 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 [RFC9293] and [RFC5961], or that has passed other tests with equivalent protection.

TCP Client: The TCP stack that originates a connection (the initiator).

TCP Server: The TCP stack that responds to a connection request (the listener).

Three-way handshake: The procedure used to establish a TCP connection as described in the TCP protocol specification [RFC9293].

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.

In a mild abuse of terminology, this document sometimes refers to 'TCP packets' instead of 'TCP segments'.

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.

1.4. Recap of Existing ECN Feedback in IP/TCP

Explicit Congestion Notification (ECN) [RFC3168] can be split into two parts conceptionally. In the forward direction, alongside the data stream, it uses a 2-bit field in the IP header. This is referred to as IP ECN later on. This signal carried in the IP (Layer 3) header is exposed to network devices and may be modified when such a device starts to experience congestion (see Table 1). The second part is the feedback mechanism, by which the original data sender is notified of the current congestion state of the intermediate path. That returned signal is carried in a protocol-specific manner, and is not to be modified by intermediate network devices. While ECN is in active use for protocols such as QUIC [RFC9000], SCTP [RFC9260], RTP [RFC6679], and Remote Direct Memory Access over Converged Ethernet [RoCEv2], this document only concerns itself with the specific implementation for the TCP protocol.

Once ECN has been negotiated for a transport layer connection, the Data Sender for either half-connection can set two possible codepoints (ECT(0) or ECT(1)) in the IP header of a data packet to indicate an ECN-capable transport (ECT). If the ECN codepoint is 0b00, 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 the ECN codepoint to 0b11, 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 (the TCP header flags at bit offsets 8 and 9 labelled Congestion Window Reduced (CWR) and Explicit Congestion notification Echo (ECE) in Figure 1) are defined as flags for the use of Classic ECN [RFC3168]. A TCP Client indicates that it supports Classic ECN feedback by setting (CWR,ECE) = (1,1) in the SYN, and an ECN-enabled TCP Server confirms Classic ECN support by setting (CWR,ECE) = (0,1) in the SYN/ACK. On reception of a CE-marked packet at the IP layer, the Data Receiver for that half-connection starts to set the Echo Congestion Experienced (ECE) flag continuously in the TCP header of ACKs, which gives the signal resilience to loss or reordering of ACKs. The Data Sender for the same half-connection confirms that it has received at least one ECE signal by responding with the CWR flag, which allows the Data Receiver to stop repeating the ECN-Echo flag. This always leads to a full RTT of ACKs with ECE set. Thus Classic ECN cannot feed back any additional CE markings arriving within this RTT.

The last bit in byte 13 of the TCP header (the TCP header flag at bit offset 7 in Figure 1) was defined as the Nonce Sum (NS) for the ECN-nonce [RFC3540]. In the absence of widespread deployment, RFC 3540 was reclassified as Historic [RFC8311] and the respective flag was marked as "Reserved", which made this TCP flag available for use by AccECN instead.

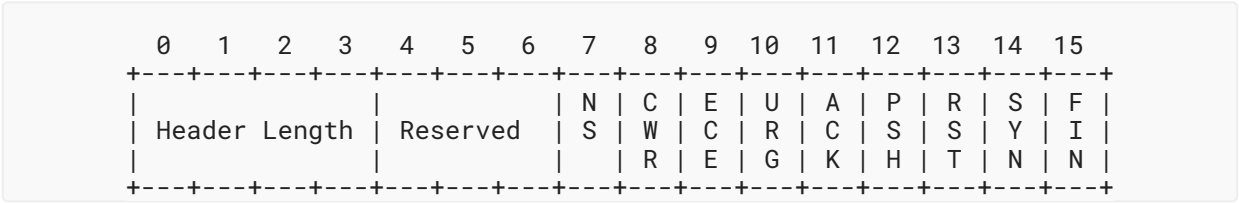

Figure 1: TCP Header Flags as Defined Before the Nonce Sum Flag Reverted to Reserved

2. AccECN Protocol Overview and Rationale

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

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

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

- an essential feedback part that reuses the TCP-ECN 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.
- a supplementary feedback part using one of two new alternative AccECN TCP Options that provide additional feedback on the number of payload bytes that arrive marked with each of the three ECN codepoints in the IP ECN field (not just CE marks). See the BCP on Byte and Packet Congestion Notification [RFC7141] for the rationale determining that conveying congested payload bytes should be preferred over just providing feedback about congested packets. This also provides greater resilience against ACK loss than the essential feedback, but it is currently 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 might prevent TCP from using any new options.

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

- this efficiently reuses scarce TCP header space, given TCP Option space is approaching saturation.
- a single upgrade path for the TCP protocol is preferable to a fork in the design that modifies the TCP header to convey all ECN feedback.
- otherwise, Classic and Accurate ECN feedback could give conflicting feedback about the same segment, which could open up new security concerns and make implementations unnecessarily complex.
- 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 feedback part is removed or zeroed out, as long as the essential feedback part gets through.

2.1. Capability Negotiation

AccECN changes 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 TCP 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 TCP Client falls back to the same variant of ECN feedback.

An AccECN TCP Client does not send an AccECN Option on the SYN as SYN option space is limited. The TCP Server sends an AccECN Option on the SYN/ACK, and the TCP Client sends one on the first ACK to test whether the network path forwards these options 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) in the IP ECN field. These byte counters reflect only the TCP payload length, excluding the TCP header and TCP Options. The fourth counter 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). The 24 LSBs of some or all of the byte counters can be optionally carried in an AccECN Option. For efficient use of limited option space, two alternative forms of the AccECN Option are specified with the fields in the opposite order to each other.

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 of [RFC3168] 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 and 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, ACKs received at the Data Sender could still cycle because a whole sequence of ACKs carrying intervening values of the field might all be lost or delayed in transit.

The fields in an 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 has been an infeasibly long sequence of ACK losses. Therefore, provided that an AccECN Option is available, it can be treated as a dependable feedback channel.

If an 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 AccECN Options 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 an AccECN Option will provide the more accurate information needed for modern congestion control and policing schemes, such as L4S, DCTCP, or ConEx. If AccECN Options are stripped, a simple algorithm to estimate the number of marked bytes from the ACE field is given in Appendix A.3.

The AccECN design has been generalized so that it ought to be able to support possible future uses of the experimental ECT(1) codepoint other than the L4S experiment [RFC9330], such as a lower severity or a more instant congestion signal than CE.

Feedback in bytes is provided to protect against the receiver or a middlebox using attacks similar to 'ACK-Division' to artificially inflate the congestion window, which is why [RFC5681] now recommends that TCP counts acknowledge bytes not packets.

2.5. Generic (Mechanistic) 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 Sections 3.2.2.3, 3.2.2.4, and 3.2.3.2 of the present document and paragraph 2 of Section 20.2 of [RFC3168]).

The initial SYN and SYN/ACK are the most critical control packets, so AccECN feeds back their IP ECN fields. Although RFC 3168 prohibits ECN-capable SYNs and SYN/ACKs, providing feedback of ECN marking on the SYN and SYN/ACK supports future scenarios in which SYNs might be ECN-enabled (without prejudging whether they ought to be). For instance, [RFC8311] updates this aspect of RFC 3168 to allow experimentation with ECN-capable TCP control packets.

Even if the TCP Client (or Server) has set the SYN (or SYN/ACK) to Not-ECT in compliance with RFC 3168, feedback on the state of the IP ECN field when it arrives at the receiver could still be useful, because middleboxes have been known to overwrite the IP ECN field as if it is still part of the old Type of Service (ToS) field [Mandalari18]. 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 sent. So, if a TCP Server received ECT or CE on a SYN, it could not know whether it was invalid 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 and Client to feed back the ECN field received on the SYN and SYN/ACK to their peer, which now 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 Three-Way Handshake

Given the ECN-nonce [RFC3540] has been reclassified as Historic [RFC8311], the TCP flag that was previously called NS (Nonce Sum) is renamed as the AE (Accurate ECN) flag (the TCP header flag at bit offset 7 in Figure 2). See the IANA Considerations in Section 7.

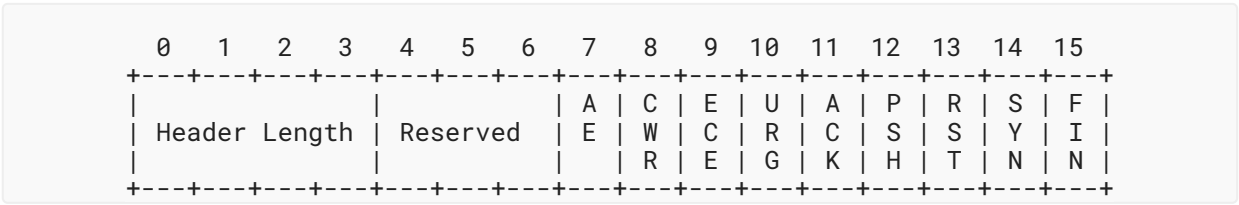

Figure 2: The New Definition of the TCP Header Flags During the TCP Three-Way Handshake

During the TCP three-way handshake at the start of a connection, to request more Accurate ECN feedback the TCP Client (host A) **MUST** set the TCP flags (AE,CWR,ECE) = (1,1,1) in the initial SYN segment.

If a TCP Server (host 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 four combinations in the top block of Table 2, it confirms that it supports AccECN. The TCP Server **MUST NOT** set one of these four combinations 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. The TCP Client **MUST NOT** enter AccECN mode (or any feedback mode) before it has received the first SYN/ACK.

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 procedures for retransmission of SYNs or SYN/ACKs are given in Section 3.1.4.

It is **RECOMMENDED** that the AccECN protocol be implemented alongside Selective Acknowledgement (SACK) [RFC2018]. If SACK is implemented with AccECN, Duplicate Selective Acknowledgement (D-SACK) [RFC2883] **MUST** also be implemented.

3.1.2. Backward Compatibility

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

AccECN: Supports more Accurate ECN feedback (the present specification).

Nonce: Supports ECN-nonce feedback [RFC3540].

ECN: Supports 'Classic' ECN feedback [RFC3168].

No ECN: Not ECN-capable. Implicit congestion notification using packet drop.

Host A	Host B	SYN A->B AE CWR ECE	SYN/ACK B->A AE CWR ECE	Feedback Mode of Host A
AccECN	AccECN	1 1 1	0 1 0	AccECN (Not-ECT 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

Host A	Host B	SYN A->B AE CWR ECE	SYN/ACK B->A AE CWR ECE	Feedback Mode of Host A
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, with each block 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, as 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 the TCP Client has set itself into Classic ECN feedback mode, it **MUST** comply with [RFC3168].

   An AccECN implementation has no need to recognize or support the Server response labelled 'Nonce' or ECN-nonce feedback more generally [RFC3540], as RFC 3540 has been reclassified as Historic [RFC8311]. AccECN is compatible with alternative ECN feedback integrity approaches to the nonce (see Section 5.3). The SYN/ACK labelled 'Nonce' with (AE,CWR,ECE) = (1,0,1) is reserved for future use. A TCP Client (A) that receives such a SYN/ACK follows the procedure for forward compatibility given in Section 3.1.3.
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, as 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 Not 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 TCP-ECN 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,CWR,ECE) = (1,1,1), 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,ECE) set to any combination other than (0,0,0), (0,1,1), or (1,1,1) and it does not have logic specific to such a combination, the Server **MUST** negotiate the use of AccECN as if the three flags had been set to (1,1,1). However, an AccECN Client implementation **MUST NOT** send a SYN with any combination other than the three listed.

If a TCP Client sent a SYN requesting AccECN feedback with (AE,CWR,ECE) = (1,1,1) and then receives a SYN/ACK with the currently reserved combination (AE,CWR,ECE) = (1,0,1) but it does not have logic specific to such a combination, the Client **MUST** enable AccECN mode as if the SYN/ACK confirmed that the Server supported AccECN and as if it fed back that the IP ECN field on the SYN had arrived unchanged. However, an AccECN Server implementation **MUST NOT** send a SYN/ACK with this combination (AE,CWR,ECE) = (1,0,1).

> 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.

All of these requirements ensure that future uses of all the Reserved combinations on a SYN or SYN/ACK (see Table 2) can rely on consistent behaviour from the installed base of AccECN implementations. See Appendix B.3 for related discussion.

3.1.4. Multiple SYNs or SYN/ACKs

3.1.4.1. Retransmitted SYNs

If the sender of an AccECN SYN (the TCP Client) times out before receiving the SYN/ACK, it **SHOULD** attempt to negotiate the use of AccECN at least one more time by continuing to set all three TCP ECN flags (AE,CWR,ECE) = (1,1,1) on the first retransmitted SYN (using the usual

retransmission timeouts). If this first retransmission also fails to be acknowledged, in deployment scenarios where AccECN path traversal might be problematic, the TCP Client **SHOULD** send subsequent retransmissions of the SYN with the three TCP-ECN flags cleared (AE,CWR,ECE) = (0,0,0). Such a retransmitted SYN **MUST** use the same initial sequence number (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, recent measurements [Mandalari18] imply that a drop is less likely to be due to middlebox interference than other intermittent causes of loss, e.g., congestion, wireless transmission loss, 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)).

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 coordinate fall-back between different modules of the stack. For instance, if taking part in an [RFC8311] experiment that allows ECT on a SYN, it would be advisable to have a fall-back strategy that tries use of AccECN without setting ETC on SYN.

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.

All fall-back strategies will need to follow all the normative 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, including the possibility that they arrive out of order. As examples, the following non-normative bullets call out those rules from Section 3.1.5 that apply to the above fall-back strategies:

- Once the TCP Client has sent SYNs with (AE,CWR,ECE) = (1,1,1) and with (AE,CWR,ECE) = (0,0,0), it might eventually receive a SYN/ACK from the Server in response to one, the other, or both, and possibly reordered.
- Such a TCP Client enters the feedback mode appropriate to the first SYN/ACK it receives according to Table 2, and it does not switch to a different mode, whatever other SYN/ACKs it might receive or send.
- If a TCP Client has entered AccECN mode but then subsequently sends a SYN or receives a SYN/ACK with (AE,CWR,ECE) = (0,0,0), it is still allowed to set ECT on packets for the rest of the connection. Note that this rule is different than that of a Server in an equivalent position (Section 3.1.5 explains).

- Having entered AccECN mode, in general a TCP Client commits to respond to any incoming congestion feedback, whether or not it sets ECT on outgoing packets (for rationale and some exceptions see Section 3.2.2.3, Section 3.2.2.4).
- Having entered AccECN mode, a TCP Client commits to using AccECN to feed back the IP ECN field in incoming packets for the rest of the connection, as specified in Section 3.2, even if it is not itself setting ECT on outgoing packets.

3.1.4.2. Retransmitted SYN/ACKs

A TCP Server might send multiple SYN/ACKs indicating different feedback modes. For instance, when falling back to sending a SYN/ACK with (AE,CWR,ECE) = (0,0,0) after previous AccECN SYN/ACKs have timed out (Section 3.2.3.2.2); or to acknowledge different retransmissions of the SYN (Section 3.1.4.1).

All fall-back strategies will need to follow all the normative rules in Section 3.1.5, which concern behaviour when SYNs or SYN/ACKs negotiating different types of feedback are sent within the same connection, including the possibility that they arrive out of order. As examples, the following non-normative bullets call out those rules from Section 3.1.5 that apply to the above fall-back strategies:

- An AccECN-capable TCP Server enters the feedback mode appropriate to the first SYN it receives using Table 2, and it does not switch to a different mode, whatever other SYNs it might receive and whatever SYN/ACKs it might send.
- If a TCP Server in AccECN mode receives a SYN with (AE,CWR,ECE) = (0,0,0), it preferably acknowledges it first using an AccECN SYN/ACK, but it can retry using a SYN/ACK with (AE,CWR,ECE) = (0,0,0).
- If a TCP Server in AccECN mode sends multiple AccECN SYN/ACKs, it uses the TCP-ECN flags in each SYN/ACK to feed back the IP ECN field on the latest SYN to have arrived.
- If a TCP Server enters AccECN mode and then subsequently sends a SYN/ACK or receives a SYN with (AE,CWR,ECE) = (0,0,0), it is prohibited from setting ECT on any packet for the rest of the connection.
- Having entered AccECN mode, in general a TCP Server commits to respond to any incoming congestion feedback, whether or not it sets ECT on outgoing packets (for rationale and some exceptions see Sections 3.2.2.3, 3.2.2.4).
- Having entered AccECN mode, a TCP Server commits to using AccECN to feed back the IP ECN field in incoming packets for the rest of the connection, as specified in Section 3.2, even if it is not itself setting ECT on outgoing packets.

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.

An implementation that supports AccECN has the rights and obligations concerning the use of ECN defined below, which update those in Section 6.1.1 of [RFC3168]. This section uses the following definitions:

'During the handshake': The connection states prior to synchronization.

'Valid SYN': A SYN that has the same port numbers and the same ISN as the SYN that first caused the Server to open the connection. An 'Acceptable' packet is defined in Section 1.3.

Handling SYNs or SYN/ACKs of multiple types (e.g., fall-back):

- Any implementation that supports AccECN:
  - **MUST NOT** switch into a different feedback mode than the one it first entered according to Table 2, no matter whether it subsequently receives valid SYNs or Acceptable SYN/ACKs of different types.
  - **SHOULD** ignore the TCP-ECN flags in SYNs or SYN/ACKs that are received after the implementation reaches the ESTABLISHED state, in line with the general TCP approach [RFC9293].

    Reason: Reaching ESTABLISHED state implies that at least one SYN and one SYN/ACK have successfully been delivered. And all the rules for handshake fall-back are designed to work based on those packets that successfully traverse the path, whatever other handshake packets are lost or delayed.
  - **MUST NOT** send a 'Classic' ECN-setup SYN [RFC3168] with (AE,CWR,ECE) = (0,1,1) and a SYN with (AE,CWR,ECE) = (1,1,1) requesting AccECN feedback within the same connection;
  - **MUST NOT** send a 'Classic' ECN-setup SYN/ACK [RFC3168] with (AE,CWR,ECE) = (0,0,1) and a SYN/ACK agreeing to use AccECN feedback within the same connection;
  - **MUST** reset the connection with a RST packet, if it receives a 'Classic' ECN-setup SYN with (AE,CWR,ECE) = (0,1,1) and a SYN requesting AccECN feedback during the same handshake;
  - **MUST** reset the connection with a RST packet, if it receives 'Classic' ECN-setup SYN/ACK with (AE,CWR,ECE) = (0,0,1) and a SYN/ACK agreeing to use AccECN feedback during the same handshake;

  The last four 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 ECN feedback modes without knowing it.
- A host in AccECN mode that is feeding back the IP ECN field on a SYN or SYN/ACK:
  - **MUST** feed back the IP ECN field on the latest valid SYN or acceptable SYN/ACK to arrive.
- A TCP Server already in AccECN mode:
  - **SHOULD** acknowledge a valid SYN arriving with (AE,CWR,ECE) = (0,0,0) by emitting an AccECN SYN/ACK (with the appropriate combination of TCP-ECN flags to feed back the IP ECN field of this latest SYN).
  - **MAY** acknowledge a valid SYN arriving with (AE,CWR,ECE) = (0,0,0) by sending a SYN/ACK with (AE,CWR,ECE) = (0,0,0).

Rationale: When a SYN arrives with (AE,CWR,ECE) = (0,0,0) at a TCP Server that is already in AccECN mode, it implies that the TCP Client had probably not received the previous AccECN SYN/ACK emitted by the TCP Server. Therefore, the first bullet recommends attempting at least one more AccECN SYN/ACK. Nonetheless, the second bullet recognizes that the Server might eventually need to fall back to a non-ECN SYN/ACK. In either case, the TCP Server remains in AccECN feedback mode (according to the earlier requirement not to switch modes).

- An AccECN-capable TCP Server already in Not ECN mode:
  - **SHOULD** respond to any subsequent valid SYN using a SYN/ACK with (AE,CWR,ECE) = (0,0,0), even if the SYN is offering to negotiate Classic ECN or AccECN feedback mode.

    Rationale: There would be no point in the Server offering any type of ECN feedback, because the Client will not be using ECN. However, there is no interoperability reason to make this rule mandatory.

If for any reason a host is not willing to provide ECN feedback on a particular TCP connection, it **SHOULD** clear the AE, CWR, and ECE flags in all SYN and/or SYN/ACK packets that it sends.

Sending ECT:

- Any implementation that supports AccECN:
  - **MUST NOT** set ECT if it is in Not ECN feedback mode.

  A Data Sender in AccECN mode:
  - **SHOULD** 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.
  - **MAY** not set ECT on any packet (for instance if it has reason to believe such a packet would be blocked).

  A TCP Server in AccECN mode:
  - **MUST NOT** set ECT on any packet for the rest of the connection, if it has received or sent at least one valid SYN or Acceptable SYN/ACK with (AE,CWR,ECE) = (0,0,0) during the handshake.

    This rule solely applies to a Server because, when a Server enters AccECN mode, it doesn't know for sure whether the Client will end up in AccECN mode. But when a Client enters AccECN mode, it can be certain that the Server is already in AccECN feedback mode.

Congestion response:

- A host in AccECN mode:
  - is obliged to respond appropriately to AccECN feedback that indicates there were ECN marks on packets it had previously sent, where 'appropriately' is defined in Section 6.1 of [RFC3168] and updated by Sections 2.1 and 4.1 of [RFC8311].

- is still obliged to respond appropriately to congestion feedback, even when it is solely sending non-ECN-capable packets (for rationale, some examples and some exceptions see Sections 3.2.2.3 and 3.2.2.4).
- is still obliged to respond appropriately to congestion feedback, even if it has sent or received a SYN or SYN/ACK packet with (AE,CWR,ECE) = (0,0,0) during the handshake.
- **MUST NOT** set CWR to indicate that it has received and responded to indications of congestion.

  For the avoidance of doubt, this is unlike an RFC 3168 data sender and this does not preclude the Data Sender from setting the bits of the ACE counter field, which includes an overloaded use of the same bit.

Receiving ECT:

- 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.

    For the avoidance of doubt, this requirement stands even if the AccECN host has also sent or received a SYN or SYN/ACK with (AE,CWR,ECE) = (0,0,0). Reason: Such a SYN or SYN/ACK implies some form of packet mangling might be present. Even if the remote peer is not setting ECT, it could still be set erroneously by packet mangling at the IP layer (see Section 3.2.2.3). In such cases, the Data Sender is best placed to decide whether ECN markings are valid, but it can only do that if the Data Receiver mechanistically feeds back any ECN markings. This approach will not lead to TCP Options being generated unnecessarily if the recommended simple scheme in Section 3.2.3.3 is used, because no byte counters will change if no packets are set to ECT.
  - **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, because 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.e1b:

- 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 and retransmissions but excluding CE on SYN packets (SYN=1; ACK=0).
- A Data Receiver that supports sending of AccECN TCP Options **MUST** increment the r.ceb, r.e0b, or r.e1b byte counters by the number of TCP payload octets in Acceptable packets marked with the CE, ECT(0), and ECT(1) codepoint in their IP ECN field, including any payload octets on control packets and retransmissions, 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.e1b, 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 Data Receiver feeds back the value of any counter, it **MUST** report the most recent value, no matter whether it is in a pure ACK, or an ACK piggybacked on a packet used by the other half-connection, whether a new payload data or a retransmission. Therefore, the feedback piggybacked 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.e1b = 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.e1b = 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.

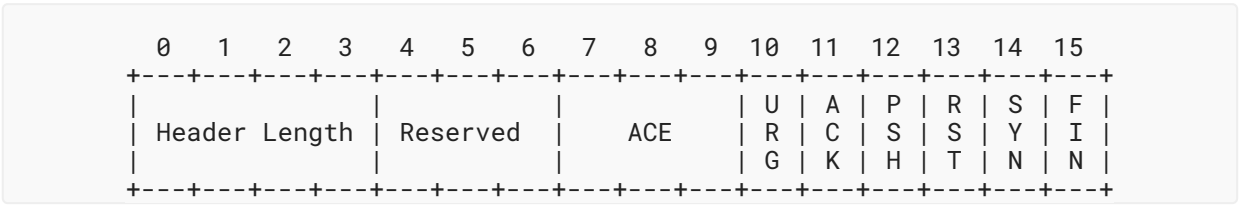

Figure 3: Definition of the ACE Field Within Bytes 13 and 14 of the TCP Header (When AccECN Has Been Negotiated and SYN=0).

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 3 least significant bits of its r.cep counter into the ACE field that it feeds back to the Data Sender. The least significant bit

is at bit offset 9 in Figure 3. A host **MUST NOT** interpret the three 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.

The TCP Client **MUST** also use the handshake encoding for the pure ACK of any retransmitted SYN/ACK that confirms that the TCP Server supports AccECN. If the final ACK of the handshake does not arrive before its retransmission timer expires, the TCP Server is follow the procedure given in Section 3.1.4.2.

IP ECN Codepoint on SYN/ACK	ACE on Pure ACK of SYN/ACK	r.cep of TCP 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, it **MUST** infer the meaning of each possible value of the ACE field from Table 4 instead of treating the ACE field as a counter. As a result, an AccECN Server **MUST** set s.cep to the respective value, also shown in Table 4.

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 the ACK of the SYN/ACK was sent and it has transitioned out of SYN-RCVD state.

ACE on ACK of SYN/ACK	IP ECN Codepoint on SYN/ACK Inferred by Server	s.cep of TCP Server in AccECN Mode
0b000	{Notes 1, 3}	Disable s.cep
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 and in SYN-RCVD state, and if it receives a value of zero on a pure ACK with SYN=0 and no SACK blocks, for the rest of the connection the Server **MUST NOT** set ECT on outgoing packets and **MUST NOT** respond to AccECN feedback. Nonetheless, as a Data Receiver, it **MUST NOT** disable AccECN feedback.

Any of the circumstances below could cause a value of zero but, whatever the cause, the actions above would be the appropriate response:

- The TCP Client has somehow entered No ECN feedback mode (most likely if the Server received a SYN or sent a SYN/ACK with (AE,CWR,ECE) = (0,0,0) after entering AccECN mode, but possible even if it didn't).
- The TCP Client genuinely might be in AccECN mode, but its count of received CE marks might have caused the ACE field to wrap to zero. This is highly unlikely, but not impossible because the Server might have already sent multiple packets while still in SYN-RCVD state, e.g., using TFO (see Section 5.2), and some might have been CE-marked. Then ACE on the first ACK seen by the Server might be zero, due to previous ACKs experiencing an unfortunate pattern of loss or delay.

- There is some form of non-compliance at the TCP Client or 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 Server that uses a stateless handshake and implements AccECN receives either of these two values in the ACK, its action is implementation-dependent and outside the scope of this document. 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. For example, 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 (defined in Appendix A.1) 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).

- 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 once at most).
- 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:

- 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 the top block of 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).

- 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.
- 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.
- 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

- TCP Client side:

  The value of the TCP-ECN flags on the SYN/ACK indicates the value of the IP ECN field when the SYN arrived at the Server. The TCP 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 TCP 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).
- TCP Server side:

  The value of the ACE field on the last ACK of the three-way handshake indicates the value of the IP ECN field when the SYN/ACK arrived at the TCP 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 TCP Client or Server was non-ECN-capable or ECN-capable because, in either case, an unsafe transition could imply that non-ECN-capable packets later in the connection might get mangled.

Once a Data Sender has entered AccECN mode it is advised to check whether it is receiving continuous feedback of CE. 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.

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 same is true for other forms of mangling (or resumption of expected marking) during later stages of a connection.

As always, once a host has entered AccECN mode, it follows the general mandatory requirements (Section 3.1.5) to remain in the same feedback mode and to continue feeding back any ECN markings on arriving packets using AccECN feedback. This follows the general approach where an AccECN Data Receiver mechanistically reflects whatever it receives (Section 2.5).

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.

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 [Mandalari18]. In this case, 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.

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

- the Not-ECT codepoint changes.

- either ECT codepoint transitions to Not-ECT.
- the CE codepoint changes.

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

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.

This section does not concern the case where the ACE field is zero when the handshake encoding has been used on the ACK of the SYN/ACK under the carefully worded conditions in Section 3.2.2.1.

If AccECN has been successfully negotiated, the Data Sender **MAY** check the value of the ACE counter in the first feedback packet (with or without data) that arrives after the three-way handshake. If the value of this ACE field is found to be 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 any or all of the following:

- the remote peer has somehow entered Not ECN feedback mode.
- a broken remote TCP implementation.
- potential mangling of the ECN fields in the TCP headers (although unlikely given they clearly survived during the handshake).

This advice is not stated normatively (in capitals), because the best strategy might depend on the likelihood to experience these scenarios, which can only be known at the time of deployment.

Note that a host in AccECN mode **MUST** continue to provide Accurate ECN feedback to its peer, even if it is no longer sending ECT itself over the other half-connection.

If reordering occurs, the first feedback 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 reordering 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 **MAY** 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 the 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. Packet Receiver Safety Procedures

The following rules define when the receiver of a packet 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 due to large receive offload (LRO) or generic receive offload (GRO) as further described below.

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 receiver of a packet **MUST** emit an ACK if 'n' CE marks have arrived since the previous ACK. If there is unacknowledged data at the receiver, 'n' **SHOULD** be 2. If there is no unacknowledged data at the receiver, '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 an 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 sequence of '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 Active Queue Management (AQM) packet schedulers 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.

In a scenario where both endpoints support AccECN, if host B has chosen to use ECN-capable pure ACKs (as allowed in [RFC8311] experiments) and enough of these ACKs become CE marked, then the 'Increment-Triggered ACKs' rule ensures that its peer (host A) gives B sufficient feedback about this congestion on the ACKs from B to A. Normally, for instance in a unidirectional data scenario from host A to B, the Data Sender (A) can piggyback that feedback on its data. But if A stops sending data, the second part of the 'Increment-Triggered ACKs' rule requires A to emit a pure ACK for at least every third CE-marked incoming ACK over the subsequent round trip.

Although TCP normally only ACKs data segments, in this case the increment-triggered ACK rule makes it mandatory for A to emit ACKs of ACKs. This is justifiable because the ACKs in this case are ECN-capable and so, even though the ACKs of these ACKs do not acknowledge new data, they feed back new congestion state (useful in case B starts sending). The minimum of 3 for 'n' in this case ensures that, even if A also uses ECN-capable pure ACKs, and even if there is pathological congestion in both directions, any resulting ping-pong of ACKs will be rapidly damped.

In the above bidirectional scenario, incoming ACKs of ACKs could be mistaken for duplicate ACKs. But ACKs of ACKs can be distinguished from duplicate ACKs because they do not contain any SACK blocks even when SACK has been negotiated. It is outside the scope of this AccECN specification to normatively specify this additional test for DupACKs, because ACKs of ACKs can only arise if the original ACKs are ECN-capable. Instead, any specification that allows ECN-capable pure ACKs **MUST** make sending ACKs of ACKs conditional on measures to distinguish ACKs of ACKs from DupACKs (see for example [ECN++]). All that is necessary here is to require that these ACKs of ACKs **MUST NOT** contain any SACK blocks (which would normally not happen anyway).

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 implementation **MAY** use an alternative algorithm as long as it satisfies the requirements in this subsection.

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** consider the ACE counter to be correct (and its count of control packets to be incomplete), unless it can be sure that it is counting all control packets correctly.

3.2.3. The AccECN Option

Two alternative AccECN Options are defined as shown in Figure 4. The initial 'E' of each field name stands for 'Echo'.

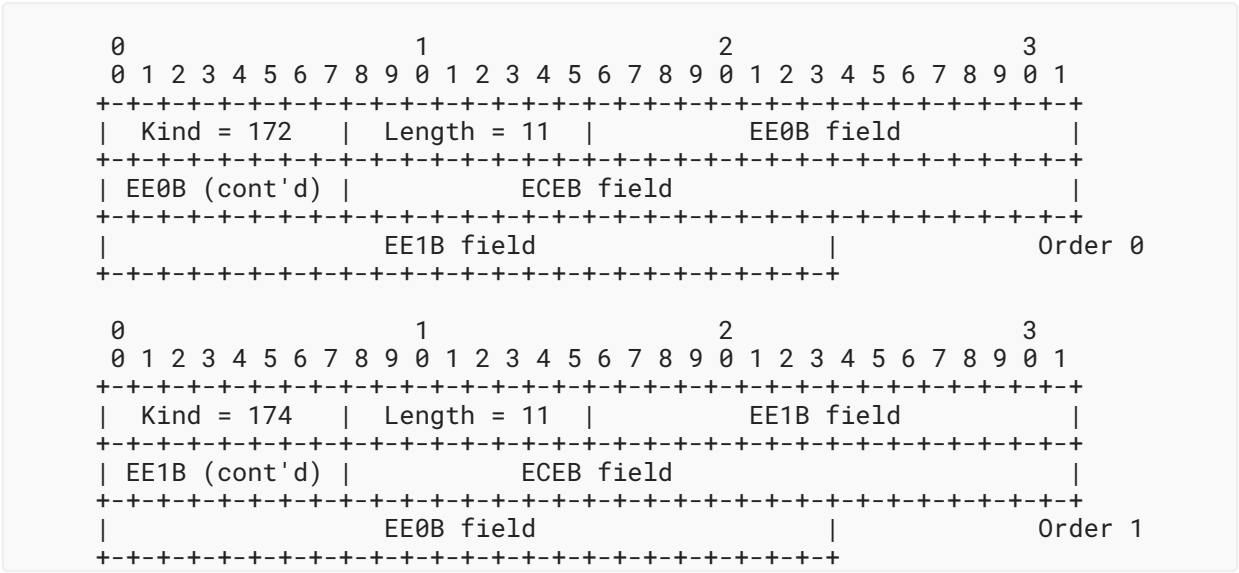

Figure 4: The Two Alternative AccECN TCP Options

Figure 4 shows two option field orders; order 0 and order 1. They both consist of three 24-bit fields. Order 0 provides the 24 least significant bits of the r.e0b, r.ceb, and r.e1b 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. In either case, the bytes within the fields are in network byte order (big-endian).

The choice to use three bytes (24 bits) fields in the options was made to strike a balance between TCP Option space usage, and the required fidelity of the counters to accommodate typical scenarios such as hardware TCP Segmentation Offloading (TSO), and periods during which no option may be transmitted (e.g., SACK loss recovery). Providing only 2 bytes (16 bits) for these counters could easily roll over within a single TSO transmission or large/generic receive offload (LRO/GRO) event. Having two distinct orderings further allows the transmission of the most pertinent changes in an abbreviated option (see below).

When a Data Receiver sends an AccECN Option, it **MUST** set the Kind field to 172 if using Order 0, or to 174 if using Order 1. These two new TCP Option Kinds are registered in Section 7 and are called AccECN0 and AccECN1, respectively.

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	Order 0	Order 1
11	EE0B, ECEB, EE1B	EE1B, ECEB, EE0B
8	EE0B, ECEB	EE1B, ECEB
5	EE0B	EE1B
2	(empty)	(empty)

Table 5: Fields included in AccECN TCP Options of each length and order

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 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.

AccECN Options have to be optional to implement, because both sender and receiver have to be able to cope without options anyway -- in cases where they do not traverse a network path. It is **RECOMMENDED** to implement both sending and receiving of AccECN Options. Support for AccECN Options 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 AccECN Options 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 logic to understand received AccECN Options, it is **RECOMMENDED** that they implement logic to send AccECN Options. Otherwise, those remote peers that implement the receiving logic will still be excluded from congestion feedback that is robust against the increasingly aggressive ACK filtering in the Internet. The logic to send AccECN Options is the simpler to implement of the two sides.

If a Data Receiver intends to send an AccECN Option at any time during the rest of the connection, it is **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 an 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 once at most).

Appendix A.1 gives an example algorithm for the Data Receiver to encode its byte counters into an 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 an 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 an 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 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 three-way handshake. 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 an AccECN Option in any of the above three cases because of insufficient option space or because 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 the TCP Server has not received an ACK to acknowledge its SYN/ACK after the normal TCP timeout or if it receives a second SYN with a request for AccECN support, then either the SYN/ACK might just have been lost, e.g., due to congestion, or a middlebox might be blocking AccECN Options. To expedite connection setup in deployment scenarios where AccECN path traversal might be problematic, the TCP Server **SHOULD** retransmit the SYN/ACK, but with no AccECN

Option. If this retransmission times out, to expedite connection setup, the TCP Server **SHOULD** retransmit the SYN/ACK with (AE,CWR,ECE) = (0,0,0) and no AccECN Option, but it remains in AccECN feedback mode (per Section 3.1.5).

> Note that a retransmitted AccECN SYN/ACK will not necessarily have the same TCP-ECN flags as the original SYN/ACK, because it feeds back the IP ECN field of the latest SYN to have arrived (by the rule in Section 3.1.5).

The above fall-back approach limits any interference by middleboxes that might drop packets with unknown options, even though it is more likely that SYN/ACK loss is due to congestion. The TCP Server **MAY** try to send another packet with an 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 AccECN Options for this half-connection.

Implementers **MAY** use other fall-back strategies if they are found to be more effective (e.g., retrying an 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.

Further it might make sense to also remove any other new or experimental fields or options on the SYN/ACK, although the required behaviour will depend on the specification of the other option(s) and on any attempt to coordinate fall-back between different modules of the stack.

If the TCP Client detects that the first data segment it sent with an AccECN Option was lost, in deployment scenarios where AccECN path traversal might be problematic, 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 an AccECN Option before fall-back, and/or caching whether AccECN Options are blocked for subsequent connections. [RFC9040] further discusses caching of TCP parameters and status information.

If a middlebox is dropping packets with options it does not recognize, a host that is sending little or no data but mostly pure ACKs will not inherently detect such losses. Such a host **MAY** detect loss of ACKs carrying the AccECN Option by detecting whether the acknowledged data always reappears as a retransmission. In such cases, the host **SHOULD** disable the sending of the AccECN Option for this half-connection.

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

Either host **MAY** include AccECN Options in one or more subsequent segments to retest whether AccECN Options can traverse the path.

Similarly, an AccECN endpoint **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 is 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 AccECN Options itself (unless it has established that sending AccECN Options 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 AccECN Options are no longer blocked on the respective path, and the AccECN endpoint **MAY** switch out of the mode that assumes AccECN Options are 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 an AccECN Option (if one exists) ought to be non-zero. If AccECN has been negotiated:

- 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 AccECN Options for this half-connection.
- the TCP Client **MAY** check that 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 AccECN Options for this half-connection.

While a host is in the mode that ignores AccECN Options, 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 an 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 the 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 AccECN Options are available, they ought to provide more unambiguous feedback. However, they supplement but do 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 an AccECN Option.

If an 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 an AccECN Option but not the ACE field. However, the Data Sender has to assume that the integrity of AccECN Options is sound, based on the above test of the well-known initial values and optionally other integrity tests (Section 5.3).

If either endpoint 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 AccECN TCP Options to provide feedback, the rules below determine when to include 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 implementations **SHOULD** generally prioritize over other TCP Options when there is insufficient space for all the options in use.

  If SACK has been negotiated [RFC2018], 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: When TCP Option space is limited, an AccECN TCP Option **MAY** be truncated to omit one or two fields from the end of the option, as indicated by the permitted variants listed in Table 5, provided that the counter(s) that have changed since the previous AccECN TCP Option are not omitted.

If there is insufficient space to include an AccECN TCP Option containing the counter(s) that have changed since the previous AccECN TCP Option, then the entire AccECN TCP Option **MUST** be omitted. (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 that acknowledges new data. 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 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 AccECN Options are not available.

If a host has determined that segments with AccECN Options 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

Given AccECN alters the TCP protocol on the wire, this section specifies new requirements on certain networking equipment that forwards TCP and inspects TCP header information.

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 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 AccECN TCP Options 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 from being extended in the future (such 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 Sections 3.2.3 or 3.2.1. However, to comply with the present AccECN specification, a middlebox **MUST NOT** change the ACE field; or those fields of an 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

Section 5.2.1 of RFC 3449 [BCP69] gives best current practice on filtering (aka thinning or coalescing) of pure TCP ACKs. It advises that filtering ACKs carrying ECN feedback ought to preserve the correct operation of ECN feedback. As the present specification updates the operation of ECN feedback, this section discusses how an ACK filter might preserve correct operation of AccECN feedback as well.

The problem divides into two parts: determining if an ACK is part of a connection that is using AccECN and then preserving the correct operation of AccECN feedback:

- 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 [ECN++]). This heuristic is simple and stateless. However, it might omit some AccECN ACKs because AccECN can be used without ECN++. Even if ECN++ is used, pure ACKs do not necessarily have to be marked as ECN-capable -- 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.
- 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. In the worst case, in feedback about a run of received packets that were all ECN-marked, the ACE field will wrap every 8 acknowledged packets. ACE field wrap might be of less concern if packets also carry AccECN TCP Options. However, note that logic to read an AccECN TCP Option is optional to implement (albeit recommended -- see Section 3.2.3). So one end writing an AccECN TCP Option into a packet does not necessarily imply that the other end will read it.

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, 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.

3.3.4. Requirements for TCP Segmentation Offload and Large Receive 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').

Offloading can happen in the transmit path, usually referred to as TCP Segmentation Offload (TSO), and the receive path where it is called Large Receive Offload (LRO).

In the transmit direction, with AccECN, all segments created from the same super-segment should retain the same ACE field, which should make TSO straighforward.

However, with TSO hardware that supports [RFC3168], the CWR bit is usually masked out on the middle and last segments. If applied to an AccECN segment, this would change the ACE field, and would be interpreted as having received numerous CE marks in the receive direction. Therefore, currently available TSO hardware with [RFC3168] support may need some minor driver changes, to adjust the bitmask for the first, middle, and last segments processed with TSO.

Initially, when Classic ECN [RFC3168] and Accurate ECN flows coexist on the same offloading engine, the host software may need to work around incompatibilities (e.g., when only global configurable TSO TCP Flag bitmasks are available), otherwise this would cause some issues.

One way around this could be to only negotiate for Accurate ECN, but not offer a fall back to Classic ECN [RFC3168]. Another way could be to allow TSO only as long as the CWR flag in the TCP header is not set -- at the cost of more processing overhead while the ACE field has this bit set.

For LRO in the receive direction, a different issue may get exposed with Classic ECN [RFC3168] supporting hardware.

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. 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 on the long marking sequences from step marking for efficiency. 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

This section clarifies which parts of RFC 3168 are updated and maps them to the relevant updated sections of the present AccECN specification.

- The whole of Section 6.1.1 of [RFC3168] is updated by Section 3.1 of the present specification.
- In Section 6.1.2 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 Section 6.1.2 of [RFC3168] still stand.

  It will be noted that [RFC8311] already updates a number of the requirements in Section 6.1.2 of [RFC3168]. Section 6.1.2 of RFC 3168 extended standard TCP congestion control [RFC5681] to cover ECN marking as well as packet drop. Whereas, [RFC8311] enables experimentation with alternative responses to ECN marking, if specified for instance by an Experimental RFC produced by the IETF Stream. [RFC8311] also strengthened the statement that "ECT(0) **SHOULD** be used" to a "**MUST**" (see [RFC8311] for the details).
- The whole of Section 6.1.3 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.
- The following text within Section 6.1.5 of [RFC3168]:

  > 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 acceptability tests equivalent in strength to those in both [RFC9293] and [RFC5961].
- 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 [RFC3168], 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 [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), the Server can infer the first two stages of the handshake:

- the TCP Client has to have requested AccECN support on the SYN;
- then, even though the Server kept no state, it 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 TCP Client did not request support for AccECN; 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 TCP 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, timestamp, window scaling, SACK, and TCP-AO. It is also compatible with Multipath TCP (MPTCP [RFC8684]) and the experimental TCP Option TCP Fast Open (TFO [RFC7413]). 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.

Because option space is limited, Section 3.2.3.3 provides guidance on how important it is to send an AccECN Option relative to other options and specifies 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 TCP 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.

AccECN feedback is compatible with the ECN++ experiment [ECN++], which allows TCP control packets and retransmissions to be ECN-capable ([RFC3168] was updated by [RFC8311] to permit such experiments). AccECN is likely to inherently support any experiment with ECN-capable packets, because it feeds back the contents of the ECN field mechanistically, without judging whether or not a packet ought to use the ECN capability (Section 2.5). This specification does not discuss implementing AccECN alongside [RFC5562], which was an earlier experimental protocol with narrower scope than ECN++ and a 5-way handshake.

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:

- 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 paragraph 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.
- 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.
- 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., Network Address Translation (NAT) and Network Address Port Translation (NAPT), resegmentation, or shifting the sequence space.

6. Summary: 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 an 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 feedback part reuses the three flags already assigned to ECN in the TCP header. The supplementary feedback part adds an additional TCP Option consuming up to 11 bytes. However, no TCP Option space 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 AccECN TCP Options 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 allow the Data Sender to immediately recover the number of the ECN markings that it missed. 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.

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 AccECN Options; 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 and 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 AccECN Options are 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.

If AccECN Options are 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 an AccECN Option. It can detect these conditions during TCP's three-way handshake so that it can fall back to operation without ECN and/or operation without AccECN Options.

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 the 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 and 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 the TCP header flag at bit offset 7 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 is now defined as the following in the "TCP Header Flags" registry in the "Transmission Control Protocol (TCP) Parameters" registry group:

Bit	Name	Reference	Assignment Notes
7	AE (Accurate ECN)	RFC 9768	Previously used as NS (Nonce Sum) by [RFC3540], which is now Historic [RFC8311]

Table 6: TCP Header Flag Reassignment

This document also defines two new TCP Options for AccECN from the TCP Option space. These values are defined as the following in the "TCP Option Kind Numbers" registry in the "Transmission Control Protocol (TCP) Parameters" registry group:

Kind	Length	Meaning	Reference
172	N	Accurate ECN Order 0 (AccECN0)	RFC 9768
174	N	Accurate ECN Order 1 (AccECN1)	RFC 9768

Table 7: New TCP Option Assignments

Early experimental implementations of the two AccECN Options used experimental option 254 per [RFC6994] with the 16-bit magic numbers 0xACC0 and 0xACC1, respectively, for Order 0 and 1, as allocated in the IANA "TCP/UDP Experimental Option Experiment Identifiers (TCP/UDP ExIDs)" registry. Even earlier experimental implementations used the single magic number 0xACCE (16 bits). Uses of these experimental options **SHOULD** migrate to use the new option kinds (172 and 174).

8. Security and Privacy Considerations

If ever the supplementary feedback part of AccECN that is based on one of the new AccECN TCP Options is unusable (due for example to middlebox interference), the essential feedback 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 AccECN Options are 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 AccECN Options are 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)). However, it is really an overt channel (not hidden) and it is no different than 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 AccECN Options pretending that they had been stripped by a middlebox. Currently, there is no known way 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.

A generic privacy concern of any new protocol is that for a while it will be used by a small population of hosts, and thus those hosts could be more easily identified. However, it is expected that AccECN will become available in operating systems over time and that it will eventually be turned on by default. Thus, an individual identification of a particular user is less of a concern than the fingerprinting of specific versions of operation systems. However, the latter can be done using different means independent of Accurate ECN.

As Accurate ECN exposes more bits in the TCP header that could be tampered with without interfering with the transport excessively, it may allow an additional way to identify specific data streams across a virtual private network (VPN) to an attacker that has access to the datastream before and after the VPN tunnel endpoints. This may be achieved by injecting or modifying the ACE field in specific patterns that can be recognized.

Overall, Accurate ECN does not change the risk profile on privacy to a user dramatically beyond what is already possible using classic ECN. However, in order to prevent such attacks and means of easier identification of flows, it is advisable for privacy-conscious users behind VPNs to not enable the Accurate ECN, or Classic ECN for that matter.

9. References

9.1. Normative References

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

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

[RFC2883] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, DOI 10.17487/RFC2883, July 2000, <https://www.rfc-editor.org/info/rfc2883>.

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

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

9.2. Informative References

[BCP69] Best Current Practice 69, <https://www.rfc-editor.org/info/bcp69>.
At the time of writing, this BCP comprises the following:

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>.

[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-17, 21 April 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-generalized-ecn-17>.

[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, <http://www.it.uc3m.es/amandala/ecn++/ecn_commag_2018.html>.

[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>.

[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>.

[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>.

[RFC6582] Henderson, T., Floyd, S., Gurtov, A., and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 6582, DOI 10.17487/RFC6582, April 2012, <https://www.rfc-editor.org/info/rfc6582>.

[RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", RFC 6679, DOI 10.17487/RFC6679, August 2012, <https://www.rfc-editor.org/info/rfc6679>.

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

[RFC7141] Briscoe, B. and J. Manner, "Byte and Packet Congestion Notification", BCP 41, RFC 7141, DOI 10.17487/RFC7141, February 2014, <https://www.rfc-editor.org/info/rfc7141>.

[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>.

[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>.

[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>.

[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>.

[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>.

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

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

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

[RoCEv2] InfiniBand Trade Association, "InfiniBand Architecture Specification", <https://www.infinibandta.org/ibta-specification/>.

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 satisfy 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 an 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 an 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:

```
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 an AccECN Option, the value it writes into the ECEB field is

```
ECEB = r.ceb % 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 [RFC2018] to calculate newlyAckedB, the amount of new data that the ACK acknowledges in bytes (newlyAckedB can be zero but not negative). If newlyAckedB 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 [RFC7323] (if present) to work out newlyAckedT, 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 ((newlyAckedB > 0) || (newlyAckedT > 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 AccECN Options are 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 AccECN Options.

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 indicates 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 > newlyAckedPkt), which could occur if newlyAckedPkt 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 overly 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 newlyAckedPkt was well below 8.

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

The simple algorithm for 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 a full-sized packet 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 3.2.2.5.2 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 AccECN Options are 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 AccECN Options 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 replace dSafer.cep with d.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 :

- 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.

- 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.

- if d.cep=7, dSafer.cep=15, and d.ceb=10200, then s=1457 and sSafer=680.

  Therefore d.cep is safe enough, because the average data segment size is more likely to have been just less than one MSS, rather than below MSS/2.

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

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

If AccECN Options are not available, the Data Sender can only decode a CE marking from the ACE field in packets. Every time an ACK arrives, to convert the number of CE markings 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 ~= flightsize / packets_in_flight,
```

where flightsize is the variable that TCP already maintains for the number of bytes in flight and '~=' means 'approximately equal to'. To avoid floating point arithmetic, it could right-bit-shift by lg(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 mark retransmitted segments as Not-ECT. However, the converse is not true; necessary retransmissions will result in undercounting.

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 IPv4 ToS byte or the IPv6 Traffic Class 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.

Classic ECN 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 Classic 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 capability negotiation 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 ECN [RFC3168], 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 of the conflicting cases where a Server might have said it supported one approach in the flags and another approach in a new TCP Option. And AccECN would have had to deal with all of the additional possibilities where a middlebox might have mangled the ECN flags, or removed TCP Options. 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 third 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 eight possible codepoints that the three TCP header flags can indicate on the SYN/ACK, four already indicated earlier (or broken) versions of ECN support, one now being Historic. 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 mechanistic 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 IP 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 two codepoints on the SYN/ACK that had been reserved for future use by AccECN in earlier draft versions of this document.

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 three-way handshake, 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 defined in the RFC series. These could be used by TCP Servers in the future to indicate a variant of the AccECN protocol. In recent measurement studies in which the response of large numbers of Servers to an AccECN SYN has been tested, e.g., [Mandalari18], a very small number of SYN/ACKs arrive with the pattern tagged as 'Nonce', and a small but more significant number arrive with the pattern tagged as 'Broken'. The 'Nonce' pattern could be a sign that a few Servers have implemented the ECN-nonce [RFC3540], which has now been reclassified as Historic [RFC8311], or it could be the random result of some unknown middlebox behaviour. The greater prevalence of the 'Broken' pattern suggests that some instances still exist of the broken code that reflects the reserved flags on the SYN.

The requirement not to reject unexpected initial values of the ACE counter (in the main TCP header) in the last paragraph of Section 3.2.2.4 ensures that three unused codepoints on the ACK of the SYN/ACK, six unused values on the first SYN=0 data packet from the Client, and seven 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 three-way handshake, it would be too late for a negotiation between the endpoints to be completed. A similar requirement not to reject unexpected initial values in AccECN TCP Options (Section 3.2.3.2.4) is for the same purpose. If traversal of AccECN TCP Options 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 eight possible in the three TCP header flags used by AccECN are unused on the initial SYN (in the order (AE,CWR,ECE)): (0,0,1), (0,1,0), (1,0,0), (1,0,1), (1,1,0). Section 3.1.3 ensures that the installed base of AccECN Servers will all assume these are equivalent to AccECN negotiation with (1,1,1) 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 the future, perhaps for uses other than ECN alongside the AccECN scheme. All possible combinations of SYN/ACK could be used in response except either (0,0,0) or reflection of the same values sent on the SYN.

In order to extend AccECN or ECN in the future, other ways could be resorted to, 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.

Acknowledgements

We want to thank Koen De Schepper, Praveen Balasubramanian, Michael Welzl, Gorry Fairhurst, David Black, Spencer Dawkins, Michael Scharf, Michael Tüxen, Yuchung Cheng, Kenjiro Cho, Olivier Tilmans, Ilpo Järvinen, Neal Cardwell, Yoshifumi Nishida, Martin Duke, Jonathan Morton, Vidhi Goel, Alex Burr, Markku Kojo, Grenville Armitage and Wes Eddy 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.

The following contributed implementations of AccECN that validated and helped to improve this specification:

Linux: Mirja Kühlewind, Ilpo Järvinen, Neal Cardwell, and Chia-Yu Chang

FreeBSD: Richard Scheffenegger

Apple OSs: Vidhi Goel

Bob Briscoe was part-funded by Apple Inc, 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 Kühlewind 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.

Authors' Addresses

Bob Briscoe
Independent
United Kingdom
Email: ietf@bobbriscoe.net
URI: http://bobbriscoe.net/

Mirja Kühlewind
Ericsson
Germany
Email: ietf@kuehlewind.net

Richard Scheffenegger
NetApp
Vienna
Austria
Email: Richard.Scheffenegger@netapp.com