

Network Working Group
Internet-Draft
Intended status: Experimental
Expires: November 20, 2017

M. Bagnulo
UC3M
B. Briscoe
Simula Research Lab
May 19, 2017

ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control
Packets
draft-bagnulo-tcpm-generalized-ecn-04

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on November 20, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Motivation	3
1.2. Experiment Goals	4
1.3. Document Structure	5
2. Terminology	5
3. Specification	6
3.1. Network (e.g. Firewall) Behaviour	6
3.2. Endpoint Behaviour	6
3.2.1. SYN	8
3.2.2. SYN-ACK	11
3.2.3. Pure ACK	12
3.2.4. Window Probe	13
3.2.5. FIN	13
3.2.6. RST	14
3.2.7. Retransmissions	14
4. Rationale	15
4.1. The Reliability Argument	15
4.2. SYNs	16
4.2.1. Argument 1a: Unrecognized CE on the SYN	16
4.2.2. Argument 1b: Unrecognized ECT on the SYN	18
4.2.3. Argument 2: DoS Attacks	20
4.3. SYN-ACKs	20
4.4. Pure ACKs	22
4.4.1. Cwnd Response to CE-Marked Pure ACKs	23
4.4.2. ACK Rate Response to CE-Marked Pure ACKs	24
4.4.3. Summary: Enabling ECN on Pure ACKs	25
4.5. Window Probes	25
4.6. FINs	26
4.7. RSTs	26
4.8. Retransmitted Packets.	27
5. Interaction with popular variants or derivatives of TCP	28
5.1. SCTP	29
5.2. IW10	29
5.3. TFO	30
6. Security Considerations	30
7. IANA Considerations	30
8. Acknowledgments	30
9. References	31
9.1. Normative References	31
9.2. Informative References	31
Authors' Addresses	33

1. Introduction

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

This document defines an experimental modification to ECN [RFC3168] that enables ECN support on all the aforementioned types of TCP packet. [I-D.ietf-tsvwg-ecn-experimentation] is a standards track procedural device that relaxes standards track requirements in RFC 3168 that would otherwise preclude these experimental modifications.

The present document also considers the implications for common derivatives and variants of TCP, such as SCTP [RFC4960], if the experiment is successful. One particular variant of TCP adds accurate ECN feedback (AccECN [I-D.ietf-tcpm-accurate-ecn]), without which ECN support cannot be added to SYNs. Nonetheless, ECN support can be added to all the other types of TCP packet whether or not AccECN is also supported.

1.1. Motivation

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

Non-ECN control packets particularly harm performance in environments where the ECN marking level is high. For example, [judd-nsdi] shows that in a data centre (DC) environment where ECN is used (in conjunction with DCTCP), the probability of being able to establish a new connection using a non-ECN SYN packet drops to close to zero even when there are only 16 ongoing TCP flows transmitting at full speed. In this data centre context, the issue is that DCTCP's aggressive response to packet marking leads to a high marking probability for

ECN-capable packets, and in turn a high drop probability for non-ECN packets. Therefore non-ECN SYNs are dropped aggressively, rendering it nearly impossible to establish a new connection in the presence of even mild traffic load.

Finally, there are ongoing experimental efforts to promote the adoption of a slightly modified variant of DCTCP (and similar congestion controls) over the Internet to achieve low latency, low loss and scalable throughput (L4S) for all communications [I-D.briscoe-tsvwg-l4s-arch]. In such an approach, L4S packets identify themselves using an ECN codepoint. With L4S and potentially other similar cases, preventing TCP control packets from obtaining the benefits of ECN would not only expose them to the prevailing level of congestion loss, but it would also classify control packet into a different queue with different network treatment, which may also lead to reordering, further degrading TCP performance.

1.2. Experiment Goals

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

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

The data gathered through the experiments described in this document, particularly under the first 2 bullets above, will help in the design of the final mechanism (if any) for adding ECN support to the different packet types considered in this document. Whenever data input is needed to assist in a design choice, it is spelled out throughout the document.

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

1.3. Document Structure

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

2. Terminology

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

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

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

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

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

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

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

ECT: ECN-Capable Transport. One of the two codepoints ECT(0) or ECT(1) in the ECN field [RFC3168] of the IP header (v4 or v6). An

ECN-capable sender sets one of these to indicate that both transport end-points support ECN. When this specification says the sender sets an ECT codepoint, by default it means ECT(0). Optionally, it could mean ECT(1), which is in the process of being redefined for use by L4S experiments [I-D.ietf-tsvwg-ecn-experimentation] [I-D.briscoe-tsvwg-ecn-l4s-id].

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

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

3. Specification

3.1. Network (e.g. Firewall) Behaviour

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

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

3.2. Endpoint Behaviour

The changes to the specification of TCP over ECN [RFC3168] defined here solely alter the behaviour of the sending host for each half-connection. All changes can be deployed at each end-point independently of others.

The feedback behaviour at the receiver depends on whether classic ECN TCP feedback [RFC3168] or Accurate ECN (AcceCN) TCP feedback [I-D.ietf-tcpm-accurate-ecn] has been negotiated. Nonetheless,

neither receiver feedback behaviour is altered by the present specification.

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

TCP packet type	ECN field if AccECN f/b negotiated*	ECN field if RFC3168 f/b negotiated*	Congestion Response
SYN	ECT	not-ECT	Reduce IW
SYN-ACK [RFC5562]	ECT	ECT	Reduce IW as in [RFC5562]
Pure ACK	ECT	ECT	Usual cwnd response and optionally [RFC5690]
W Probe	ECT	ECT	Usual cwnd response
FIN	ECT	ECT	None or optionally [RFC5690]
RST	ECT	ECT	N/A
Re-XMT	ECT	ECT	Usual cwnd response

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

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

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

It can be seen that the sender can set ECT in all cases, except if it is not requesting AccECN feedback on the SYN. Therefore it is RECOMMENDED that the experimental AccECN specification [I-D.ietf-tcpm-accurate-ecn] is implemented (as well as the present specification), because it is expected that ECT on the SYN will give the most significant performance gain, particularly for short flows. Nonetheless, this specification also caters for the case where AccECN feedback is not implemented.

3.2.1. SYN

3.2.1.1. Setting ECT on the SYN

With classic [RFC3168] ECN feedback, the SYN was never expected to be ECN-capable, so the flag provided to feed back congestion was put to another use (it is used in combination with other flags to indicate that the responder supports ECN). In contrast, Accurate ECN (AccECN) feedback [I-D.ietf-tcpm-accurate-ecn] provides two codepoints in the SYN-ACK for the responder to feed back whether or not the SYN arrived marked CE.

Therefore, a TCP initiator MUST NOT set ECT on a SYN unless it also attempts to negotiate Accurate ECN feedback in the same SYN.

For the experiments proposed here, if the SYN is requesting AccECN feedback, the TCP sender will also set ECT on the SYN. It can ignore the prohibition in section 6.1.1 of RFC 3168 against setting ECT on such a SYN.

The following subsections about the SYN solely apply to this case where the initiator sent an ECT SYN.

MEASUREMENTS NEEDED: Measurements are needed to verify that if SYN packets with the ECT(0)/ECT(1)/CE codepoints are properly delivered by the network. We need to learn if there are cases if SYN packets are dropped because having the the ECT(0)/ECT(1)/CE codepoints. We also need to learn if the network clears SYN packet with the the ECT(0)/ECT(1)/CE codepoints. In addition, we need measurements to learn how current deployed base of servers react to SYN packets with ECT(0)/ECT(1)/CE codepoints whether they discard it, or process it and return a SYN/ACK packet proceeding with the connection. It would be also useful to measure how the network elements and the servers react to all possible combinations of ECN codepoints and NS/CWR/ECE flags.

3.2.1.2. Caching Lack of Support for ECT on SYNs

Until AccECN servers become widely deployed, a TCP initiator that sets ECT on a SYN (which implies the same SYN also requests AccECN, as required above) SHOULD also maintain a cache per server to record any failure of previous attempts.

The initiator will record any server's SYN-ACK response that does not support AccECN. Subsequently the initiator will not set ECT on a SYN to such a server, but it can still always request AccECN support (because the response will state any earlier stage of ECN evolution that the server supports with no performance penalty). The initiator will discover a server that has upgraded to support AccECN as soon as it next connects, then it can remove the server from its cache and subsequently always set ECT for that server.

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

There is no need to cache successful attempts, because the default ECT SYN behaviour performs optimally on success anyway. Servers that do not support ECN as a whole probably do not need to be recorded separately from non-support of AccECN because the response to a request for AccECN immediately states which stage in the evolution of ECN the server supports (AccECN [I-D.ietf-tcpm-accurate-ecn], classic ECN [RFC3168] or no ECN).

The above strategy is named "optimistic ECT and cache failures". It is believed to be sufficient based on initial measurements and assumptions detailed in Section 4.2.1, which also gives alternative strategies in case larger scale measurements uncover different scenarios.

3.2.1.3. SYN Congestion Response

If the SYN-ACK returned to the TCP initiator confirms that the server supports AccECN, it will also indicate whether or not the SYN was CE-marked. If the SYN was CE-marked, the initiator MUST reduce its Initial Window (IW) and SHOULD reduce it to 1 SMSS (sender maximum segment size).

If the SYN-ACK shows that the server does not support AccECN, the TCP initiator MUST conservatively reduce its Initial Window and SHOULD reduce it to 1 SMSS. A reduction to greater than 1 SMSS MAY be appropriate (see Section 4.2.1). Conservatism is necessary because a non-AccECN SYN-ACK cannot show whether the SYN was CE-marked.

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

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

If an initial window of 10 (IW10 [RFC6928]) is implemented, Section 5 gives additional recommendations.

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

An ECT SYN might be lost due to an over-zealous path element (or server) blocking ECT packets that do not conform to RFC 3168. However, loss is commonplace for numerous other reasons, e.g. congestion loss at a non-ECN queue on the forward or reverse path, transmission errors, etc. Alternatively, the cause of the blockage might be the attempt to negotiate AccECN, or possibly other unrelated options on the SYN.

To expedite connection set-up if, after sending an ECT SYN, the retransmission timer expires, the TCP initiator SHOULD send a SYN with the not-ECT codepoint in the IP header. If other experimental fields or options were on the SYN, it will also be necessary to follow their specifications for fall-back too. It would make sense to co-ordinate all the strategies for fall-back in order to isolate the specific cause of the problem.

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

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

3.2.2. SYN-ACK

3.2.2.1. Setting ECT on the SYN-ACK

For the experiments proposed here, the TCP implementation will set ECT on SYN-ACKs. It can ignore the requirement in section 6.1.1 of RFC 3168 to set not-ECT on a SYN-ACK.

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

Some classic ECN implementations might ignore a CE-mark on a SYN-ACK, or even ignore a SYN-ACK packet entirely if it is set to ECT or CE. This is a possibility because an RFC 3168 implementation would not necessarily expect a SYN-ACK to be ECN-capable.

FOR DISCUSSION: To eliminate this problem, the WG could decide to prohibit setting ECT on SYN-ACKs unless AccECN has been negotiated. However, this issue already came up when the IETF first decided to experiment with ECN on SYN-ACKs [RFC5562] and it was decided to go ahead without any extra precautionary measures because the risk was low. This was because the probability of encountering the problem was believed to be low and the harm if the problem arose was also low (see Appendix B of RFC 5562).

MEASUREMENTS NEEDED: Server-side experiments could determine whether this specific problem is indeed rare across the current installed base of clients that support ECN.

3.2.2.2. SYN-ACK Congestion Response

A host that sets ECT on SYN-ACKs MUST reduce its initial window in response to any congestion feedback, whether using classic ECN or AccECN. It SHOULD reduce it to 1 SMSS. This is different to the behaviour specified in an earlier experiment that set ECT on the SYN-ACK [RFC5562]. This is justified in Section 4.3.

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

The congestion response to CE-marking on a SYN-ACK for a server that implements either the TCP Fast Open experiment (TFO [RFC7413]) or the

initial window of 10 experiment (IW10 [RFC6928]) is discussed in Section 5.

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

After the responder sends a SYN-ACK with ECT set, if its retransmission timer expires it SHOULD resend a SYN-ACK with not-ECT set. If other experimental fields or options were on the SYN, it will also be necessary to follow their specifications for fall-back too. It would make sense to co-ordinate all the strategies for fall-back in order to isolate the specific cause of the problem.

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

This fall-back strategy is the same as that for ECT SYN-ACKs in [RFC5562]. Other fall-back strategies MAY be adopted if found to be more effective, e.g. one retransmission attempt using ECT before reverting to not-ECT.

3.2.3. Pure ACK

For the experiments proposed here, the TCP implementation will set ECT on pure ACKs. It can ignore the requirement in section 6.1.4 of RFC 3168 to set not-ECT on a pure ACK.

A host that sets ECT on pure ACKs MUST reduce its congestion window in response to any congestion feedback, in order to regulate any data segments it might be sending amongst the pure ACKs. It MAY also implement AckCC [RFC5690] to regulate the pure ACK rate, but this is not required. Note that, in comparison, TCP Congestion Control [RFC5681] does not require a TCP to detect or respond to loss of pure ACKs at all; it requires no reduction in congestion window or ACK rate.

The question of whether the receiver of pure ACKs is required to feed back any CE marks on them is a matter for the relevant feedback specification ([RFC3168] or [I-D.ietf-tcpm-accurate-ecn]). It is outside the scope of the present specification. Currently AccECN feedback is required to count CE marking of any control packet including pure ACKs. Whereas RFC 3168 is silent on this point, so

feedback of CE-markings might be implementation specific (see Section 4.4.1).

DISCUSSION: An AccECN deployment or an implementation of RFC 3168 that feeds back CE on pure ACKs will be at a disadvantage compared to an RFC 3168 implementation that does not. To solve this, the WG could decide to prohibit setting ECT on pure ACKs unless AccECN has been negotiated. If it does, the penultimate sentence of the Introduction will need to be modified.

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

3.2.4. Window Probe

For the experiments proposed here, the TCP sender will set ECT on window probes. It can ignore the prohibition in section 6.1.6 of RFC 3168 against setting ECT on a window probe.

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

A receive window of zero indicates that the application is not consuming data fast enough and does not imply anything about network congestion. Once the receive window opens, the congestion window might become the limiting factor, so it is correct that CE-marked probes reduce the congestion window. However, CE-marking on window probes does not reduce the rate of the probes themselves. This is unlikely to present a problem, given the duration between window probes doubles [RFC1122] as long as the receiver is advertising a zero window (currently minimum 1 second, maximum at least 1 minute [RFC6298]).

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

3.2.5. FIN

A TCP implementation can set ECT on a FIN.

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

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

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

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

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

3.2.6. RST

A TCP implementation can set ECT on a RST.

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

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

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

3.2.7. Retransmissions

For the experiments proposed here, the TCP sender will set ECT on retransmitted segments. It can ignore the prohibition in section 6.1.5 of RFC 3168 against setting ECT on retransmissions.

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

sender, which in turn would be fooled into continually halving its congestion window.

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

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

4. Rationale

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

4.1. The Reliability Argument

Section 5.2 of RFC 3168 states:

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

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

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

criterion, since either the packet is delivered and the CE signal is delivered to the endpoint, or the packet is dropped and the original congestion signal (packet loss) is delivered to the endpoint.

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

4.2. SYNs

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

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

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

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

4.2.1. Argument 1a: Unrecognized CE on the SYN

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

The accurate ECN (AcceCN) protocol [I-D.ietf-tcpm-accurate-ecn] has since been designed to solve this problem, using a two-pronged approach. First AcceCN uses the 3 ECN bits in the TCP header as 8 codepoints, so there is space for the responder to feed back whether there was CE on the SYN. Second a TCP initiator can always request AcceCN support on every SYN, and any responder reveals its level of ECN support: AcceCN, classic ECN, or no ECN. Therefore, if a

responder does indicate that it supports AcceCN, the initiator can be sure that, if there is no CE feedback on the SYN-ACK, then there really was no CE on the SYN.

An initiator can combine AcceCN with three possible strategies for setting ECT on a SYN:

- (S1): Pessimistic ECT and cache successes: The initiator always requests AcceCN in the SYN, but without setting ECT. Then it records those servers that confirm that they support AcceCN in a cache. On a subsequent connection to any server that supports AcceCN, the initiator can then set ECT on the SYN.
- (S2): Optimistic ECT: The initiator always sets ECT optimistically on the initial SYN and it always requests AcceCN support. Then, if the server response shows it has no AcceCN logic (so it cannot feed back a CE mark), the initiator conservatively behaves as if the SYN was CE-marked, by reducing its initial window.
 - A. No cache: The optimistic ECT strategy ought to work fairly well without caching any responses.
 - B. Cache failures: The optimistic ECT strategy can be improved by recording solely those servers that do not support AcceCN. On subsequent connections to these non-AcceCN servers, the initiator will still request AcceCN but not set ECT on the SYN. Then, the initiator can use its full initial window (if it has enough request data to need it). Longer term, as servers upgrade to AcceCN, the initiator will remove them from the cache and use ECT on subsequent SYNs to that server.
- (S3): ECT by configuration: In a controlled environment, the administrator can make sure that servers support ECN-capable SYN packets. Examples of controlled environments are single-tenant DCs, and possibly multi-tenant DCs if it is assumed that each tenant mostly communicates with its own VMs.

For unmanaged environments like the public Internet, pragmatically the choice is between strategies (S1) and (S2B):

- o The "pessimistic ECT and cache successes" strategy (S1) suffers from exposing the initial SYN to the prevailing loss level, even if the server supports ECT on SYNs, but only on the first connection to each AcceCN server.

- o The "optimistic ECT and cache failures" strategy (S2B) exploits a server's support for ECT on SYNs from the very first attempt. But if the server turns out not to support AccECN, the initiator has to conservatively limit its initial window - usually unnecessarily. Nonetheless, initiator request data (as opposed to server response data) is rarely larger than 1 SMSS anyway {ToDo: reference? (this information was given informally by Yuchung Cheng)}.

The normative specification for ECT on a SYN in Section 3.2.1 uses the "optimistic ECT and cache failures" strategy (S2B) on the assumption that an initial window of 1 SMSS is usually sufficient for client requests anyway. Clients that often initially send more than 1 SMSS of data could use strategy (S1) during initial deployment, and strategy (S2B) later (when the probability of servers supporting AccECN and the likelihood of seeing some CE marking is higher). Also, as deployment proceeds, caching successes (S1) starts off small then grows, while caching failures (S2B) becomes large at first, then shrinks.

MEASUREMENTS NEEDED: Measurements are needed to determine whether one or the other strategy would be sufficient for any particular client, or whether a particular client would need both strategies in different circumstances.

4.2.2. Argument 1b: Unrecognized ECT on the SYN

Given, until now, ECT-marked SYN packets have been prohibited, it cannot be assumed they will be accepted. According to a study using 2014 data [ecn-pam] from a limited range of vantage points, out of the top 1M Alexa web sites, 4791 (0.82%) IPv4 sites and 104 (0.61%) IPv6 sites failed to establish a connection when they received a TCP SYN with any ECN codepoint set in the IP header and the appropriate ECN flags in the TCP header. Of these, about 41% failed to establish a connection due to the ECN flags in the TCP header even with a Not-ECT ECN field in the IP header (i.e. despite full compliance with RFC 3168). Therefore adding the ECN-capability to SYNs was increasing connection establishment failures by about 0.4%.

MEASUREMENTS NEEDED: In order to get these failures fixed, data will be needed on which of the possible causes below is behind them.

RFC 3168 says "a host MUST NOT set ECT on SYN [...] packets", but it does not say what the responder should do if an ECN-capable SYN arrives. So perhaps some responder implementations are checking that the SYN complies with RFC 3168, then silently ignoring non-compliant SYNs (or perhaps returning a RST). Also some middleboxes (e.g.

firewalls) might be discarding non-compliant SYNs. For the future, [I-D.ietf-tsvwg-ecn-experimentation] updates RFC 3168 to clarify that middleboxes "SHOULD NOT" do this, but that does not alter the past.

Whereas RSTs can be dealt with immediately, silent failures introduce a retransmission timeout delay (default 1 second) at the initiator before it attempts any fall back strategy. Ironically, making SYNs ECN-capable is intended to avoid the timeout when a SYN is lost due to congestion. Fortunately, where discard of ECN-capable SYNs is due to policy it will occur predictably, not randomly like congestion. So the initiator can avoid it by caching those sites that do not support ECN-capable SYNs. This further justifies the use of the "optimistic ECT and cache failures" strategy in Section 3.2.1.

MEASUREMENTS NEEDED: Experiments are needed to determine whether blocking of ECT on SYNs is widespread, and how many occurrences of problems would be masked by how few cache entries.

If blocking is too widespread for the "optimistic ECT and cache failures" strategy (S2B), the "pessimistic ECT and cache successes" strategy (Section 4.2.1) would be better.

MEASUREMENTS NEEDED: Then measurements would be needed on whether failures were still widespread on the second connection attempt after the more careful ("pessimistic") first connection.

If so, it might be necessary to send a not-ECT SYN soon after the first ECT SYN (possibly with a delay between them - effectively reducing the retransmission timeout) and only accept the non-ECT connection if it returned first. This would reduce the performance penalty for those deploying ECT SYN support.

FOR DISCUSSION: If this becomes necessary, how much delay ought to be required before the second SYN? Certainly less than the standard RTO (1 second). But more or less than the maximum RTT expected over the surface of the earth (roughly 250ms)? Or even back-to-back?

However, based on the data above from [ecn-pam], even a cache of a dozen or so sites ought to avoid all ECN-related performance problems with roughly the Alexa top thousand. So it is questionable whether sending two SYNs will be necessary, particularly given failures at well-maintained sites could reduce further once ECT SYNs are standardized.

4.2.3. Argument 2: DoS Attacks

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

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

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

Further experiments are needed to test how much malicious hosts can use ECT to augment flooding attacks without triggering AQMs to turn off ECN support (flying "just under the radar"). If it is found that ECT can only slightly augment flooding attacks, the risk of such attacks will need to be weighed against the performance benefits of ECT SYNs.

4.3. SYN-ACKs

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

The IETF has already specified an experiment with ECN-capable SYN-ACK packets [RFC5562]. It was inspired by the ECN+ paper, but it

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

The present ECN experiment uses the ECN+ congestion response, not ECN+/TryOnce. First we argue against the rationale for ECN+/TryOnce given in sections 4.4 and 6.2 of [RFC5562]. It starts with a rather too literal interpretation of the requirement in RFC 3168 that says TCP's response to a single CE mark has to be "essentially the same as the congestion control response to a **single** dropped packet." TCP's response to a dropped initial (SYN or SYN-ACK) packet is to wait for the retransmission timer to expire (currently 1s). However, this long delay assumes the worst case between two possible causes of the loss: a) heavy overload; or b) the normal capacity-seeking behaviour of other TCP flows. When the network is still delivering CE-marked packets, it implies that there is an AQM at the bottleneck and that it is not overloaded. This is because an AQM under overload will disable ECN (as recommended in section 7 of RFC 3168 and repeated in section 4.2.1 of RFC 7567). So scenario (a) can be ruled out. Therefore, TCP's response to a CE-marked SYN-ACK can be similar to its response to the loss of any packet, rather than backing off as if the special initial packet of a flow has been lost.

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

Therefore, we have to work out a reasoned argument for what to do. If an AQM is CE-marking packets, it implies there is already a queue and it is probably already somewhere around the AQM's operating point - it is unlikely to be well below and it might be well above. So, it does not seem sensible to add a number of packets at once. On the other hand, it is highly unlikely that the SYN-ACK itself pushed the AQM into congestion, so it will be safe to introduce another single segment immediately (1 RTT after the SYN-ACK). Therefore, starting to probe for capacity with a slow start from an initial window of 1 segment seems appropriate to the circumstances. This is the approach adopted in Section 3.2.2.

4.4. Pure ACKs

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

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

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

Later on, in section 6.1.4 it reads:

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

We next address each of the arguments presented above.

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

The second argument says that ECN ought not to be enabled unless there is a mechanism to respond to it. However, actually there is a mechanism to respond to congestion on a pure ACK that RFC 3168 has overlooked - the congestion window mechanism. When data segments and pure ACKs are interspersed, congestion notifications ought to regulate the congestion window, whether they are on data segments or

on pure ACKs. Otherwise, if ECN is disabled on Pure ACKs, and if (say) 70% of the segments in one direction are Pure ACKs, about 70% of the congestion notifications will be missed and the data segments will not be correctly regulated.

So RFC 3168 ought to have considered two congestion response mechanisms - reducing the congestion window (cwnd) and reducing the ACK rate - and only the latter was missing. Further, RFC 3168 was incorrect to assume that, if one ACK was a pure ACK, all segments in the same direction would be pure ACKs. Admittedly a continual stream of pure ACKs in one direction is quite a common case (e.g. a file download). However, it is also common for the pure ACKs to be interspersed with data segments (e.g. HTTP/2 browser requests controlling a web application). Indeed, it is more likely that any congestion experienced by pure ACKs will be due to mixing with data segments, either within the same flow, or within competing flows.

This insight swings the argument towards enabling ECN on pure ACKs so that CE marks can drive the cwnd response to congestion (whenever data segments are interspersed with the pure ACKs). Then to separately decide whether an ACK rate response is also required (when they are ECN-enabled). The two types of response are addressed separately in the following two subsections, then a final subsection draws conclusions.

4.4.1. Cwnd Response to CE-Marked Pure ACKs

If the sender of pure ACKs sets them to ECT, the bullets below assess whether the three stages of the congestion response mechanism will all work for each type of congestion feedback (classic ECN [RFC3168] and AccECN [I-D.ietf-tcpm-accurate-ecn]):

Detection: The receiver of a pure ACK can detect a CE marking on it:

- * Classic feedback: the receiver will not expect CE marks on pure ACKs, so it will be implementation-dependent whether it happens to check for CE marks on all packets.
- * AccECN feedback: the AccECN specification requires the receiver of any TCP packets to count any CE marks on them (whether or not control packets are ECN-capable).

Feedback: TCP never ACKs a pure ACK, but the receiver of a CE-mark on a pure ACK can feed it back when it sends a subsequent data segment (if it ever does):

- * Classic feedback: the receiver (of the pure ACKs) would set the echo congestion experienced (ECE) flag in the TCP header as normal.
- * AccECN feedback: the receiver continually feeds back a count of the number of CE-marked packets that it has received (and, if possible, a count of CE-marked bytes).

Congestion response: In either case (classic or AccECN feedback), if the TCP sender does receive feedback about CE-markings on pure ACKs, it will react in the usual way by reducing its congestion window accordingly. This will regulate the rate of any data packets it is sending amongst the pure ACKs.

4.4.2. ACK Rate Response to CE-Marked Pure ACKs

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

Since RFC 3168 was published, Acknowledgement Congestion Control (AckCC) techniques have been documented in [RFC5690] (informational). So any pair of TCP end-points can choose to agree to regulate the delayed ACK ratio in response to lost or CE-marked pure ACKs. However, the protocol has a number of open deployment issues (e.g. it relies on two new TCP options, one of which is required on the SYN where option space is at a premium and, if either option is blocked by a middlebox, no fall-back behaviour is specified). The new TCP options addressed two problems, namely that TCP had: i) no mechanism to allow ECT to be set on pure ACKs; and ii) no mechanism to feed back loss or CE-marking of pure ACKs. A combination of the present specification and AccECN addresses both these problems, at least for ECN marking. So it might now be possible to design an ECN-specific ACK congestion control scheme without the extra TCP options proposed in RFC 5690. However, such a mechanism is out of scope of the present document.

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

probability becomes high. Therefore discard can still be relied on to thin out ECN-enabled pure ACKs as a last resort.

4.4.3. Summary: Enabling ECN on Pure ACKs

In the case when AccECN has been negotiated, the arguments for ECT (and CE) on pure ACKs heavily outweigh those against. ECN is always more and never less reliable for delivery of congestion notification. The cwnd response has been overlooked as a mechanism for responding to congestion on pure ACKs, so it is incorrect not to set ECT on pure ACKs when they are interspersed with data segments. And when they are not, packet discard still acts as the "congestion response of last resort". In contrast, not setting ECT on pure ACKs is certainly detrimental to performance, because when a pure ACK is lost it can prevent the release of new data. Separately, AckCC (or perhaps an improved variant exploiting AccECN) could optionally be used to regulate the spacing between pure ACKs. However, it is not clear whether AckCC is justified.

In the case when Classic ECN has been negotiated, there is still an argument for ECT (and CE) on pure ACKs, but it is less clear-cut. Some existing RFC 3168 implementations might happen to (unintentionally) provide the correct feedback to support a cwnd response. Even for those that did not, setting ECT on pure ACKs would still be better for performance than not setting it and do no extra harm. If AckCC was required, it is designed to work with RFC 3168 ECN.

4.5. Window Probes

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

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

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

The reliability argument has already been addressed in Section 4.1.

Allowing ECT on window probes could considerably improve performance because, once the receive window has reopened, if a window probe is lost the sender will stall until the next window probe reaches the

receiver, which might be after the maximum retransmission timeout (at least 1 minute [RFC6928]).

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

4.6. FINs

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

4.7. RSTs

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

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

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

- o As explained above, a congestion response by the sender of a CE-marked RST message is not possible;
- o So the only reason for the sender setting ECT on a RST would be to improve the reliability of the message's delivery;
- o RST messages are used to both mount and mitigate attacks:

- * Spoofed RST messages are used by attackers to terminate ongoing connections, although the mitigations in RFC 5961 have considerably raised the bar against off-path RST attacks;
- * Legitimate RST messages allow endpoints to inform their peers to eliminate existing state that correspond to non existing connections, liberating resources e.g. in DoS attacks scenarios;
- o AQMs are advised to disable ECN marking during persistent overload, so:
 - * it is harder for an attacker to exploit ECN to intensify an attack;
 - * it is harder for a legitimate user to exploit ECN to more reliably mitigate an attack
- o Prohibiting ECT on a RST would deny the benefit of ECN to legitimate RST messages, but not to attackers who can disregard RFCs;
- o If ECT were prohibited on RSTs
 - * it would be easy for security middleboxes to discard all ECN-capable RSTs;
 - * However, unlike a SYN flood, it is already easy for a security middlebox (or host) to distinguish a RST flood from legitimate traffic [RFC5961], and even if a some legitimate RSTs are accidentally removed as well, legitimate connections still function.

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

4.8. Retransmitted Packets.

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

The reliability argument has already been addressed in Section 4.1.

Protection against DoS attacks is not afforded by prohibiting ECT on retransmitted packets. An attacker can set CE on spoofed retransmissions whether or not it is prohibited by an RFC. Protection against the DoS attack described in section 6.1.5 of RFC 3168 is solely afforded by the requirement that "the TCP data receiver SHOULD ignore the CE codepoint on out-of-window packets". Therefore in Section 3.2.7 the sender is allowed to set ECT on retransmitted packets, in order to reduce the chance of them being dropped. We also strengthen the receiver's requirement from "SHOULD ignore" to "MUST ignore". And we generalize the receiver's requirement to include failure of any validity check, not just out-of-window checks, in order to include the more stringent validity checks in RFC 5961 that have been developed since RFC 3168.

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

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

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

5. Interaction with popular variants or derivatives of TCP

The following subsections discuss any interactions between setting ECT on all all packets and using the following popular variants or derivatives of TCP: SCTP, IW10 and TFO. This section is informative not normative, because no interactions have been identified that require any change to specifications. The subsection on IW10 discusses potential changes to specifications but recommends that no changes are needed.

TCP variants that have been assessed and found not to interact adversely with ECT on TCP control packets are: SYN cookies (see

Appendix A of [RFC4987] and section 3.1 of [RFC5562]), TCP Fast Open (TFO [RFC7413]) and L4S [I-D.briscoe-tsvwg-l4s-arch].

5.1. SCTP

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

5.2. IW10

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

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

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

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

- o as long as the initiator is caching failures to negotiate AccECN, subsequent attempts to access the same server will not use ECT on the SYN anyway, so there will no longer be any need to conservatively reduce IW;
- o currently it is not common for a TCP initiator (client) to have more than one data segment to send {ToDo: evidence/reference?} - IW10 is primarily exploited by TCP servers.

If a responder receives feedback that the SYN-ACK was CE-marked, Section 3.2.2.2 mandates that it reduces its initial window to 1 SMSS. When the responder also implements IW10, it is particularly important to adhere to this requirement in order to avoid overflowing a queue that is clearly already congested.

5.3. TFO

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

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

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

6. Security Considerations

Section 3.2.6 considers the question of whether ECT on RSTs will allow RST attacks to be intensified. There are several security arguments presented in RFC 3168 for preventing the ECN marking of TCP control packets and retransmitted segments. We believe all of them have been properly addressed in Section 4, particularly Section 4.2.3 and Section 4.8 on DoS attacks using spoofed ECT-marked SYNs and spoofed CE-marked retransmissions.

7. IANA Considerations

There are no IANA considerations in this memo.

8. Acknowledgments

Thanks to Mirja Kuehlewind and David Black for their useful reviews.

The work of Marcelo Bagnulo has been performed in the framework of the H2020-ICT-2014-2 project 5G NORMA. His contribution reflects the

consortiums view, but the consortium is not liable for any use that may be made of any of the information contained therein.

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<http://www.rfc-editor.org/info/rfc3168>>.
- [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, <<http://www.rfc-editor.org/info/rfc5562>>.
- [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, <<http://www.rfc-editor.org/info/rfc5961>>.
- [I-D.ietf-tcpm-accurate-ecn] Briscoe, B., Kuehlewind, M., and R. Scheffenegger, "More Accurate ECN Feedback in TCP", draft-ietf-tcpm-accurate-ecn-02 (work in progress), October 2016.
- [I-D.ietf-tsvwg-ecn-experimentation] Black, D., "Explicit Congestion Notification (ECN) Experimentation", draft-ietf-tsvwg-ecn-experimentation-02 (work in progress), April 2017.

9.2. Informative References

- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<http://www.rfc-editor.org/info/rfc793>>.
- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<http://www.rfc-editor.org/info/rfc1122>>.

- [RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, DOI 10.17487/RFC3540, June 2003, <<http://www.rfc-editor.org/info/rfc3540>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<http://www.rfc-editor.org/info/rfc4960>>.
- [RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <<http://www.rfc-editor.org/info/rfc4987>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<http://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, <<http://www.rfc-editor.org/info/rfc5690>>.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <<http://www.rfc-editor.org/info/rfc6298>>.
- [RFC6928] Chu, J., Dukkkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <<http://www.rfc-editor.org/info/rfc6928>>.
- [RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <<http://www.rfc-editor.org/info/rfc7413>>.
- [RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <<http://www.rfc-editor.org/info/rfc7567>>.
- [I-D.briscoe-tsvwg-ecn-l4s-id]
Schepper, K., Briscoe, B., and I. Tsang, "Identifying Modified Explicit Congestion Notification (ECN) Semantics for Ultra-Low Queuing Delay", draft-briscoe-tsvwg-ecn-l4s-id-02 (work in progress), October 2016.

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

Briscoe, B., Schepper, K., and M. Bagnulo, "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture", draft-briscoe-tsvwg-l4s-arch-02 (work in progress), March 2017.

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

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

[judd-nsdi]

Judd, G., "Attaining the promise and avoiding the pitfalls of TCP in the Datacenter", USENIX Symposium on Networked Systems Design and Implementation (NSDI'15) pp.145-157, May 2015.

[ecn-pam]

Trammell, B., Kuehlewind, M., Boppart, D., Learmonth, I., Fairhurst, G., and R. Scheffenegger, "Enabling Internet-Wide Deployment of Explicit Congestion Notification", Int'l Conf. on Passive and Active Network Measurement (PAM'15) pp193-205, 2015.

[ECN-PLUS]

Kuzmanovic, A., "The Power of Explicit Congestion Notification", ACM SIGCOMM 35(4):61--72, 2005.

Authors' Addresses

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

Phone: 34 91 6249500
Email: marcelo@it.uc3m.es
URI: <http://www.it.uc3m.es>

Bob Briscoe
Simula Research Lab

Email: ietf@bobbbriscoe.net
URI: <http://bobbbriscoe.net/>

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 23, 2020

P. Balasubramanian
Y. Huang
M. Olson
Microsoft
July 22, 2019

HyStart++: Modified Slow Start for TCP
draft-balasubramanian-tcpm-hystartplusplus-01

Abstract

This informational memo describes HyStart++, a simple modification to the slow start phase of TCP congestion control algorithms. HyStart++ combines the use of one variant of HyStart and Limited Slow Start (LSS) to prevent overshooting of the ideal sending rate value, while also mitigating poor performance which can result from false positives when HyStart is used alone. This memo also describes the details of the current implementation in the Windows operating system.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 23, 2020.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	2
2. Terminology	3
3. HyStart++ Algorithm	3
3.1. Use of HyStart Delay Increase and Limited Slow Start	3
3.2. Algorithm Details	3
3.3. Constant used and tuning	5
4. Security Considerations	5
5. IANA Considerations	5
6. Acknowledgements	5
7. References	5
7.1. Normative References	5
7.2. Informative References	6
Authors' Addresses	6

1. Introduction

[RFC0793] and [RFC5681] describe the slow start mechanism for TCP. The slow start algorithm is used when congestion window (cwnd) is less than the slow start threshold (ssthresh). During slow start, a TCP increments cwnd by at most SMSS bytes per ACK. In absence of packet loss signals, slow start effectively doubles the congestion window each round trip time.

While traditional TCP slow start can ramp up very quickly, it frequently overshoots the ideal sending rate and causes a lot of unnecessary packet drops. TCP has several mechanisms for loss recovery, but they are only effective for moderate loss. When these techniques are unable to recover lost packets, a last-resort retransmission timeout (RTO) is used to trigger packet recovery. In most operating systems, the minimum RTO is set to a large value (200 ms or 300ms) to prevent spurious timeouts. This results in a long idle time which drastically impairs flow completion times.

HyStart++ adds delay increase as a signal to exit slow start before any packet loss occurs. This is one of two algorithms specified in [HyStart]. After the HyStart delay algorithm finds an exit point, LSS is used for further congestion window increases until the first packet loss occurs.

This document describes HyStart++ as implemented in the Microsoft Windows operating system. HyStart++ is widely deployed on the public

Internet. A precise documentation of running code enables follow-up IETF Experimental or Standards Track RFCs. It also enables other implementations and sharing of results for various workloads.

2. Terminology

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

3. HyStart++ Algorithm

3.1. Use of HyStart Delay Increase and Limited Slow Start

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

After the HyStart "Delay Increase" algorithm triggers an exit from slow start, LSS (described in [RFC3742]) is used to increase Cwnd until the first packet loss occurs. LSS is used because the HyStart exit is often premature as a result of RTT fluctuations or transient queue buildup. LSS grows the cwnd fast but much slower than traditional slow start. LSS helps avoid massive packet losses and subsequent time spent in loss recovery or retransmission timeout.

3.2. Algorithm Details

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

Define windowEnd as a sequence number initialize to SND.UNA

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

At the start of each round during slow start:

lastRoundMinRTT = currentRoundMinRTT

currentRoundMinRTT = infinity

rttSampleCount = 0

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

Update the cwnd

$$\text{cwnd} = \text{cwnd} + \min(N, \text{SMSS})$$

Keep track of minimum observed RTT

$$\text{currentRoundMinRTT} = \min(\text{currentRoundMinRTT}, \text{currRTT})$$

where currRTT is the measured RTT based on the incoming ACK

$$\text{rttSampleCount} += 1$$

For rounds where cwnd is at or higher than MIN_SSTHRESH and N_RTT_SAMPLE RTT samples have been obtained, check if delay increase triggers slow start exit

$$\text{if } (\text{cwnd} \geq \text{MIN_SSTHRESH} \text{ AND } \text{rttSampleCount} \geq \text{N_RTT_SAMPLE})$$

$$\text{Eta} = \text{clamp}(\text{MIN_ETA}, \text{lastRoundMinRTT} / 8, \text{MAX_ETA})$$

$$\text{if } (\text{currentRoundMinRTT} \geq (\text{lastRoundMinRTT} + \text{Eta}))$$

$$\text{ssthresh} = \text{cwnd}$$

$$\text{exit slow start and enter LSS}$$

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

$$K = \text{cwnd} / (\text{LSS_DIVISOR} * \text{ssthresh})$$

$$\text{cwnd} = \max(\text{cwnd} + N / K, \text{CA_cwnd}())$$

CA_cwnd() denotes the cwnd that a congestion control algorithm would have increased to if congestion avoidance started instead of LSS. LSS grows cwnd very fast but for long-lived flows in high BDP networks, the congestion avoidance algorithm could increase cwnd much faster. For example, CUBIC congestion avoidance [RFC8312] in convex region can ramp up cwnd rapidly. Taking the max can help improve performance when exiting slow start prematurely.

HyStart++ ends when cwnd exceeds ssthresh or when congestion is observed.

3.3. Constant used and tuning

The Windows operating system implementation of HyStart++ uses the following constants:

MIN_SSTHRESH = 16

MIN_ETA = 4 msec

MAX_ETA = 16 msec

LSS_DIVISOR = 0.25

N_RTT_SAMPLE = 8

An implementation MAY experiment with these constants and tune them for different network characteristics. Windows operating system implementation uses the same values for all connections.

An implementation MAY choose to use HyStart++ for all slow starts including the ones post a retransmission timeout, or a long idle period. The Windows operating system implementation uses HyStart++ only for the initial slow start and uses traditional slow start for subsequent ones. This is acceptable because subsequent slow starts will use the discovered ssthresh value to exit slow start.

4. Security Considerations

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

5. IANA Considerations

This document has no actions for IANA.

6. Acknowledgements

Neal Cardwell suggested the idea for using the maximum of cwnd value computed by LSS and congestion avoidance after exiting slow start.

7. References

7.1. Normative References

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

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3742] Floyd, S., "Limited Slow-Start for TCP with Large Congestion Windows", RFC 3742, DOI 10.17487/RFC3742, March 2004, <<https://www.rfc-editor.org/info/rfc3742>>.
- [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>>.

7.2. Informative References

- [HyStart] Ha, S. and I. Ree, "Hybrid Slow Start for High-Bandwidth and Long-Distance Networks", DOI 10.1145/1851182.1851192, International Workshop on Protocols for Fast Long-Distance Networks, March 2010, <<https://doi.org/10.1016/j.comnet.2011.01.014>>.
- [RFC8312] Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", RFC 8312, DOI 10.17487/RFC8312, February 2018, <<https://www.rfc-editor.org/info/rfc8312>>.

Authors' Addresses

Praveen Balasubramanian
Microsoft
One Microsoft Way
Redmond, WA 98052
USA

Phone: +1 425 538 2782
Email: pravb@microsoft.com

Yi Huang
Microsoft

Phone: +1 425 703 0447
Email: huanyi@microsoft.com

Matt Olson
Microsoft

Phone: +1 425 538 8598
Email: maolson@microsoft.com

TCPM Working Group
Internet-Draft
Intended status: Experimental
Expires: May 7, 2020

C. Gomez
UPC
J. Crowcroft
University of Cambridge
November 4, 2019

TCP ACK Pull
draft-gomez-tcpm-ack-pull-01

Abstract

Delayed Acknowledgments (ACKs) allow reducing protocol overhead in many scenarios. However, in some cases, Delayed ACKs may significantly degrade network and device performance in terms of link utilization, latency, memory usage and/or energy consumption. This document defines the TCP ACK Pull (AKP) mechanism, which allows a sender to request the ACK for a data segment to be sent without additional delay by the receiver. AKP makes use of one of the reserved bits in the TCP header, which is defined in this specification as the AKP flag.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 7, 2020.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	2
2. Conventions used in this document	4
3. ACK Pull Mechanism	4
4. The ACK Pull Flag	4
5. IANA Actions	4
6. Security Considerations	5
7. Acknowledgments	5
8. Annex: Alternative approaches	5
9. References	6
9.1. Normative References	6
9.2. Informative References	6
Authors' Addresses	7

1. Introduction

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

However, there exist traffic patterns and scenarios for which Delayed ACKs can actually be detrimental to performance. When a segment carrying a message of a size up to one Maximum Segment Size (MSS) is transferred, if the message does not elicit an application-layer response, and a second data segment is not transferred earlier than the Delayed ACK timeout, the ACK is unnecessarily delayed, with a number of negative consequences.

When the Nagle algorithm is used, in some cases the sender may be prevented from sending more data while awaiting the Delayed ACK. In some high bit rate environment (e.g. Gigabit Ethernet) use cases, such a delay may be very large, and link utilization may be

dramatically reduced, as the Delayed ACK timeout is several orders of magnitude greater than the Round Trip Time (RTT) [RFC8490].

Delayed ACKs are also detrimental in Internet of Things (IoT) scenarios, where TCP is being increasingly used [I-D.ietf-lwig-tcp-constrained-node-networks]. Many IoT devices, such as sensors, transfer small messages (e.g. containing sensor readings) rather infrequently, therefore if the receiver uses Delayed ACKs, the ACK will often be unnecessarily delayed. The sender cannot release the memory resources associated to a transferred data segment until the ACK is received and processed. This may be a problem for many IoT devices, which are typically memory-constrained, and may even lead to subsequent packet drops if their scarce memory resources are blocked while awaiting an ACK. Moreover, if the IoT device uses a radio interface for communication, in some scenarios Delayed ACKs will lead to increased energy consumption (e.g. with the radio interface of the device staying in receive mode while awaiting the ACK). Since many IoT devices run on small batteries, the device lifetime may be significantly decreased. Furthermore, the delay suffered by the ACK may interact negatively with layer two mechanisms, especially in wireless network technologies where devices remain in low-power states for long intervals [RFC 8352], potentially leading to a further exacerbated delay (by even one or more orders of magnitude).

One approach that cannot be recommended as a general solution to solve the described problems is disabling Delayed ACKs at the receiving TCP. In fact, the latter may interact with a wide variety of devices and many of those may still benefit from the advantages of Delayed ACKs. In addition, in some cases, a sender may offer a mixed traffic pattern comprising single data segments that will lead to unnecessarily delayed ACKs, with other data segments upon which Delayed ACKs will act as intended. Therefore, the solution has to be provided at a per-segment granularity.

Since the presented problem is about low performance in various scenarios, another requirement for the solution is to provide a sender with a mechanism to request an immediate ACK for a data segment without incurring overhead in terms of header size increase or additional packets sent. For example, in IoT scenarios, every additional communicated byte consumes scarce resources (e.g. energy, bandwidth, computational resources); even further, each additional communicated packet may involve significant energy overhead.

This document defines the TCP ACK Pull (AKP) mechanism and an AKP flag in the TCP header. AKP allows a sender to request an ACK to be sent by a receiving TCP without additional delay upon reception of a data segment, by setting the AKP flag in that data segment. The AKP

flag uses one of the reserved bits in the TCP header. More specifically, the AKP flag uses bit 6 of byte 13 of the TCP header.

2. Conventions used in this document

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

3. ACK Pull Mechanism

When a TCP sender needs a data segment to be acknowledged by the receiving TCP without additional delay, the sender sets the AKP flag of the data segment TCP header. A receiving TCP conforming to this specification MUST process the AKP flag of a received segment. If the AKP flag is set, the receiving TCP MUST send an ACK without additional delay, regardless of whether the receiving TCP uses the Delayed ACKs mechanism.

4. The ACK Pull Flag

The AKP flag is defined as bit number 6 of the 13th byte of the TCP header. Figure 1 illustrates bytes 13 and 14 of the TCP header, including the AKP flag.

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

Figure 1: Definition of the AKP field within bytes 13 and 14 of the TCP Header.

(Note: as of the writing, bit 7 in the above figure is reserved, although this may change with the publication of [I-D.ietf-tcpm-accurate-ecn].)

5. IANA Actions

This document assigns bit 6 of the TCP header flags to the AKP flag. This flag will be defined as shown in Figure 2:

Bit	Name	Reference
6	AKP (ACK Pull)	RFC XXXX

Figure 2

[TO BE REMOVED: IANA is requested to update the existing entry in the Transmission Control Protocol (TCP) Header Flags registration (<https://www.iana.org/assignments/tcp-header-flags/tcp-header-flags.xhtml#tcp-header-flags-1>) for Bit 6 to 'AKP (ACK Pull)'.]

6. Security Considerations

TCP ACK Pull introduces a possible Denial of Service (DoS) attack on a resource-constrained receiver. An attacker might send a large number of messages to a victim node, requesting an immediate ACK in response to each one of them. This attack is easily avoided by ignoring the TCP ACK Pull flag.

7. Acknowledgments

Stuart Cheshire, Ted Lemon, Michael Scharf, and Christoph Paasch participated in a discussion that was seminal to this document.

Carles Gomez has been funded in part by the Spanish Government (Ministerio de Ciencia, Innovacion y Universidades) through the Jose Castillejo grant CAS18/00170 and by European Regional Development Fund (ERDF) and the Spanish Government through project TEC2016-79988-P, AEI/FEDER, UE. His contribution to this work has been carried out during his stay as a visiting scholar at the Computer Laboratory of the University of Cambridge.

8. Annex: Alternative approaches

Several mechanisms that have been proposed in the past allow increasing the amount of ACKs sent by the receiver. Some examples are Acknowledgment Congestion Control (AckCC) [RFC5690] and Tail Loss Probe (TLP) [I-D.ietf-tcpm-rack].

In AckCC, the sender tells the receiver the ACK Ratio R to use, where the receiver sends one ACK per R data packets received. AckCC defines a 2-byte "TCP ACK Congestion Control Permitted Option" for negotiating use of AckCC, whereas it defines a 3-byte "ACK Ratio TCP option" to communicate the ACK Ratio value from the sender to the receiver.

TLP is intended to avoid RTO-expiration-based retransmission when tail loss occurs by inducing additional ACKs at the receiver. This is achieved by sending a probe segment after a probe time-out (PTO) when data have been sent but not confirmed.

Another approach that allows eliciting an immediate ACK after sending a data segment is sending a subsequent segment carrying e.g. an already sent data byte. Another workaround, which is used in the Contiki operating system (a popular operating system for constrained devices in IoT scenarios) is to split the data to be sent into two segments of smaller size. A standard compliant TCP receiver will acknowledge the second MSS of data, which can improve throughput. However, this 'split hack' may not always work since a TCP receiver is required to acknowledge every second full-sized segment, but not two consecutive small segments. Furthermore, the overhead of sending two IP packets instead of one is another downside of the 'split hack'.

Note that all the approaches in this Annex involve increasing TCP header size of some segments, or involve sending additional packets. The main advantage of the AKP mechanism defined in this specification is allowing a sender to request an immediate ACK while incurring no overhead.

9. References

9.1. Normative References

- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

9.2. Informative References

- [I-D.ietf-lwig-tcp-constrained-node-networks] Gomez, C., Crowcroft, J., and M. Scharf, "TCP Usage Guidance in the Internet of Things (IoT)", draft-ietf-lwig-tcp-constrained-node-networks-08 (work in progress), June 2019.

- [I-D.ietf-tcpm-accurate-ecn]
Briscoe, B., Kuehlewind, M., and R. Scheffenegger, "More Accurate ECN Feedback in TCP", draft-ietf-tcpm-accurate-ecn-09 (work in progress), July 2019.
- [I-D.ietf-tcpm-rack]
Cheng, Y., Cardwell, N., Dukkkipati, N., and P. Jha, "RACK: a time-based fast loss detection algorithm for TCP", draft-ietf-tcpm-rack-06 (work in progress), November 2019.
- [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>>.
- [RFC8352] Gomez, C., Kovatsch, M., Tian, H., and Z. Cao, Ed., "Energy-Efficient Features of Internet of Things Protocols", RFC 8352, DOI 10.17487/RFC8352, April 2018, <<https://www.rfc-editor.org/info/rfc8352>>.
- [RFC8490] Bellis, R., Cheshire, S., Dickinson, J., Dickinson, S., Lemon, T., and T. Pusateri, "DNS Stateful Operations", RFC 8490, DOI 10.17487/RFC8490, March 2019, <<https://www.rfc-editor.org/info/rfc8490>>.

Authors' Addresses

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

Email: carlesgo@entel.upc.edu

Jon Crowcroft
University of Cambridge
JJ Thomson Avenue
Cambridge, CB3 0FD
United Kingdom

Email: jon.crowcroft@cl.cam.ac.uk

TCP Maintenance and Minor Extensions
Internet-Draft
Intended status: Experimental
Expires: 7 May 2020

R.W. Grimes
P. Heist
4 November 2019

Some Congestion Experienced in TCP
draft-grimes-tcpm-tcpsce-01

Abstract

This memo classifies a TCP code point ESCE ("Echo Some Congestion Experienced") for use in feedback of IP code point SCE ("Some Congestion Experienced").

Status of This Memo

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

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

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

This Internet-Draft will expire on 7 May 2020.

Copyright Notice

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

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

Table of Contents

1. Terminology	2
2. Introduction	2
3. Background	3
4. TCP Receiver	3
4.1. Single ACK implementation	3
4.2. Simple Delayed ACK implementation	3
4.3. Dithered Delayed ACK implementation	3
4.4. Advanced ACK implementation	4
4.5. ACK Thinning	4
5. TCP Sender	4
6. Related Work	4
6.1. More Accurate ECN Feedback in TCP	4
7. IANA Considerations	5
8. Security Considerations	5
9. Acknowledgements	5
10. Normative References	5
11. Informative References	6
Authors' Addresses	6

1. Terminology

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 [RFC2119] and [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Introduction

This memo requests a TCP header codepoint for use as ESCE.

This memo limits its scope to the definition of the TCP codepoint ESCE, with a few brief illustrations of how it may be used.

SCE provides early and proportional feedback to the CC (congestion control) algorithms for transport protocols, including but not limited to TCP. The [sce-repo] is a Linux kernel modified to support SCE, including:

- * Enhancements to Linux's [cake] (Common Applications Kept Enhanced) AQM to support SCE signaling
- * Modifications to the TCP receive path to reflect SCE signals back to the sender

- * The addition of three new TCP CC algorithms that modify the originals to add SCE support: Reno-SCE, DCTCP-SCE and Cubic-SCE (work in progress as of this writing)

3. Background

[I-D.morton-tsvwg-sce] defines the IP SCE codepoint.

4. TCP Receiver

The mechanism defined to feed back SCE signals to the sender explicitly makes use of the ESCE ("Echo Some Congestion Experienced") code point in the TCP header.

4.1. Single ACK implementation

Upon receipt of a packet an ACK is immediately generated, the SCE codepoint is copied into the ESCE codepoint of the ACK. This keeps the count of bytes SCE marked or not marked properly reflected in the ACK packet(s). This valid implementation has the downside of increasing ACK traffic. This implementation is NOT RECOMMENDED, but useful for experimental work.

4.2. Simple Delayed ACK implementation

Upon receipt of a packet without an SCE codepoint traditional delayed ACK processing is performed. Upon receipt of a packet with an SCE codepoint immediate ACK processing SHOULD be done, this allows some delaying of ACK's, but creates earlier feedback of the congested state. This has the negative effect of over signalling ESCE.

4.3. Dithered Delayed ACK implementation

Upon receipt of a packet the SCE codepoint is stored in the TCP state. Multiple packets state may be stored. Upon generation of an ACK, normal or delayed, the stored SCE state is used to set the state of ESCE. If no SCE state is in the TCP state, then the ESCE code point MUST NOT be set. If all of the packets to be ACKed have SCE state set then the ESCE code point MUST be set in the ACK. If some of the packets to be ACKed have SCE state set then some proportional number of ACK packets SHOULD be sent with the ESCE code point set. Though this may defer a ESCE congestion signal when there is not a next packet for some time it is generally accepted that such sparse flows are not the source of congestion and thus the delayed signal is of low impact. The goal is to have the same number of bytes marked with ESCE as arrived with SCE.

4.4. Advanced ACK implementation

The Advanced ACK implementation actually immediately flushes any pending ACK's up to the `_previous_` segment when the state of the SCE marking `_changes_`, allowing consecutive packets with the same SCE state to be coalesced by the normal delayed-ack logic. The ACK volume is then inflated only slightly compared to an unmarked connection, and may actually involve fewer acks than a connection involving CE marks or losses, during which delayed acks are temporarily disabled.

4.5. ACK Thinning

Ack thinning is something that has been considered, given that [cake] includes an optional ack-filter which does thinning. We have, for example, added consideration of the ESCE bit to Cake's ack-filter. Mathematically, the most extreme errors possible in either direction, due to ack thinning, are easily corrected during subsequent RTTs.

5. TCP Sender

The recommended response to each single segment marked with ESCE is to reduce cwnd by an amortised $1/\sqrt{\text{cwnd}}$ segments. If the growth rate is greater than that provided by the Reno-linear algorithm - eg. slow-start exponential or CUBIC polynomial - then the growth rate SHOULD also be reduced.

Other responses, such as the $1/\text{cwnd}$ from DCTCP, are also acceptable but may perform less well.

There are no changes to the response functions with respect to CE or packet loss specified by this draft, hence [RFC3168] and [RFC8511] are still applicable

This is still an area of continued investigation.

6. Related Work

6.1. More Accurate ECN Feedback in TCP [I-D.ietf-tcpm-accurate-ecn]

AccECN replaces the [RFC3168] definition of the ECE and CWR bits (and the former NS bit) with its own three-bit field. This new interpretation is predicated on successfully negotiating AccECN, and is not useful to SCE implementations because it provides no information about any ECT(1) codepoints received, and SCE does not need or use the extra information about CE marks that the three-bit field does provide. Hence SCE may be considered mutually exclusive with AccECN on any given connection.

AccECN supports a fallback to [RFC3168] style signalling during the three-way handshake by recognising the normal requests and responses of an [RFC3168] endpoint. SCE endpoints also exhibit [RFC3168] behaviour during the handshake, so this mutual exclusivity occurs naturally. There will therefore be no confusion on the wire between the two experiments, even though SCE does not explicitly negotiate its upgrade from plain [RFC3168] behaviour.

The latter is consistent with the (now historic) Nonce Sum specification, which also did not explicitly negotiate support, and used the same additional ECN codepoint and TCP header bit that SCE is now requesting.

7. IANA Considerations

This document requests one of the reserved bits in the TCP header, with the former TCP NS ("Nonce Sum") bit (bit 7) being suggested due to similarities with its previous usage. [RFC8311] (section 3) obsoletes the NS codepoint making it available for use.

8. Security Considerations

There are no Security considerations.

9. Acknowledgements

TBD

10. Normative References

- [I-D.morton-tsvwg-sce]
Morton, J. and R. Grimes, "The Some Congestion Experienced ECN Codepoint", draft-morton-tsvwg-sce-00 (work in progress), 2 July 2019, <<https://www.ietf.org/archive/id/draft-morton-tsvwg-sce-00>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [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>>.

11. Informative References

[cake] "Cake - Common Applications Kept Enhanced", November 2019,
<<http://www.bufferbloat.net/projects/codel/wiki/Cake>>.

[I-D.ietf-tcpm-accurate-ecn]
Briscoe, B., Kuehlewind, M., and R. Scheffenegger, "More
Accurate ECN Feedback in TCP", draft-ietf-tcpm-accurate-
ecn-09 (work in progress), 8 July 2019,
<[https://www.ietf.org/archive/id/draft-ietf-tcpm-accurate-
ecn-09](https://www.ietf.org/archive/id/draft-ietf-tcpm-accurate-ecn-09)>.

[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>>.

[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>>.

[sce-repo] "Some Congestion Experienced Reference Implementation
GitHub Repository", November 2019,
<<https://github.com/chromi/sce/>>.

Authors' Addresses

Rodney W. Grimes
Redacted
Portland, OR 97217
United States

Email: rgrimes@freebsd.org

Peter G. Heist
Redacted
463 11 Liberec 30
Czech Republic

Email: pete@heistp.net

TCP Maintenance & Minor Extensions (tcpm)
Internet-Draft
Intended status: Experimental
Expires: January 9, 2020

B. Briscoe
CableLabs
M. Kuehlewind
Ericsson
R. Scheffenegger
NetApp
July 8, 2019

More Accurate ECN Feedback in TCP
draft-ietf-tcpm-accurate-ecn-09

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on January 9, 2020.

Copyright Notice

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

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

Table of Contents

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

4.2. Compatibility with Other TCP Options and Experiments . . .	29
4.3. Compatibility with Feedback Integrity Mechanisms	29
5. Protocol Properties	30
6. IANA Considerations	32
7. Security Considerations	33
8. Acknowledgements	34
9. Comments Solicited	34
10. References	34
10.1. Normative References	34
10.2. Informative References	35
Appendix A. Example Algorithms	37
A.1. Example Algorithm to Encode/Decode the AcceCN Option . .	37
A.2. Example Algorithm for Safety Against Long Sequences of ACK Loss	38
A.2.1. Safety Algorithm without the AcceCN Option	38
A.2.2. Safety Algorithm with the AcceCN Option	40
A.3. Example Algorithm to Estimate Marked Bytes from Marked Packets	41
A.4. Example Algorithm to Beacon AcceCN Options	42
A.5. Example Algorithm to Count Not-ECT Bytes	43
Appendix B. Rationale for Usage of TCP Header Flags	43
B.1. Three TCP Header Flags in the SYN-SYN/ACK Handshake . . .	43
B.2. Four Codepoints in the SYN/ACK	44
B.3. Space for Future Evolution	45
Authors' Addresses	46

1. Introduction

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

This documents specifies an experimental scheme for ECN feedback in the TCP header to provide more than one feedback signal per RTT. It will be called the more accurate ECN feedback scheme, or AcceCN for short. If AcceCN progresses from experimental to the standards

track, it is intended to be a complete replacement for classic TCP/ECN feedback, not a fork in the design of TCP. AccECN feedback complements TCP's loss feedback and it supplements classic TCP/ECN feedback, so its applicability is intended to include all public and private IP networks (and even any non-IP networks over which TCP is used today), whether or not any nodes on the path support ECN of whatever flavour.

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

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

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

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

1.1. Document Roadmap

The following introductory sections outline the goals of AccECN (Section 1.2) and the goal of experiments with ECN (Section 1.3) so that it is clear what success would look like. Then terminology is defined (Section 1.4) and a recap of existing prerequisite technology is given (Section 1.5).

Section 2 gives an informative overview of the AccECN protocol. Then Section 3 gives the normative protocol specification. Section 4 assesses the interaction of AccECN with commonly used variants of TCP, whether standardised or not. Section 5 summarises the features and properties of AccECN.

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

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

1.2. Goals

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

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

1.3. Experiment Goals

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

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

scale path traversal in the Internet is needed before finalizing this specification on the Standards Track.

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

1.4. Terminology

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

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

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

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

Pure ACK: A TCP acknowledgement without a data payload.

TCP client: The TCP stack that originates a connection.

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

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

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

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

1.5. Recap of Existing ECN feedback in IP/TCP

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

IP-ECN codepoint (binary)	Codepoint name	Description
00	Not-ECT	Not ECN-Capable Transport
01	ECT(1)	ECN-Capable Transport (1)
10	ECT(0)	ECN-Capable Transport (0)
11	CE	Congestion Experienced

Table 1: The ECN Field in the IP Header

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

The last bit in byte 13 of the TCP header was defined as the Nonce Sum (NS) for the ECN Nonce [RFC3540]. In the absence of widespread deployment RFC 3540 has been reclassified as historic [RFC8311] and

the respective flag has been marked as "reserved", making this TCP flag available for use by the AccECN experiment instead.

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

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

2. AccECN Protocol Overview and Rationale

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

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

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

- o an essential part that re-uses ECN TCP header bits to feed back the number of arriving CE marked packets. This provides more accuracy than classic ECN feedback, but limited resilience against ACK loss;
- o a supplementary part using a new AccECN TCP Option that provides additional feedback on the number of bytes that arrive marked with each of the three ECN codepoints (not just CE marks). This provides greater resilience against ACK loss than the essential feedback, but it is more likely to suffer from middlebox interference.

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

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

- o this efficiently reuses scarce TCP header space, given TCP option space is approaching saturation;

- o a single upgrade path for the TCP protocol is preferable to a fork in the design;
- o otherwise classic and accurate ECN feedback could give conflicting feedback on the same segment, which could open up new security concerns and make implementations unnecessarily complex;
- o middleboxes are more likely to faithfully forward the TCP ECN flags than newly defined areas of the TCP header.

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

2.1. Capability Negotiation

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

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

2.2. Feedback Mechanism

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

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

Each ACK carries the three least significant bits (LSBs) of the packet-based CE counter using the ECN bits in the TCP header, now

renamed the Accurate ECN (ACE) field (see Figure 2 later). The LSBs of each of the three byte counters are carried in the AccECN Option.

2.3. Delayed ACKs and Resilience Against ACK Loss

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

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

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

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

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

2.4. Feedback Metrics

The CE packet counter in the ACE field and the CE byte counter in the AccECN Option both provide feedback on received CE-marks. The CE packet counter includes control packets that do not have payload data, while the CE byte counter solely includes marked payload bytes. If both are present, the byte counter in the option will provide the more accurate information needed for modern congestion control and policing schemes, such as L4S, DCTCP or ConEx. If the option is stripped, a simple algorithm to estimate the number of marked bytes from the ACE field is given in Appendix A.3.

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

2.5. Generic (Dumb) Reflector

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

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

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

Even if the TCP client (or server) has set the SYN (or SYN/ACK) to not-ECT in compliance with RFC 3168, feedback on the state of the ECN field when it arrives at the receiver could still be useful, because middleboxes have been known to overwrite the ECN IP field as if it is still part of the old Type of Service (ToS) field [Mandalari18]. If

a TCP client has set the SYN to Not-ECT, but receives feedback that the ECN field on the SYN arrived with a different codepoint, it can detect such middlebox interference and send Not-ECT for the rest of the connection (see [I-D.kuehlewind-tcpm-ecn-fallback]). Today, if a TCP server receives ECT or CE on a SYN, it cannot know whether it is invalid (or valid) because only the TCP client knows whether it originally marked the SYN as Not-ECT (or ECT). Therefore, prior to AccECN, the server's only safe course of action was to disable ECN for the connection. Instead, the AccECN protocol allows the server to feed back the received ECN field to the client, which then has all the information to decide whether the connection has to fall-back from supporting ECN (or not).

3. AccECN Protocol Specification

3.1. Negotiating to use AccECN

3.1.1. Negotiation during the TCP handshake

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

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

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

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

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

The procedure for the client to follow if a SYN/ACK does not arrive before its retransmission timer expires is given in Section 3.1.3.

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

AccECN: More Accurate ECN Feedback (the present specification)

Nonce: ECN Nonce feedback [RFC3540]

ECN: 'Classic' ECN feedback [RFC3168]

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

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

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

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

1. The top block shows the case already described where both endpoints support AccECN and how the TCP server (B) indicates congestion feedback.

2. The second block shows the cases where the TCP client (A) supports AccECN but the TCP server (B) supports some earlier variant of TCP feedback, indicated in its SYN/ACK. Therefore, as soon as an AccECN-capable TCP client (A) receives the SYN/ACK shown it MUST set both its half connections into the feedback mode shown in the rightmost column.
3. The third block shows the cases where the TCP server (B) supports AccECN but the TCP client (A) supports some earlier variant of TCP feedback, indicated in its SYN. Therefore, as soon as an AccECN-enabled TCP server (B) receives the SYN shown, it MUST set both its half connections into the feedback mode shown in the rightmost column.
4. The fourth block displays a combination labelled 'Broken' . Some older TCP server implementations incorrectly set the reserved flags in the SYN/ACK by reflecting those in the SYN. Such broken TCP servers (B) cannot support ECN, so as soon as an AccECN-capable TCP client (A) receives such a broken SYN/ACK it MUST fall-back to Not ECN mode for both its half connections.

The following exceptional cases need some explanation:

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

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

3.1.2. Forward Compatibility

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

For the avoidance of doubt, the negotiation tabulated in Table 2 solely concerns the three TCP header flags shown (AE, CWR and ECE).

An AccECN host (client or server) MUST ignore the three remaining reserved TCP header flags on all packets.

3.1.3. Retransmission of the SYN

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

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

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

3.2. AccECN Feedback

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

stateless handshake (see Section 4.1) and to be distinct from cases where the fields are incorrectly zeroed (e.g. by middleboxes – see Section 3.2.7.4).

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

3.2.1. Initialization of Feedback Counters at the Data Sender

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

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

3.2.2. The ACE Field

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

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

Figure 2: Definition of the ACE field within bytes 13 and 14 of the TCP Header (when AcceECN 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 AcceECN connection has been established.

A host MUST interpret the AE, CWR and ECE flags as the 3-bit ACE counter on a segment with the SYN flag cleared (SYN=0) that it sends or receives if both of its half-connections are set into AccECN mode having successfully negotiated AccECN (see Section 3.1). A host MUST NOT interpret the 3 flags as a 3-bit ACE field on any segment with SYN=1 (whether ACK is 0 or 1), or if AccECN negotiation is incomplete or has not succeeded.

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

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

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

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

Table 3: Meaning of the ACE field on the ACK of the SYN/ACK

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

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

{Note 3}: In the case where a server that implements AccECN is also using a stateless handshake (termed a SYN cookie) it will not remember whether it entered AccECN mode. The values 0b000 or 0b001 will remind it that it did not enter AccECN mode, because AccECN does not use them (see Section 4.1 for details).

If a stateless server that implements AccECN receives either of these two values in the ACK, its action is implementation-dependent and outside the scope of this spec, It will certainly not take the action in the third column because, after it receives either of these values, it is not in AccECN mode. I.e., it will not disable ECN (at least not just because ACE is 0b000) and it will not set s.cep.

3.2.3. Testing for Zeroing of the ACE Field

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

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

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

Note that the Data Sender MUST NOT test whether the arriving counter in the initial ACE field has been initialized to a specific valid

value - the above check solely tests whether the ACE fields have been incorrectly zeroed. This allows hosts to use different initial values as an additional signalling channel in future.

3.2.4. Testing for Mangling of the IP/ECN Field

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

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

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

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

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

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

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

mobile networks [Mandalari18]. Then one end might see no unsafe transition and continue sending ECN-capable packets, while the other end sees an unsafe transition and stops sending ECN-capable packets.

3.2.5. Safety against Ambiguity of the ACE Field

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

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

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

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

3.2.6. The AccECN Option

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

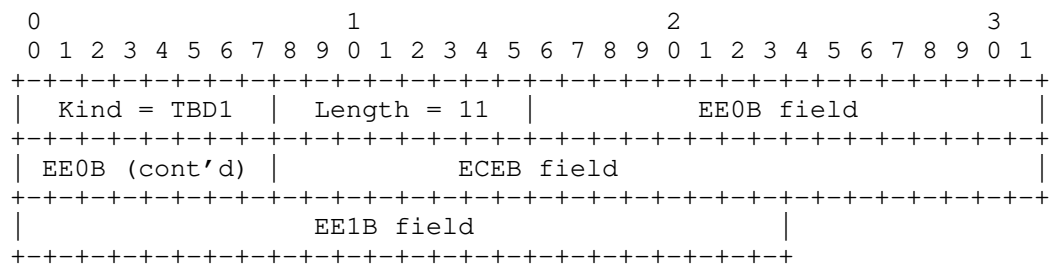


Figure 3: The AccECN Option

When a Data Receiver sends an AcceCN Option, it MUST set the Kind field to TBD1, which is registered in Section 6 as a new TCP option Kind called AcceCN. An experimental TCP option with Kind=254 MAY be used for initial experiments, with magic number 0xACCE.

Appendix A.1 gives an example algorithm for the Data Receiver to encode its byte counters into the AcceCN Option, and for the Data Sender to decode the AcceCN Option fields into its byte counters.

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

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

Length=11: EE0B, ECEB, EE1B

Length=8: EE0B, ECEB

Length=5: EE0B

Length=2: (empty)

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

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

The AcceCN Option has to be optional to implement, because both sender and receiver have to be able to cope without the option anyway – in cases where it does not traverse a network path. It is RECOMMENDED to implement both sending and receiving of the AcceCN Option. If sending of the AcceCN Option is implemented, the fallbacks described in this document will need to be implemented as well (unless solely for a controlled environment where path traversal is not considered a problem). Even if a developer does not implement

sending of the AcceECN Option, it is RECOMMENDED that they still implement logic to receive and understand any AcceECN Options sent by remote peers.

If a Data Receiver intends to send the AcceECN Option at any time during the rest of the connection it is strongly recommended to also test path traversal of the AcceECN Option as specified in the next section.

3.2.7. Path Traversal of the AcceECN Option

3.2.7.1. Testing the AcceECN Option during the Handshake

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

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

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

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

3.2.7.2. Testing for Loss of Packets Carrying the AcceECN Option

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

falling back to classic ECN feedback rather than non-ECN on the third retransmission).

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

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

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

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

3.2.7.3. Testing for Absence of the AccECN Option

If the TCP client has successfully negotiated AccECN but does not receive an AccECN Option on the SYN/ACK (e.g. because it has been stripped by a middlebox or not sent by the server), the client switches into a mode that assumes that the AccECN Option is not available for this half connection.

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

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

(unless it has established that sending the AcceCN Option is causing packets to be blocked as in Section 3.2.7.2).

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

3.2.7.4. Test for Zeroing of the AcceCN Option

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

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

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

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

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

3.2.7.5. Consistency between AccECN Feedback Fields

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

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

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

3.2.8. Usage of the AccECN TCP Option

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

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

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

disables change-triggered ACKs, whether partially like this or otherwise, the operator will also be responsible for ensuring a co-ordinated sender algorithm is deployed;

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

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

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

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

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

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

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

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

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

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

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

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

The ACE field changes with every received CE marking, so today's receive offloading could lead to many interrupts in high congestion

situations. Although that would be useful (because congestion information is received sooner), it could also significantly increase processor load, particularly in scenarios such as DCTCP or L4S where the marking rate is generally higher.

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

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

4. Interaction with Other TCP Variants

This section is informative, not normative.

4.1. Compatibility with SYN Cookies

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

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

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

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

If the pure ACK that acknowledges a SYN cookie contains an ACE field with the value 0b000 or 0b001, these values indicate that the client

did not request support for AccECN and therefore the server does not enter AccECN mode for this connection. Further, 0b001 on the ACK implies that the server sent an ECN-capable SYN/ACK, which was marked CE in the network, and the non-AccECN client fed this back by setting ECE on the ACK of the SYN/ACK.

4.2. Compatibility with Other TCP Options and Experiments

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

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

4.3. Compatibility with Feedback Integrity Mechanisms

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

- o The Data Sender can test the integrity of the receiver's ECN (or loss) feedback by occasionally setting the IP-ECN field to a value normally only set by the network (and/or deliberately leaving a sequence number gap). Then it can test whether the Data Receiver's feedback faithfully reports what it expects (similar to para 2 of Section 20.2 of [RFC3168]). Unlike the ECN Nonce [RFC3540], this approach does not waste the ECT(1) codepoint in the IP header, it does not require standardisation and it does not rely on misbehaving receivers volunteering to reveal feedback information that allows them to be detected. However, setting the CE mark by the sender might conceal actual congestion feedback from the network and should therefore only be done sparsely.
- o Networks generate congestion signals when they are becoming congested, so networks are more likely than Data Senders to be concerned about the integrity of the receiver's feedback of these signals. A network can enforce a congestion response to its ECN markings (or packet losses) using congestion exposure (ConEx) audit [RFC7713]. Whether the receiver or a downstream network is suppressing congestion feedback or the sender is unresponsive to the feedback, or both, ConEx audit can neutralise any advantage that any of these three parties would otherwise gain.

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

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

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

5. Protocol Properties

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

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

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

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

Timeliness: While the same ECN markings are arriving continually at the Data Receiver, it can defer ACKs as TCP does normally, but it

will immediately send an ACK as soon as a different ECN marking arrives.

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

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

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

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

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

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

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

Backward Compatibility: If only one endpoint supports the AccECN scheme, it will fall-back to the most advanced ECN feedback scheme supported by the other end.

Backward Compatibility: If the AccECN Option is stripped by a middlebox, AccECN still provides basic congestion feedback in the ACE field. Further, AccECN can be used to detect mangling of the IP ECN field; mangling of the TCP ECN flags; blocking of ECT-marked segments; and blocking of segments carrying the AccECN Option. It can detect these conditions during TCP's 3WSH so that it can fall back to operation without ECN and/or operation without the AccECN Option.

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

6. IANA Considerations

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

Bit	Name	Reference
7	AE (Accurate ECN)	RFC XXXX

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

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

Kind	Length	Meaning	Reference
TBD1	N	Accurate ECN (AccECN)	RFC XXXX

[TO BE REMOVED: This registration should take place at the following location: <http://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml#tcp-parameters-1>]

Early implementation before the IANA allocation MUST follow [RFC6994] and use experimental option 254 and magic number 0xACCE (16 bits), then migrate to the new option after the allocation.

7. Security Considerations

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

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

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

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

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

8. Acknowledgements

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

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

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

9. Comments Solicited

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

10. References

10.1. Normative References

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

- [RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, DOI 10.17487/RFC6994, August 2013, <<https://www.rfc-editor.org/info/rfc6994>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

10.2. Informative References

- [I-D.ietf-tcpm-generalized-ecn]
Bagnulo, M. and B. Briscoe, "ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control Packets", draft-ietf-tcpm-generalized-ecn-03 (work in progress), October 2018.
- [I-D.ietf-tsvwg-l4s-arch]
Briscoe, B., Schepper, K., and M. Bagnulo, "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture", draft-ietf-tsvwg-l4s-arch-03 (work in progress), October 2018.
- [I-D.kuehlewind-tcpm-ecn-fallback]
Kuehlewind, M. and B. Trammell, "A Mechanism for ECN Path Probing and Fallback", draft-kuehlewind-tcpm-ecn-fallback-01 (work in progress), September 2013.
- [Mandalari18]
Mandalari, A., Lutu, A., Briscoe, B., Bagnulo, M., and Oe. Alay, "Measuring ECN++: Good News for ++, Bad News for ECN over Mobile", IEEE Communications Magazine , March 2018.
- [RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, DOI 10.17487/RFC3540, June 2003, <<https://www.rfc-editor.org/info/rfc3540>>.
- [RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <<https://www.rfc-editor.org/info/rfc4987>>.
- [RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", RFC 5562, DOI 10.17487/RFC5562, June 2009, <<https://www.rfc-editor.org/info/rfc5562>>.

- [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <<https://www.rfc-editor.org/info/rfc5925>>.
- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <<https://www.rfc-editor.org/info/rfc6824>>.
- [RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <<https://www.rfc-editor.org/info/rfc7413>>.
- [RFC7560] Kuehlewind, M., Ed., Scheffenegger, R., and B. Briscoe, "Problem Statement and Requirements for Increased Accuracy in Explicit Congestion Notification (ECN) Feedback", RFC 7560, DOI 10.17487/RFC7560, August 2015, <<https://www.rfc-editor.org/info/rfc7560>>.
- [RFC7713] Mathis, M. and B. Briscoe, "Congestion Exposure (ConEx) Concepts, Abstract Mechanism, and Requirements", RFC 7713, DOI 10.17487/RFC7713, December 2015, <<https://www.rfc-editor.org/info/rfc7713>>.
- [RFC8257] Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <<https://www.rfc-editor.org/info/rfc8257>>.
- [RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <<https://www.rfc-editor.org/info/rfc8311>>.
- [RFC8511] Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst, "TCP Alternative Backoff with ECN (ABE)", RFC 8511, DOI 10.17487/RFC8511, December 2018, <<https://www.rfc-editor.org/info/rfc8511>>.

Appendix A. Example Algorithms

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

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

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

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

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

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

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

where `'%'` is the modulo operator.

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

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

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

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

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

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

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

A.2.1. Safety Algorithm without the AccECN Option

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

$$\text{DIVACE} = 2^3$$

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

$$\text{ACE} = \text{r.cep} \% \text{DIVACE}.$$

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

d.cep between the ACE field and its local s.cep counter, using modulo arithmetic as follows:

```
if ((newlyAcedB > 0) || (newlyAcedB == 0 && newlyAcedT > 0))
    d.cep = (ACE + DIVACE - (s.cep % DIVACE)) % DIVACE
```

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

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

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

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

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

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

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

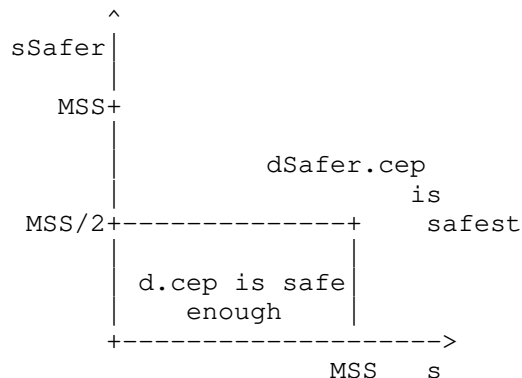
If missing acknowledgement numbers arrive later (due to reordering), Section 3.2.5 says "the Data Sender MAY attempt to neutralise the effect of any action it took based on a conservative assumption that it later found to be incorrect". To do this, the Data Sender would have to store the values of all 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 AcceECN Option

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

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

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



The following examples give the reasoning behind the algorithm, assuming $MSS=1,460$ [B]:

- o if $d.cep=0$, $dSafer.cep=8$ and $d.ceb=1,460$, then $s=infinity$ and $sSafer=182.5$.
Therefore even though the average size of 8 data segments is unlikely to have been as small as $MSS/8$, $d.cep$ cannot have been correct, because it would imply an average segment size greater than the MSS .
- o if $d.cep=2$, $dSafer.cep=10$ and $d.ceb=1,460$, then $s=730$ and $sSafer=146$.
Therefore $d.cep$ is safe enough, because the average size of 10 data segments is unlikely to have been as small as $MSS/10$.
- o if $d.cep=7$, $dSafer.cep=15$ and $d.ceb=10,200$, then $s=1,457$ and $sSafer=680$.
Therefore $d.cep$ is safe enough, because the average data segment size is more likely to have been just less than one MSS , rather than below $MSS/2$.

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

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

If the AccECN Option is not available, the Data Sender can only decode CE-marking from the ACE field in packets. Every time an ACK arrives, to convert this into an estimate of CE-marked bytes, it needs an average of the segment size, s_{ave} . Then it can add or subtract s_{ave} from the value of $d.ceb$ as the value of $d.cep$ increments or decrements.

To calculate `s_ave`, it could keep a record of the byte numbers of all the boundaries between packets in flight (including control packets), and recalculate `s_ave` on every ACK. However it would be simpler to merely maintain a counter `packets_in_flight` for the number of packets in flight (including control packets), which it could update once per RTT. Either way, it would estimate `s_ave` as:

```
s_ave ~= flightsize / packets_in_flight,
```

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

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

```
s_ave = a * s + (1-a) * s_ave,
```

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

A.4. Example Algorithm to Beacon AccECN Options

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

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

For optimised integer arithmetic, `BEACON_FREQ = 4` could be used, rather than 3, so that the division could be implemented as an integer right bit-shift by `lg(BEACON_FREQ)`.

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

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

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

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

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

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

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

Appendix B. Rationale for Usage of TCP Header Flags

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

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

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

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

The third flag was used in a way that could be distinguished from the ECN nonce, in case any nonce deployment was encountered. Previous usage of this flag for the ECN nonce was integrated into the original ECN negotiation. This further justified the 3rd flag's use for AcceECN, 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, AcceECN overloads all three bits again as a 3-bit counter.

B.2. Four Codepoints in the SYN/ACK

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

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

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

B.3. Space for Future Evolution

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

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

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

Future non-AcceECN variants: Five codepoints out of the 8 possible in the 3 TCP header flags used by AcceECN are unused on the initial SYN (in the order AE,CWR,ECE): 001, 010, 100, 101, 110. Section 3.1.2 ensures that the installed base of AcceECN servers

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

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

Authors' Addresses

Bob Briscoe
CableLabs
UK

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

Mirja Kuehlewind
Ericsson
Germany

EMail: ietf@kuehlewind.net

Richard Scheffenegger
NetApp
Vienna
Austria

EMail: Richard.Scheffenegger@netapp.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 21, 2020

M. Kuehlewind
Ericsson
November 18, 2019

Registration Policy for TCP Header Flags
draft-kuehlewind-tcpm-flags-registry-00

Abstract

RFC2780 specifies the registration policy for reserved TCP header flags as Standards Action. RFC3168 created an IANA registry for these header flags and registered bit 8 (CWR) and 9 (ECE). This draft changes the registration policy of the registration to IETF Review as usually new TCP mechanisms that could use the remaining reserved flags will be first specified as experimental. Not noting any of those experiments in the registry would undermine the purpose of having a registry. However, care must be taken, as only a few reserved flags are left and if a new (experimental) mechanism sees deployment in the Internet, the flag cannot be unassigned anymore or used for something else.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 21, 2020.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	2
2. Notation	3
3. Registration Policy for TCP Header Flags	3
4. IANA Considerations	4
5. Security Consideration	4
6. Contributors	4
7. Acknowledgments	4
8. References	4
8.1. Normative References	4
8.2. Informative References	5
Author's Address	5

1. Introduction

[RFC2780] specifies the registration policy for reserved TCP header flags as Standards Action. In section 9.2 (Reserved Bits in TCP Header) it says:

"The reserved bits in the TCP header are assigned following a Standards Action process."

Earlier on, in section 2 (Temporary Assignments) it also says

"From time to time temporary assignments are made in the values for fields in these headers for use in experiments. IESG Approval is required for any such temporary assignments."

However, it does not specify what exactly is meant with a temporary assignment and how this could work for TCP header given they can hardly be re-purposed once deployed on the Internet.

[RFC3168] created a IANA registry for these header flags and registered bit 8 (CWR) and 9 (ECE). [RFC3540] assigned bit 7 to the experimental ECN Nonce extension and [RFC8311] recently declared RFC3540 as historic and changed the assignment to reserved, as ECN Nonce was not deployed on the Internet. The purpose of RFC8311 was to concentrate updates to Standard Track RFCs in one document in order to enable new experimentation with various ECN-related mechanisms. However, RFC8311 does not provide any recommendation of the use of bit 7 and the TCP flags registry.

2. Notation

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

3. Registration Policy for TCP Header Flags

This draft changes the registration policy of the registration to IETF Review as usually new TCP mechanisms that could use the remaining reserved flags will be first specified as experimental. Not noting any of those experiments in the registry would undermine the purpose of having a registry. However, assignments based on experimental RFC should be marked as such in the "Comments" field and potentially even provide hints about the nature of the experiment or provide a pointer to a section in an RFC where the experiment is explained.

Further, care must be taken, when assigning TCP header flags for experimental use, as a) only a few reserved flags are left, and b) if a new (experimental) mechanism sees deployment in the Internet, the flag cannot be unassigned anymore or used for something else. Therefore, any experimental RFC that registers a reserved flag in the TCP flags registry MUST provide ways to alter the proposed mechanism at the end of the experimental phase without using additional TCP header flags. E.g. it would be possible to add an additional negotiation mechanism after the TCP handshake that would make it possible to use different versions of the general mechanism/extension that was negotiated or indicated during the TCP handshake using the newly assigned flag. Further, any experimental extension SHOULD discuss the scope of the experiment and potential failure cases or open questions that need to be answered when running the experiment and explain how these could be addressed in an updated version.

TCP flags can only be de-assigned if no deployment of an experimental extension happened. This should be evaluated some years after the assignment to an experimental extension, in order to change the registry entry back to "RESERVED" and move the respective experimental RFC to history, or assign it permanently. This might be done by IESG Approval or based on an Standards track document. However, even when reversed to "RESERVED", the experiment should still be noted (as failed and over) in the "Comments" field of the registry entry.

4. IANA Considerations

IANA is requested to change the registration policy of the "TCP Header Flags" registry (<https://www.iana.org/assignments/tcp-header-flags/tcp-header-flags.xhtml>) to IETF Review or IESG Approval and note this accordingly on the registry page.

In addition, the registry should be updated with a new column called "Comments". The text in the "name" field of the entry for bit 7 there should be moved into this new column while the name will then only say "RESERVED". Further, bits 4, 5, 6 should be added to the registry and marked as "RESERVED".

Moreover, as a matter of cleaning-up, IANA is requested to move the registry to a sub-registry on the Transmission Control Protocol (TCP) Parameters page (<https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml>).

5. Security Consideration

TBD

6. Contributors

7. Acknowledgments

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2780] Bradner, S. and V. Paxson, "IANA Allocation Guidelines For Values In the Internet Protocol and Related Headers", BCP 37, RFC 2780, DOI 10.17487/RFC2780, March 2000, <<https://www.rfc-editor.org/info/rfc2780>>.
- [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>>.
- [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>>.

8.2. Informative References

- [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>>.
- [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>>.

Author's Address

Mirja Kuehlewind
Ericsson

Email: mirja.kuehlewind@ericsson.com

TCPM
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2020

M. Scharf
Hochschule Esslingen
V. Murgai
Cisco Systems Inc
M. Jethanandani
VMware
November 04, 2019

YANG Model for Transmission Control Protocol (TCP) Configuration
draft-scharf-tcpm-yang-tcp-03

Abstract

This document specifies a YANG model for TCP on devices that are configured by network management protocols. The YANG model defines groupings for fundamental parameters that can be modified in many TCP implementations. The model includes definitions from YANG Groupings for TCP Client and TCP Servers (I-D.ietf-netconf-tcp-client-server). The model is NMDA (RFC 8342) compliant.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 7, 2020.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	2
2. Requirements Language	3
3. Model Overview	3
3.1. Modeling Scope	3
3.2. Basic TCP Configuration Parameters	5
3.3. Model Design	7
3.4. Tree Diagram	8
3.5. Example Usage	8
4. TCP Configuration YANG Model	8
5. IANA Considerations	14
5.1. The IETF XML Registry	14
5.2. The YANG Module Names Registry	15
6. Security Considerations	15
7. References	15
7.1. Normative References	15
7.2. Informative References	17
Appendix A. Acknowledgements	18
Appendix B. Changes compared to previous versions	18
Authors' Addresses	19

1. Introduction

The Transmission Control Protocol (TCP) [RFC0793] is used by many applications in the Internet, including control and management protocols. Therefore, TCP is implemented on network elements that can be configured via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. This document specifies a YANG [RFC7950] 1.1 model for configuring TCP on network elements that support YANG data models, and is Network Management Datastore Architecture (NMDA) [RFC8342] compliant. This document includes definitions from YANG Groupings for TCP Clients and TCP Servers [I-D.ietf-netconf-tcp-client-server]. The model focuses on fundamental and standard TCP functions that are widely implemented. The model can be augmented to address more advanced or implementation-specific TCP features.

Many protocol stacks on internet hosts use other methods to configure TCP, such as operating system configuration or policies. Many TCP/IP stacks cannot be configured by network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040] and they do not use YANG data models. Yet, such TCP implementations often also have means to

configure the parameters listed in this document. All parameters defined in this document are optional.

This specification is orthogonal to Management Information Base (MIB) for the Transmission Control Protocol (TCP) [RFC4022]. The TCP Extended Statistics MIB [RFC4898] is also available, and there are MIBs for UDP Management Information Base for the User Datagram Protocol (UDP) [RFC4113] and Stream Control Transmission Protocol (SCTP) Management Information Base (MIB) [RFC3873]. It is possible to translate a MIB into a YANG model, for instance using Translation of Structure of Management Information Version 2 (SMIV2) MIB Modules to YANG Modules [RFC6643]. However, this approach is not used in this document, as such a translated model would not be up-to-date.

There are also other related YANG models. Examples are:

- o Application protocol models may include TCP parameters, for example in case of BGP YANG Model for Service Provider Networks [I-D.ietf-idr-bgp-model].
- o TCP header attributes are modeled in other models, such as YANG Data Model for Network Access Control Lists (ACLs) [RFC8519] and Distributed Denial-of-Service Open Thread Signaling (DOTS) Data Channel Specification [I-D.ietf-dots-data-channel].
- o TCP-related configuration of a NAT (e.g., NAT44, NAT64, Destination NAT, ...) is defined in A YANG Module for Network Address Translation (NAT) and Network Prefix Translation (NPT) [RFC8512] and A YANG Data Model for Dual-Stack Lite (DS-Lite) [RFC8513].

2. Requirements Language

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

3. Model Overview

3.1. Modeling Scope

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

- o Global configuration: Many TCP implementations have configuration parameters that affect all TCP connections. Typical examples include enabling or disabling optional protocol features.
- o Interface configuration: It can be useful to use different TCP parameters on different interfaces, e.g., different device ports or IP interfaces. In that case, TCP parameters can be part of the interface configuration. Typical examples are the Maximum Segment Size (MSS) or configuration related to hardware offloading.
- o Connection parameters: Many implementations have means to influence the behavior of each TCP connection, e.g., on the programming interface used by applications. A typical example are socket options in the socket API, such as disabling the Nagle algorithm by TCP_NODELAY. If an application uses such an interface, it is possible that the configuration of the application or application protocol includes TCP-related parameters. An example is the BGP YANG Model for Service Provider Networks [I-D.ietf-idr-bgp-model].
- o Policies: Setting of TCP parameters can also be part of system policies, templates, or profiles. An example would be the preferences defined in the TAPS interface An Abstract Application Layer Interface to Transport Services [I-D.ietf-taps-interface].

There is no ground truth for setting certain TCP parameters, and traditionally different implementation have used different modeling approaches. For instance, one implementation may define a given configuration parameter globally, while another one uses per-interface settings, and both approaches work well for the corresponding use cases. Also, different systems may use different default values.

The YANG model defined in this document includes definitions from the YANG Groupings for TCP Clients and TCP Servers [I-D.ietf-netconf-tcp-client-server]. Similar to the base model, this specification defines YANG groupings. This allows reuse of these groupings in different YANG data models. It is intended that these groupings will be used either standalone or for TCP-based protocols as part of a stack of protocol-specific configuration models.

In addition to configuration of the TCP protocol engine, a TCP implementation typically also offers access to operational state and statistics. This includes amongst others:

- o Statistics: Counters for the number of active/passive opens, sent and received segments, errors, and possibly other detailed debugging information
- o TCP connection table: Access to status information for all TCP connections
- o TCP listener table: Information about all TCP listening endpoints

Similar to the TCP MIB [RFC4022], this document also specifies a TCP connection table.

TODO: A future version of this document may include statistics equivalent to the TCP MIB [RFC4022]:

- o active-opens
- o passive-opens
- o attempt-fails
- o establish-resets
- o currently-established
- o in-segments
- o out-segments
- o retransmitted-segments
- o in-errors
- o out-resets

3.2. Basic TCP Configuration Parameters

There are a number of basic system parameters that are configurable on many TCP implementations, even if not all TCP implementations may indeed have exactly all these settings. Also, the syntax, semantics and scope (e.g., global or interface-specific) can be different in different system architectures.

The following list of fundamental parameters considers both TCP implementations on hosts and on routers:

- o Keepalives (see also [I-D.ietf-netconf-tcp-client-server])

- * Idle-time (in seconds): integer
- * Probe-interval (in seconds): integer
- * Max-probes: integer
- o Maximum MSS (in byte): integer
- o FIN timeout (in seconds): integer
- o SACK (disable/enable): boolean
- o Timestamps (disable/enable): boolean
- o Path MTU Discovery (disable/enable): boolean
- o ECN
- * Enabling (disable/passive/active): enumeration

TCP-AO is increasingly supported on routers and also requires configuration.

Some other parameters are also common but not ubiquitously supported, or modeled in very different ways:

- o Delayed ACK timeout (in ms)
- o Initial RTO value (in ms)
- o Maximum number of retransmissions
- o Window scaling
- o Maximum number of connections

TCP can be implemented in different ways and design choices by the protocol engine often affect configuration options. In a number of areas there are major differences between different software architectures. As a result, there are not many commonalities in the corresponding configuration parameters:

- o Window size: TCP stacks can either store window state variables (such as the congestion window) in segments or in bytes.
- o Buffer sizes: The memory management depends on the operating system. As the size of buffers can vary over several orders of

magnitude, very different implementations exist. This typically influences TCP flow control.

- o Timers: Timer implementation is another area in which TCP stacks may differ.
- o Congestion control algorithms: Many congestion control algorithms have configuration parameters, but except for fundamental properties they often tie into the specific implementation.

This document only models fundamental system parameters that are configurable on many TCP implementations, and for which the configuration is reasonably similar.

3.3. Model Design

[[Editor's node: This section requires further work.]]

This document extends the YANG model "ietf-tcp-common" defined in [I-D.ietf-netconf-tcp-client-server]. The intention is to define YANG groupings for TCP parameters so that they can be used in different YANG models.

As an example for the configuration of SACK, a YANG model could import the YANG model "ietf-tcp-common" as well as the model defined in this document as follows:

```
module example-tcp {
  namespace "http://example.com/tcp";
  prefix tcp;

  import ietf-tcp {
    prefix tcp;
  }

  import ietf-tcp-common {
    prefix tcpcmn;
  }

  container example-tcp-config {
    description "Example TCP stack configuration";

    uses tcpcmn:tcp-common-grouping;
    uses tcp:tcp-sack-grouping;
  }
}
```

3.4. Tree Diagram

This section provides a abridged tree diagram for the YANG module defined in this document. Annotations used in the diagram are defined in YANG Tree Diagrams [RFC8340].

```
module: ietf-tcp
  +--rw tcp!
    +--rw connections
      ...
```

3.5. Example Usage

[[Editor's note: This section is TBD.]]

4. TCP Configuration YANG Model

Editor's note: How to use ietf-tcp-common as basis? For instance, is the tcp-system-grouping therein needed?

<CODE BEGINS> file "ietf-tcp@2019-11-04.yang"

```
module ietf-tcp {
  yang-version "1.1";
  namespace "urn:ietf:params:xml:ns:yang:ietf-tcp";
  prefix "tcp";

  import ietf-tcp-client {
    prefix "tcpc";
  }
  import ietf-tcp-server {
    prefix "tcps";
  }
  import ietf-tcp-common {
    prefix "tcpcmn";
  }
  import ietf-inet-types {
    prefix "inet";
  }

  organization
    "IETF TCPM Working Group";

  contact
    "WG Web:  <http://tools.ietf.org/wg/tcpm>
    WG List:  <tcpm@ietf.org>

    Authors: Michael Scharf (michael.scharf at hs-esslingen dot de)
```


Vishal Murgai (vmurgai at cisco dot com)
Mahesh Jethanandani (mjethanandani at gmail dot com)";

```
description
  "This module focuses on fundamental and standard TCP functions
  that widely implemented. The model can be augmented to address
  more advanced or implementation specific TCP features.";

revision "2019-11-04" {
  description
    "Initial Version";
  reference
    "RFC XXX, TCP Configuration.";
}

// Features
feature server {
  description
    "TCP Server configuration supported.";
}

feature client {
  description
    "TCP Client configuration supported.";
}

// TCP-AO Groupings

grouping mkt {
  leaf options {
    type binary;
    description
      "This flag indicates whether TCP options other than TCP-AO
      are included in the MAC calculation. When options are
      included, the content of all options, in the order present,
      is included in the MAC, with TCP-AO's MAC field zeroed out.
      When the options are not included, all options other than
      TCP-AO are excluded from all MAC calculations (skipped over,
      not zeroed).

      Note that TCP-AO, with its MAC field zeroed out, is always
      included in the MAC calculation, regardless of the setting
      of this flag; this protects the indication of the MAC length
      as well as the key ID fields (KeyID, RNextKeyID). The option
      flag applies to TCP options in both directions
      (incoming and outgoing segments).";
    reference
      "RFC 5925: The TCP Authentication Option.";
```

```
    }

    leaf key-id {
        type uint8;
        description
            "TBD";
    }

    leaf rnext-key-id {
        type uint8;
        description
            "TBD";
    }
    description
        "A Master Key Tuple (MKT) describes TCP-AO properties to be
        associated with one or more connections.";
}

grouping ao {
    leaf enable {
        type boolean;
        default "false";
        description
            "Enable support of TCP-Authentication Option (TCP-AO).";
    }

    leaf current-key {
        type binary;
        description
            "The Master Key Tuple (MKT) currently used to authenticate
            outgoing segments, whose SendID is inserted in outgoing
            segments as KeyID. Incoming segments are authenticated
            using the MKT corresponding to the segment and its TCP-AO
            KeyID as matched against the MKT TCP connection identifier
            and the MKT RecvID. There is only one current-key at any
            given time on a particular connection.

            Every TCP connection in a non-IDLE state MUST have at most
            one current_key specified.";
        reference
            "RFC 5925: The TCP Authentication Option.";
    }

    leaf rnext-key {
        type binary;
        description
            "The MKT currently preferred for incoming (received)
            segments, whose RecvID is inserted in outgoing segments as
```

```
    RNextKeyID.

    Each TCP connection in a non-IDLE state MUST have at most
    one rnext_key specified.";
  reference
    "RFC 5925: The TCP Authentication Option.";
}

leaf-list sne {
  type uint32;
  min-elements 1;
  max-elements 2;
  description
    "A pair of Sequence Number Extensions (SNEs). SNEs are used
    to prevent replay attacks. Each SNE is initialized to zero
    upon connection establishment.";
  reference
    "RFC 5925: The TCP Authentication Option.";
}

leaf-list mkt {
  type binary;
  min-elements 1;
  description
    "One or more MKTs. These are the MKTs that match this
    connection's socket pair.";
  reference
    "RFC 5925: The TCP Authentication Option.";
}
description
  "Authentication Option (AO) for TCP.";
reference
  "RFC 5925: The TCP Authentication Option.";
}

// TCP general configuration groupings

grouping tcp-mss-grouping {
  description "Maximum Segment Size (MSS) parameters";

  leaf tcp-mss {
    type uint16;
    description
      "Sets the max segment size for TCP connections.";
  }
} // grouping tcp-mss-grouping

grouping tcp-mtu-grouping {
```

```
    description "Maximum Transfer Unit (MTU) discovery parameters";

    leaf tcp-mtu-discovery {
        type boolean;
        default false;
        description
            "Turns path mtu discovery for TCP connections on (true) or
             off (false)";
    }
} // grouping tcp-mtu-grouping

grouping tcp-sack-grouping {
    description "Selective Acknowledgements (SACK) parameters";

    leaf sack-enable {
        type boolean;

        description
            "Enable support of Selective Acknowledgements (SACK)";
    }
} // grouping tcp-sack-grouping

grouping tcp-timestamps-grouping {
    description "Timestamp parameters";

    leaf timestamps-enable {
        type boolean;

        description
            "Enable support of timestamps";
    }
} // grouping tcp-timestamps-grouping

grouping tcp-fin-timeout-grouping {
    description "TIME WAIT timeout parameters";

    leaf fin-timeout {
        type uint16;
        units "seconds";
        description
            "When a connection is closed actively, it must linger in
             TIME-WAIT state for a time 2xMSL (Maximum Segment Lifetime).
             This parameter sets the TIME-WAIT timeout duration in
             seconds.";
    }
} // grouping tcp-fin-timeout-grouping

grouping tcp-ecn-grouping {
```

```
description "Explicit Congestion Notification (ECN) parameters";

leaf ecn-enable {
  type enumeration {
    enum disable;
    enum passive;
    enum active;
  }
  description
    "Enabling of ECN.";
}
} // grouping tcp-ecn-grouping

// augment statements

container tcp {
  presence "The container for TCP configuration.";

  description
    "TCP container.";

  container connections {
    list connection {
      key "local-address remote-address local-port remote-port";

      leaf local-address {
        type inet:ip-address;
        description
          "Local address that forms the connection identifier.";
      }

      leaf remote-address {
        type inet:ip-address;
        description
          "Remote address that forms the connection identifier.";
      }

      leaf local-port {
        type inet:port-number;
        description
          "Local TCP port that forms the connection identifier.";
      }

      leaf remote-port {
        type inet:port-number;
        description
          "Remote TCP port that forms the connection identifier.";
```

```
    }

    container common {
      uses tcpcmn:tcp-common-grouping;
      uses ao;
      description
        "Common definitions of TCP configuration. This includes
        parameters such as keepalives and idle time, that can
        be part of either the client or server.";
    }

    container server {
      if-feature server;
      uses tcps:tcp-server-grouping;
      description
        "Definitions of TCP server configuration.";
    }

    container client {
      if-feature client;
      uses tcpc:tcp-client-grouping;
      description
        "Definitions of TCP client configuration.";
    }
  }
  description
    "Connection related parameters.";
}
description
  "A container of all TCP connections.";
}
}
```

<CODE ENDS>

5. IANA Considerations

5.1. The IETF XML Registry

This document registers two URIs in the "ns" subregistry of the IETF XML Registry [RFC3688]. Following the format in IETF XML Registry [RFC3688], the following registrations are requested:

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

5.2. The YANG Module Names Registry

This document registers a YANG modules in the YANG Module Names registry YANG - A Data Modeling Language [RFC6020]. Following the format in YANG - A Data Modeling Language [RFC6020], the following registrations are requested:

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

6. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

7. References

7.1. Normative References

- [I-D.ietf-netconf-tcp-client-server]
Watsen, K. and M. Scharf, "YANG Groupings for TCP Clients and TCP Servers", draft-ietf-netconf-tcp-client-server-03 (work in progress), October 2019.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.
- [RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <<https://www.rfc-editor.org/info/rfc8342>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

7.2. Informative References

- [I-D.ietf-dots-data-channel]
Boucadair, M. and R. K, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Data Channel Specification", draft-ietf-dots-data-channel-31 (work in progress), July 2019.
- [I-D.ietf-idr-bgp-model]
Jethanandani, M., Patel, K., Hares, S., and J. Haas, "BGP YANG Model for Service Provider Networks", draft-ietf-idr-bgp-model-07 (work in progress), October 2019.
- [I-D.ietf-taps-interface]
Trammell, B., Welzl, M., Enghardt, T., Fairhurst, G., Kuehlewind, M., Perkins, C., Tiesel, P., Wood, C., and T. Pauly, "An Abstract Application Layer Interface to Transport Services", draft-ietf-taps-interface-04 (work in progress), July 2019.
- [RFC3873] Pastor, J. and M. Belinchon, "Stream Control Transmission Protocol (SCTP) Management Information Base (MIB)", RFC 3873, DOI 10.17487/RFC3873, September 2004, <<https://www.rfc-editor.org/info/rfc3873>>.
- [RFC4022] Raghunarayan, R., Ed., "Management Information Base for the Transmission Control Protocol (TCP)", RFC 4022, DOI 10.17487/RFC4022, March 2005, <<https://www.rfc-editor.org/info/rfc4022>>.
- [RFC4113] Fenner, B. and J. Flick, "Management Information Base for the User Datagram Protocol (UDP)", RFC 4113, DOI 10.17487/RFC4113, June 2005, <<https://www.rfc-editor.org/info/rfc4113>>.
- [RFC4898] Mathis, M., Heffner, J., and R. Raghunarayan, "TCP Extended Statistics MIB", RFC 4898, DOI 10.17487/RFC4898, May 2007, <<https://www.rfc-editor.org/info/rfc4898>>.
- [RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules", RFC 6643, DOI 10.17487/RFC6643, July 2012, <<https://www.rfc-editor.org/info/rfc6643>>.

- [RFC8512] Boucadair, M., Ed., Sivakumar, S., Jacquenet, C., Vinapamula, S., and Q. Wu, "A YANG Module for Network Address Translation (NAT) and Network Prefix Translation (NPT)", RFC 8512, DOI 10.17487/RFC8512, January 2019, <<https://www.rfc-editor.org/info/rfc8512>>.
- [RFC8513] Boucadair, M., Jacquenet, C., and S. Sivakumar, "A YANG Data Model for Dual-Stack Lite (DS-Lite)", RFC 8513, DOI 10.17487/RFC8513, January 2019, <<https://www.rfc-editor.org/info/rfc8513>>.
- [RFC8519] Jethanandani, M., Agarwal, S., Huang, L., and D. Blair, "YANG Data Model for Network Access Control Lists (ACLs)", RFC 8519, DOI 10.17487/RFC8519, March 2019, <<https://www.rfc-editor.org/info/rfc8519>>.

Appendix A. Acknowledgements

Michael Scharf was supported by the StandICT.eu project, which is funded by the European Commission under the Horizon 2020 Programme.

The following persons have contributed to this document by reviews:
Mohamed Boucadair

Appendix B. Changes compared to previous versions

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

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

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

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

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

- o Editorial improvements

Authors' Addresses

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

Email: michael.scharf@hs-esslingen.de

Vishal Murgai
Cisco Systems Inc

Email: vmurgai@cisco.com

Mahesh Jethanandani
VMware

Email: mjethanandani@gmail.com