        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.

Table of Contents

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 require 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 an 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 an 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 over-
arching 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
   then 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., Dukkipati, 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-sctpecn]
             Stewart, R., Tuexen, M., and X. Dong, "ECN for Stream
             Control Transmission Protocol (SCTP)", draft-stewart-
             tsvwg-sctpecn-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@bobbriscoe.net
   URI: http://bobbriscoe.net/

                    More Accurate ECN Feedback in TCP
                     draft-ietf-tcpm-accurate-ecn-06

   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). Recently,ew TCP
   mechanisms like Congestion Exposure (ConEx) or Data Center TCP
   (DCTCP) 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 overloads
   the three existing ECN-related flags in the TCP header and provides
   additional information in a new TCP option.

Status of This Memo

Copyright Notice

Table of Contents

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
   more accurate ECN feedback information than provided by the feedback
   scheme as specified in [RFC3168] whenever more than one marking is
   received in one RTT. 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 as well as the currently reserved and previously called NS flag in the main TCP header with new definitions, so both ends have to support the new wire protocol before it can be used. Therefore during the TCP handshake the two ends use the three ECN-related flags in the TCP header to negotiate the most advanced feedback protocol that they can both support.

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) [I-D.ietf-tcpm-alternativebackoff-ecn].

It is likely (but not required) that the AccECN protocol will be implemented along with the following experimental additions to the TCP-ECN protocol: ECN-capable TCP control packets and retransmissions [I-D.ietf-tcpm-generalized-ecn], which includes the ECN-capable SYN/ ACK experiment [RFC5562]; and testing receiver non-compliance [I-D.moncaster-tcpm-rcv-cheat].

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 it is deployed and if it satisfies the requirements of [RFC7560] in the consensus opinion of the IETF tcpm working group. In short, this requires that it improves the accuracy and timeliness of TCP's ECN feedback, as claimed in Section 5, while striking a balance between the conflicting requirements of resilience, integrity and minimisation of overhead. It also requires that it is not unduly complex, and that it is compatible with prevalent equipment behaviours in the current Internet (e.g. hardware offloading and middleboxes), whether or not they comply with standards.

Testing will mostly focus on fall-back strategies in case of middlebox interference. Current recommended strategies are specified

in Sections 3.1.2, 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 feasibiliy 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
patter which especially can have an impact on hardware offload.
Further experimentation is needed to advise if this should a hard
requirement or just prefer 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 | Codepoint | Description |
   | (binary) | name | |
   +--------------------+--------------+--------------------------+
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
  +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
  |               |           | N | C | E | U | A | P | R | S | F |
  | Header Length |  Reserved | S | W | C | R | C | S | S | Y | I |
  |               |           | R | E | G | K | H | T | N | N |
  +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```
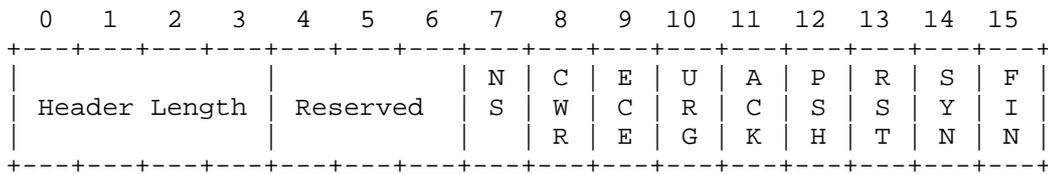
        Figure 1: The (post-ECN Nonce) definition of the TCP header flags

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 cyling 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 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
[I-D.moncaster-tcpm-rcv-cheat]).

The initial SYN is the most critical control packet, so AccECN
provides feedback on whether it is CE marked. Although RFC 3168
prohibits an ECN-capable SYN, providing feedback of CE 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 CE feedback, 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 renames the TCP flag at bit 7 of
   the TCP header flags from NS (Nonce Sum) to AE (Accurate ECN) (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 3.1.2.

   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
		AE	CWR	ECE	AE	CWR	ECE	
AccECN	AccECN	1	1	1	0	1	0	AccECN (Not-ECT on SYN)
AccECN	AccECN	1	1	1	0	1	1	AccECN (ECT1 on SYN)
AccECN	AccECN	1	1	1	1	0	0	AccECN (ECT0 on SYN)
AccECN	AccECN	1	1	1	1	1	0	AccECN (CE on SYN)
AccECN	Nonce	1	1	1	1	0	1	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 also 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. Retransmission of the SYN

If the sender of an AccECN SYN times out before receiving the SYN/
ACK, the sender SHOULD attempt to negotiate the use of AccECN at
least one more time by continuing to set all three TCP ECN flags on
the first retransmitted SYN (using the usual retransmission time-
outs). If this first retransmission also fails to be acknowledged,
the sender SHOULD send subsequent retransmissions of the SYN 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.e1b. 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.e1b 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.e1b.=
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 AccECN 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.e1b intended to track the equivalent
counters at the Data Receiver. When a host enters AccECN mode, it
initializes them to s.cep = 5, s.e0b = 1 and s.ceb = s.e1b.= 0.

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

```
   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
 +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
 |                   |               |           | U | A | P | R | S | F |
 |  Header Length    |   Reserved    |    ACE    | R | C | S | S | Y | I |
 |                   |               |           | G | K | H | T | N | N |
 +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```
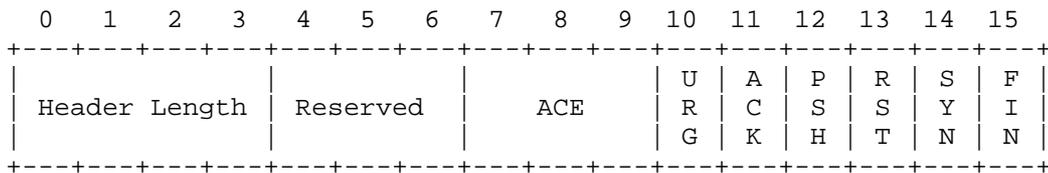
Figure 2: Definition of the ACE field within bytes 13 and 14 of the
      TCP Header (when AccECN has been negotiated and SYN=0).

The original definition of these three flags in the TCP header,
including the addition of support for the ECN Nonce, is shown for
comparison in Figure 1. This specification does not rename these
three TCP flags to ACE unconditionally; it merely overloads them with
another name and definition once an AccECN connection has been
established.

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 3WHS, a TCP client (A) in AccECN mode MUST use the ACE field 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, 2}	Disable ECN
0b001	{Notes 2, 3}	5
0b010	Not-ECT	5
0b011	ECT(1)	5
0b100	ECT(0)	5
0b101	Currently Unused {Note 3}	5
0b110	CE	6
0b111	Currently Unused {Note 3}	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 a server is in AccECN mode, there ought to be no valid case where the ACE field on the last ACK of the 3WHS has a value of 0b000 or 0b001.

However, 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. Then these two values remind it that it did not enter AccECN mode (see Section 4.1 for details).

{Note 3}: If the server is in AccECN mode, these values are Currently Unused but the AccECN server's behaviour is still defined for forward compatibility.

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 of the IP/ECN field,
   for the remainder of the connection the client MUST NOT send ECN-
   capable packets, but it MUST continue to feed back any ECN markings
   on arriving packets.

   The value of the ACE field on the last ACK of the 3WHS indicates the
   value of the IP/ECN field when the SYN/ACK arrived at the client.
   The server can compare this with how it originally set the IP/ECN
   field on the SYN/ACK. If this comparison implies an unsafe
   transition of the IP/ECN field, for the remainder of the connection
   the server MUST NOT send ECN-capable packets, but it MUST continue to
   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 has to 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.e1b counters, respectively. The initial 'E'
of each field name stands for 'Echo'.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Kind = TBD1 |  Length = 11  |            EE0B field         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| EE0B (cont'd) |            ECEB field                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     EE1B field                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
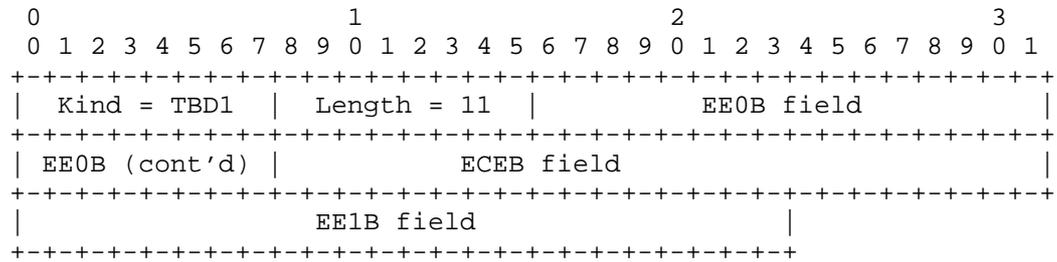
Figure 3: The AccECN Option

The Data Receiver 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 feedback 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 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 use of the AccECN option is optional for the Data Receiver. If
the Data Receiver intents to use the AccECN option at any time during
the rest of the connection it strongly recommended to also test its
path traversal by including it in the SYN/ACK as specified in the
next section. By default the use of the AccECN option is
RECOMMENDED.

3.2.7. Path Traversal of the AccECN Option

3.2.7.1. Testing the AccECN Option during the Handshake

The TCP client MUST NOT include the AccECN TCP Option on the SYN.
Nonetheless, if the AccECN negotiation using the ECN flags in the
main TCP header (Section 3.1) is successful, it implicitly declares
that the endpoints also support the AccECN TCP Option. A fall-back
strategy for the loss of the SYN (possibly due to middlebox
interference) is specified in Section 3.1.2.

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

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

A host MAY NOT include an AccECN Option in any of these three cases
if it has cached knowledge that the packet would be likely to be

blocked on the path to the other host if it included an AccECN
Option.

3.2.7.2. Testing for Loss of Packets Carrying the AccECN Option

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

   If the TCP client has successfully negotiated AccECN but does not
   receive an AccECN Option on the SYN/ACK, it 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 r.e0b
   counter to a non-zero value. Therefore, in either direction the
   initial value of the EE0B field in the AccECN Option (if one exists)
   ought to be non-zero. If AccECN has been negotiated:

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

   o the TCP client MAY check the initial value of the EE0B field on
      the SYN/ACK. If the initial value of the EE0B field is zero, the
      client will switch into a mode that ignores the AccECN Option for
      this half connection.

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

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 will lead to some additional ACKs but it feeds
back the timing and the order in which ECN marks are received with
minimal additional complexity.

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).

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
      [I-D.moncaster-tcpm-rcv-cheat]. 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 3WHS so that
    it can fall back to operation without ECN and/or operation without
    the AccECN Option.

Forward Compatibility: The behaviour of endpoints and middleboxes is
    carefully defined for all reserved or currently unused codepoints
    in the scheme, 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 is being reclassified as historic [RFC8311].
The flag will now be defined as:

```
+-----+------------------+-----------+
| Bit | Name             | Reference |
+-----+------------------+-----------+
| 7   | AE (Accurate ECN)| RFC XXXX  |
+-----+------------------+-----------+
```

[TO BE REMOVED: This registration should take place at the following
location: https://www.iana.org/assignments/tcp-header-flags/tcp-
header-flags.xhtml#tcp-header-flags-1]

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 Data Sender 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
   and Michael Tuexen 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-alternativebackoff-ecn]
             Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst,
             "TCP Alternative Backoff with ECN (ABE)", draft-ietf-tcpm-
             alternativebackoff-ecn-06 (work in progress), February
             2018.

   [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-02 (work in progress),
             October 2017.

   [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-01 (work in
             progress), October 2017.

   [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.

   [I-D.moncaster-tcpm-rcv-cheat]
             Moncaster, T., Briscoe, B., and A. Jacquet, "A TCP Test to
             Allow Senders to Identify Receiver Non-Compliance", draft-
             moncaster-tcpm-rcv-cheat-03 (work in progress), July 2014.

   [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.

             (to appear)

   [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>.

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:

      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

      ECEB = r.ceb % 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. Then 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:

      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

      ACE = r.cep % DIVACE.

   If the Data Sender received an earlier value of the counter that had
   been delayed due to ACK reordering, it might incorrectly calculate
   that the ACE field had wrapped. Therefore, on the arrival of every
   ACK, the Data Sender 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 ((newlyAckedB > 0) || (newlyAckedB == 0 && newlyAckedT > 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 newlyAckedB is the
amount of data that an ACK acknowledges since the previous ACK, then
the Data Sender could assume that this acknowledges newlyAckedPkt
full-sized segments, where newlyAckedPkt = newlyAckedB/MSS. Then it
could assume that the ACE field incremented by

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

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

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

The simple algorithm for dSafer.cep above requires no monitoring of
prevailing conditions and it would still be safe if, for example,
segments were on average at least 5% of 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:
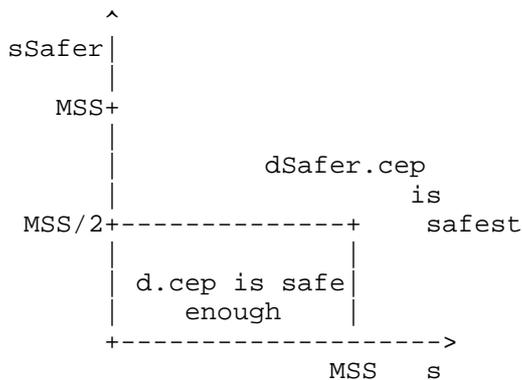
```
s.cep += dSafer.cep
```

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 AccECN Option

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

```
          ^
  sSafer|
        |
    MSS+
        |
        |          dSafer.cep
        |                 is
  MSS/2+-------------+    safest
        |            |
        | d.cep is safe|
        |    enough   |
        +-------------------->
                    MSS    s
```

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.e1b. Note that, because
r.e0b is initialized to 1 and the other two counters are initialized
to 0, the initial sum will be 1, which matches the initial offset of
the TCP sequence number on completion of the 3WHS.

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

Authors' Addresses

Bob Briscoe
CableLabs
UK

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

Mirja Kuehlewind
ETH Zurich
Zurich
Switzerland

EMail: mirja.kuehlewind@tik.ee.ethz.ch

   Richard Scheffenegger
   Vienna
   Austria

   EMail: rscheff@gmx.at

Network Working Group   N. Khademi
Internet-Draft   M. Welzl
Intended status: Experimental   University of Oslo
Expires: September 21, 2018   G. Armitage
                                       Swinburne University of Technology
                                                         G. Fairhurst
                                                University of Aberdeen
                                                       March 20, 2018

                 TCP Alternative Backoff with ECN (ABE)
                draft-ietf-tcpm-alternativebackoff-ecn-07

   Abstract

      Active Queue Management (AQM) mechanisms allow for burst tolerance
      while enforcing short queues to minimise the time that packets spend
      enqueued at a bottleneck. This can cause noticeable performance
      degradation for TCP connections traversing such a bottleneck,
      especially if there are only a few flows or their bandwidth-delay-
      product is large. An Explicit Congestion Notification (ECN) signal
      indicates that an AQM mechanism is used at the bottleneck, and
      therefore the bottleneck network queue is likely to be short. This
      document therefore proposes an update to RFC3168, which changes the
      TCP sender-side ECN reaction in congestion avoidance to reduce the
      Congestion Window (cwnd) by a smaller amount than the congestion
      control algorithm's reaction to inferred packet loss.

   Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

   Explicit Congestion Notification (ECN) [RFC3168] makes it possible
   for an Active Queue Management (AQM) mechanism to signal the presence
   of incipient congestion without incurring packet loss. This lets the
   network deliver some packets to an application that would have been
   dropped if the application or transport did not support ECN. This
   packet loss reduction is the most obvious benefit of ECN, but it is
   often relatively modest. Other benefits of deploying ECN have been
   documented in RFC8087 [RFC8087].

   The rules for ECN were originally written to be very conservative,
   and required the congestion control algorithms of ECN-Capable

transport protocols to treat ECN congestion signals exactly the same
as they would treat an inferred packet loss [RFC3168].

Research has demonstrated the benefits of reducing network delays
that are caused by interaction of loss-based TCP congestion control
and excessive buffering [BUFFERBLOAT]. This has led to the creation
of new AQM mechanisms like PIE [RFC8033] and CoDel
[CODEL2012][RFC8289], which prevent bloated queues that are common
with unmanaged and excessively large buffers deployed across the
Internet [BUFFERBLOAT].

The AQM mechanisms mentioned above aim to keep a sustained queue
short while tolerating transient (short-term) packet bursts.
However, currently used loss-based congestion control mechanisms
cannot always utilise a bottleneck link well where there are short
queues. For example, a TCP sender must be able to store at least an
end-to-end bandwidth-delay product (BDP) worth of data at the
bottleneck buffer if it is to maintain full path utilisation in the
face of loss-induced reduction of cwnd [RFC5681], which effectively
doubles the amount of data that can be in flight, the maximum round-
trip time (RTT) experience, and the path's effective RTT using the
network path.

Modern AQM mechanisms can use ECN to signal the early signs of
impending queue buildup long before a tail-drop queue would be forced
to resort to dropping packets. It is therefore appropriate for the
transport protocol congestion control algorithm to have a more
measured response when an early-warning signal of congestion is
received in the form of an ECN CE-marked packet. Recognizing these
changes in modern AQM practices, more recent rules have relaxed the
strict requirement that ECN signals be treated identically to
inferred packet loss [RFC8311]. Following these newer, more flexible
rules, this document defines a new sender-side-only congestion
control response, called "ABE" (Alternative Backoff with ECN). ABE
improves TCP's average throughput when routers use AQM controlled
buffers that allow for short queues only.

2. Definitions

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
   "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
   document are to be interpreted as described in RFC 2119 [RFC2119].

3. Specification

   This specification updates the congestion control algorithm of an
   ECN-Capable TCP transport protocol by changing the TCP sender
   response to feedback from the TCP receiver that indicates reception

of a CE-marked packet, i.e., receipt of a packet with the ECN-Echo
flag (defined in [RFC3168]) set.

It updates the following text in section 6.1.2 of the ECN
specification [RFC3168] :

> The indication of congestion should be treated just as a
> congestion loss in non-ECN-Capable TCP. That is, the TCP source
> halves the congestion window "cwnd" and reduces the slow start
> threshold "ssthresh".

Replacing this with:

> Receipt of a packet with the ECN-Echo flag SHOULD trigger the TCP
> source to set the slow start threshold (ssthresh) to 0.8 times the
> FlightSize, with a lower bound of 2 * SMSS applied to the result.
> As in [RFC5681], the TCP sender also reduces the cwnd value to no
> more than the new ssthresh value. RFC 3168 section 6.1.2 provides
> guidance on setting a cwnd less than 2 * SMSS.

4. Discussion

   Much of the technical background to ABE can be found in a research
   paper [ABE2017]. This paper used a mix of experiments, theory and
   simulations with NewReno [RFC5681] and CUBIC [RFC8312] to evaluate
   the technique. The technique was shown to present "...significant
   performance gains in lightly-multiplexed [few concurrent flows]
   scenarios, without losing the delay-reduction benefits of deploying
   CoDel or PIE". The performance improvement is achieved when reacting
   to ECN-Echo in congestion avoidance (when ssthresh > cwnd) by
   multiplying cwnd and ssthresh with a value in the range [0.7,0.85].
   Applying ABE when cwnd <= ssthresh is not currently recommended, but
   may benefit from additional attention, experimentation and
   specification.

4.1. Why Use ECN to Vary the Degree of Backoff?

   AQM mechanisms such as CoDel [RFC8289] and PIE [RFC8033] set a delay
   target in routers and use congestion notifications to constrain the
   queuing delays experienced by packets, rather than in response to
   impending or actual bottleneck buffer exhaustion. With current
   default delay targets, CoDel and PIE both effectively emulate a
   bottleneck with a short queue (section II, [ABE2017]) while also
   allowing short traffic bursts into the queue. This provides
   acceptable performance for TCP connections over a path with a low
   BDP, or in highly multiplexed scenarios (many concurrent transport
   flows). However, in a lightly-multiplexed case over a path with a

large BDP, conventional TCP backoff leads to gaps in packet
transmission and under-utilisation of the path.

Instead of discarding packets, an AQM mechanism is allowed to mark
ECN-Capable packets with an ECN CE-mark. The reception of a CE-mark
feedback not only indicates congestion on the network path, it also
indicates that an AQM mechanism exists at the bottleneck along the
path, and hence the CE-mark likely came from a bottleneck with a
controlled short queue. Reacting differently to an ECN-signalled
congestion than to an inferred packet loss can then yield the benefit
of a reduced back-off when queues are short. Using ECN can also be
advantageous for several other reasons [RFC8087].

The idea of reacting differently to inferred packet loss and
detection of an ECN-signalled congestion pre-dates this document.
For example, previous research proposed using ECN CE-marked feedback
to modify TCP congestion control behaviour via a larger
multiplicative decrease factor in conjunction with a smaller additive
increase factor [ICC2002]. The goal of this former work was to
operate across AQM bottlenecks using Random Early Detection (RED)
that were not necessarily configured to emulate a short queue (The
current usage of RED as an Internet AQM method is limited [RFC7567]).

4.2. Focus on ECN as Defined in RFC3168

Some transport protocol mechanisms rely on ECN semantics that differ
from the original ECN definition [RFC3168]. For instance, Accurate
ECN [I-D.ietf-tcpm-accurate-ecn] permits more frequent and detailed
feedback. Use of such mechanisms (including Accurate ECN, Datacenter
TCP (DCTCP) [RFC8257], or Congestion Exposure (ConEx) [RFC7713]) is
out of scope for this document. This specification focuses on ECN as
defined in [RFC3168].

4.3. Choice of ABE Multiplier

ABE decouples the reaction of a TCP sender to inferred packet loss
and ECN-signalled congestion in the congestion avoidance phase. To
achieve this, ABE uses a different scaling factor in Equation 4 in
Section 3.1 of [RFC5681]. The description respectively uses
beta_{loss} and beta_{ecn} to refer to the multiplicative decrease
factors applied in response to inferred packet loss, and in response
to a receiver indicating ECN-signalled congestion. For non-ECN-
enabled TCP connections, only beta_{loss} applies.

In other words, in response to inferred packet loss:

    ssthresh = max (FlightSize * beta_{loss}, 2 * SMSS)

and in response to an indication of an ECN-signalled congestion:

    ssthresh = max (FlightSize * beta_{ecn}, 2 * SMSS)

    and

    cwnd = ssthresh

    (If ssthresh == 2 * SMSS, RFC 3168 section 6.1.2 provides guidance
    on setting a cwnd lower than 2 * SMSS.)

where FlightSize is the amount of outstanding data in the network,
upper-bounded by the smaller of the sender's cwnd and the receiver's
advertised window (rwnd) [RFC5681]. The higher the values of
beta_{loss} and beta_{ecn}, the less aggressive the response of any
individual backoff event.

The appropriate choice for beta_{loss} and beta_{ecn} values is a
balancing act between path utilisation and draining the bottleneck
queue. More aggressive backoff (smaller beta_*) risks underutilising
the path, while less aggressive backoff (larger beta_*) can result in
slower draining of the bottleneck queue.

The Internet has already been running with at least two different
beta_{loss} values for several years: the standard value is 0.5
[RFC5681], and the Linux implementation of CUBIC [RFC8312] has used a
multiplier of 0.7 since kernel version 2.6.25 released in 2008. ABE
proposes no change to beta_{loss} used by current TCP
implementations.

The recommendation in Section 3 in this document corresponds to a
value of beta_{ecn}=0.8. This recommended beta_{ecn} value is only
applicable for the standard TCP congestion control [RFC5681]. The
selection of beta_{ecn} enables tuning the response of a TCP
connection to shallow AQM marking thresholds. beta_{loss}
characterizes the response of a congestion control algorithm to
packet loss, i.e., exhaustion of buffers (of unknown depth).
Different values for beta_{loss} have been suggested for TCP
congestion control algorithms. Consequently, beta_{ecn} is likely to
be an algorithm-specific parameter rather than a constant multiple of
the algorithm's existing beta_{loss}.

A range of tests (section IV, [ABE2017]) with NewReno and CUBIC over
CoDel and PIE in lightly-multiplexed scenarios have explored this
choice of parameter. The results of these tests indicate that CUBIC
connections benefit from beta_{ecn} of 0.85 (cf. beta_{loss} = 0.7),

and NewReno connections see improvements with beta_{ecn} in the range
0.7 to 0.85 (cf. beta_{loss} = 0.5).

5. ABE Deployment Requirements

   This update is a sender-side only change. Like other changes to
   congestion control algorithms, it does not require any change to the
   TCP receiver or to network devices. It does not require any ABE-
   specific changes in routers or the use of Accurate ECN feedback
   [I-D.ietf-tcpm-accurate-ecn] by a receiver.

   RFC3168 states that the congestion control response to an ECN-
   signalled congestion is the same as the response to a dropped packet
   [RFC3168]. [RFC8311] updates this specification to allow systems to
   provide a different behaviour when they experience ECN-signalled
   congestion rather than packet loss. The present specification
   defines such an experiment and has thus been assigned an Experimental
   status before being proposed as a Standards-Track update.

   The purpose of the Internet experiment is to collect experience with
   deployment of ABE, and confirm the safety in deployed networks using
   this update to TCP congestion control.

   When used with bottlenecks that do not support ECN-marking the
   specification does not modify the transport protocol.

   To evaluate the benefit, this experiment therefore requires support
   in AQM routers for ECN-marking of packets carrying the ECN-Capable
   Transport, ECT(0), codepoint [RFC3168].

   If the method is only deployed by some senders, and not by others,
   the senders that use this method can gain some advantage, possibly at
   the expense of other flows that do not use this updated method.
   Because this advantage applies only to ECN-marked packets and not to
   packet loss indications, an ECN-Capable bottleneck will still fall
   back to dropping packets if an TCP sender using ABE is too
   aggressive, and the result is no different than if the TCP sender was
   using traditional loss-based congestion control.

   A TCP sender reacts to loss or ECN marks only once per round-trip
   time. Hence, if a sender would first be notified of an ECN mark and
   then learn about loss in the same round-trip, it would only react to
   the first notification (ECN) but not to the second (loss). RFC3168
   specified a reaction to ECN that was equal to the reaction to loss
   [RFC3168].

   ABE also responds to congestion once per RTT, and therefore it does
   not respond to further loss within the same RTT, since ABE has

already reduced the congestion window. If congestion persists after
such reduction, ABE continues to reduce the congestion window in each
consecutive RTT. This consecutive reduction can protect the network
against long-standing unfairness in the case of AQM algorithms that
do not keep a small average queue length.

The result of this Internet experiment ought to include an
investigation of the implications of experiencing an ECN-CE mark
followed by loss within the same RTT. At the end of the experiment,
this will be reported to the TCPM WG (or IESG).

6. Acknowledgements

7. IANA Considerations

XX RFC ED - PLEASE REMOVE THIS SECTION XXX

This document includes no request to IANA.

8. Implementation Status

ABE is implemented as a patch for Linux and FreeBSD. It is meant for
research and available for download from
http://heim.ifi.uio.no/naeemk/research/ABE/. This code was used to
produce the test results that are reported in [ABE2017]. The FreeBSD
code has been committed to the mainline kernel on March 19, 2018
[ABE-FreeBSD].

9. Security Considerations

   The described method is a sender-side only transport change, and does
   not change the protocol messages exchanged. The security
   considerations for ECN [RFC3168] therefore still apply.

   This is a change to TCP congestion control with ECN that will
   typically lead to a change in the capacity achieved when flows share
   a network bottleneck. This could result in some flows receiving more
   than their fair share of capacity. Similar unfairness in the way
   that capacity is shared is also exhibited by other congestion control
   mechanisms that have been in use in the Internet for many years
   (e.g., CUBIC [RFC8312]). Unfairness may also be a result of other
   factors, including the round trip time experienced by a flow. ABE
   applies only when ECN-marked packets are received, not when packets
   are lost, hence use of ABE cannot lead to congestion collapse.

10. Revision Information

   XX RFC ED - PLEASE REMOVE THIS SECTION XXX

   -07. Addressed comments following WGLC.

   o Updated Reference citations

   o Removed paragraph containing a wrong statement related to timeout
      in section 4.1.

   o Discuss what happens when cwnd <= ssthresh

   o Added text on Concern about lower bound of 2*SMSS

   -06. Addressed Michael Scharf's comments.

   -05. Refined the description of the experiment based on feedback at
   IETF-100. Incorporated comments from David Black.

   -04. Incorporates review comments from Lawrence Stewart and the
   remaining comments from Roland Bless. References are updated.

   -03. Several review comments from Roland Bless are addressed.
   Consistent terminology and equations. Clarification on the scope of
   recommended beta_{ecn} value.

   -02. Corrected the equations in Section 4.3. Updated the
   affiliations. Lower bound for cwnd is defined. A recommendation for
   window-based transport protocols is changed to cover all transport
   protocols that implement a congestion control reduction to an ECN

congestion signal. Added text about ABE's FreeBSD mainline kernel status including a reference to the FreeBSD code review page. References are updated.

-01. Text improved, mainly incorporating comments from Stuart Cheshire. The reference to a technical report has been updated to a published version of the tests [ABE2017]. Used "AQM Mechanism" throughout in place of other alternatives, and more consistent use of technical language and clarification on the intended purpose of the experiments required by EXP status. There was no change to the technical content.

-00. draft-ietf-tcpm-alternativebackoff-ecn-00 replaces draft-khademi-tcpm-alternativebackoff-ecn-01. Text describing the nature of the experiment was added.

Individual draft -01. This I-D now refers to draft-black-tsvwg-ecn-experimentation-02, which replaces draft-khademi-tsvwg-ecn-response-00 to make a broader update to RFC3168 for the sake of allowing experiments. As a result, some of the motivating and discussing text that was moved from draft-khademi-alternativebackoff-ecn-03 to draft-khademi-tsvwg-ecn-response-00 has now been re-inserted here.

Individual draft -00. draft-khademi-tsvwg-ecn-response-00 and draft-khademi-tcpm-alternativebackoff-ecn-00 replace draft-khademi-alternativebackoff-ecn-03, following discussion in the TSVWG and TCPM working groups.

11. References

11.1. 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>.

   [RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF
              Recommendations Regarding Active Queue Management",
              BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015,
              <https://www.rfc-editor.org/info/rfc7567>.

   [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>.

11.2. Informative References

   [ABE-FreeBSD]
              "ABE patch review in FreeBSD",
              <https://svnweb.freebsd.org/
              base?view=revision&revision=331214>.

   [ABE2017] Khademi, N., Armitage, G., Welzl, M., Fairhurst, G.,
              Zander, S., and D. Ros, "Alternative Backoff: Achieving
              Low Latency and High Throughput with ECN and AQM", IFIP
              NETWORKING 2017, Stockholm, Sweden, June 2017.

   [BUFFERBLOAT]
              Gettys, J. and K. Nichols, "Bufferbloat: Dark Buffers in
              the Internet", November 2011.

   [CODEL2012]
              Nichols, K. and V. Jacobson, "Controlling Queue Delay",
              July 2012, <http://queue.acm.org/detail.cfm?id=2209336>.

   [I-D.ietf-tcpm-accurate-ecn]
              Briscoe, B., Kuehlewind, M., and R. Scheffenegger, "More
              Accurate ECN Feedback in TCP", draft-ietf-tcpm-accurate-
              ecn-06 (work in progress), March 2018.

   [ICC2002] Kwon, M. and S. Fahmy, "TCP Increase/Decrease Behavior
              with Explicit Congestion Notification (ECN)", IEEE
              ICC 2002, New York, New York, USA, May 2002,
              <http://dx.doi.org/10.1109/ICC.2002.997262>.

   [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>.

   [RFC8033] Pan, R., Natarajan, P., Baker, F., and G. White,
              "Proportional Integral Controller Enhanced (PIE): A
              Lightweight Control Scheme to Address the Bufferbloat
              Problem", RFC 8033, DOI 10.17487/RFC8033, February 2017,
              <https://www.rfc-editor.org/info/rfc8033>.

   [RFC8087] Fairhurst, G. and M. Welzl, "The Benefits of Using
              Explicit Congestion Notification (ECN)", RFC 8087,
              DOI 10.17487/RFC8087, March 2017,
              <https://www.rfc-editor.org/info/rfc8087>.

   [RFC8289] Nichols, K., Jacobson, V., McGregor, A., Ed., and J.
              Iyengar, Ed., "Controlled Delay Active Queue Management",
              RFC 8289, DOI 10.17487/RFC8289, January 2018,
              <https://www.rfc-editor.org/info/rfc8289>.

   [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

   Naeem Khademi
   University of Oslo
   PO Box 1080 Blindern
   Oslo N-0316
   Norway

   Email: naeemk@ifi.uio.no

   Michael Welzl
   University of Oslo
   PO Box 1080 Blindern
   Oslo N-0316
   Norway

   Email: michawe@ifi.uio.no

Grenville Armitage
Internet For Things (I4T) Research Group
Swinburne University of Technology
PO Box 218
John Street, Hawthorn
Victoria 3122
Australia

Email: garmitage@swin.edu.au

Godred Fairhurst
University of Aberdeen
School of Engineering, Fraser Noble Building
Aberdeen AB24 3UE
UK

Email: gorry@erg.abdn.ac.uk

TCPM WG   J. Touch
Internet Draft
Intended status: Informational   M. Welzl
Expires: July 2018   S. Islam
                                                     University of Oslo
                                                                 J. You
                                                                 Huawei
                                                       January 19, 2018

                    TCP Control Block Interdependence
                     draft-touch-tcpm-2140bis-03.txt

Status of this Memo

This Internet-Draft will expire on July 19, 2018.

Copyright Notice

Abstract

This memo describes interdependent TCP control blocks, where part of the TCP state is shared among similar concurrent or consecutive connections. TCP state includes a combination of parameters, such as connection state, current round-trip time estimates, congestion control information, and process information. Most of this state is maintained on a per-connection basis in the TCP Control Block (TCB), but implementations can (and do) share certain TCB information across connections to the same host. Such sharing is intended to improve overall transient transport performance, while maintaining backward-compatibility with existing implementations. The sharing described herein is limited to only the TCB initialization and so has no effect on the long-term behavior of TCP after a connection has been established.

Table of Contents

1. Introduction

   TCP is a connection-oriented reliable transport protocol layered
   over IP [RFC793]. Each TCP connection maintains state, usually in a
   data structure called the TCP Control Block (TCB). The TCB contains
   information about the connection state, its associated local
   process, and feedback parameters about the connection's transmission
   properties. As originally specified and usually implemented, most
   TCB information is maintained on a per-connection basis. Some
   implementations can (and now do) share certain TCB information
   across connections to the same host. Such sharing is intended to
   lead to better overall transient performance, especially for
   numerous short-lived and simultaneous connections, as often used in
   the World-Wide Web [Be94],[Br02].

   This document discusses TCB state sharing that affects only the TCB
   initialization, and so has no effect on the long-term behavior of
   TCP after a connection has been established. Path information shared
   across SYN destination port numbers assumes that TCP segments having
   the same host-pair experience the same path properties, irrespective
   of TCP port numbers. The observations about TCB sharing in this
   document apply similarly to any protocol with congestion state,
   including SCTP [RFC4960] and DCCP [RFC4340], as well as for
   individual subflows in Multipath TCP [RFC6824].

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 RFC 2119 [RFC2119].

   In this document, these words will appear with that interpretation
   only when in ALL CAPS. Lower case uses of these words are not to be
   interpreted as carrying significance described in RFC 2119.

   In this document, the characters ">>" preceding an indented line(s)
   indicates a statement using the key words listed above. This
   convention aids reviewers in quickly identifying or finding the
   portions of this RFC covered by these keywords.

3. Terminology

   Host - a source or sink of TCP segments associated with a single IP
   address

   Host-pair - a pair of hosts and their corresponding IP addresses

   Path - an Internet path between the IP addresses of two hosts

4. The TCP Control Block (TCB)

   A TCB describes the data associated with each connection, i.e., with
   each association of a pair of applications across the network. The
   TCB contains at least the following information [RFC793]:

        Local process state
            pointers to send and receive buffers
            pointers to retransmission queue and current segment
            pointers to Internet Protocol (IP) PCB
        Per-connection shared state
            macro-state
                connection state
                timers
                flags
                local and remote host numbers and ports
                TCP option state
            micro-state
                send and receive window state (size*, current number)
                round-trip time and variance
                cong. window size (snd_cwnd)*
                cong. window size threshold (ssthresh)*
                max window size seen*
                sendMSS#
                MMS_S#
                MMS_R#
                PMTU#
                round-trip time and variance#

   The per-connection information is shown as split into macro-state
   and micro-state, terminology borrowed from [Co91]. Macro-state
   describes the finite state machine; we include the endpoint numbers
   and components (timers, flags) used to help maintain that state.
   Macro-state describes the protocol for establishing and maintaining
   shared state about the connection. Micro-state describes the
   protocol after a connection has been established, to maintain the
   reliability and congestion control of the data transferred in the
   connection.

We further distinguish two other classes of shared micro-state that
are associated more with host-pairs than with application pairs. One
class is clearly host-pair dependent (#, e.g., MSS, MMS, PMTU, RTT),
and the other is host-pair dependent in its aggregate (*, e.g.,
congestion window information, current window sizes, etc.).

5. TCB Interdependence

   There are two cases of TCB interdependence. Temporal sharing occurs
   when the TCB of an earlier (now CLOSED) connection to a host is used
   to initialize some parameters of a new connection to that same host,
   i.e., in sequence. Ensemble sharing occurs when a currently active
   connection to a host is used to initialize another (concurrent)
   connection to that host.

6. An Example of Temporal Sharing

   The TCB data cache is accessed in two ways: it is read to initialize
   new TCBs and written when more current per-host state is available.
   New TCBs are initialized using context from past connections as
   follows:

            TEMPORAL SHARING - TCB Initialization

            Safe?   Cached TCB   New TCB
            ----
            yes   old_MMS_S   old_MMS_S or not cached

            yes   old_MMS_R   old_MMS_R or not cached

            yes   old_sendMSS   old_sendMSS

            yes   old_PMTU   old_PMTU

            TBD   old_RTT   old_RTT

            TBD   old_RTTvar   old_RTTvar

            varies old_option   (option specific)

            TBD   old_ssthresh   old_ssthresh

            TBD   old_snd_cwnd   old_snd_cwnd

   Table entries indicate which are considered to be safe to share
   temporally. The other entries are discussed in section 8.

Most cached TCB values are updated when a connection closes. The
exceptions are MMS_R and MMS_S, which are reported by IP [RFC1122],
PMTU which is updated after Path MTU Discovery
[RFC1191][RFC4821][RFC8201], and sendMSS, which is updated if the
MSS option is received in the TCP SYN header.

Sharing sendMSS information affects only data in the SYN of the next
connection, because sendMSS information is typically included in
most TCP SYN segments. Caching PMTU can accelerate the efficiency of
PMTUD, but can also result in black-holing until corrected if in
error. Caching MMS_R and MMS_S may be of little direct value as they
are reported by the local IP stack anyway.

[TBD - complete this section with details for TFO and other options
whose state may, must, or must not be shared] The way in which other
TCP option state can be shared depends on the details of that
option. E.g., TFO state includes the TCP Fast Open Cookie [RFC7413]
or, in case TFO fails, a negative TCP Fast Open response (from [RFC
7413]: "The client MUST cache negative responses from the server in
order to avoid potential connection failures. Negative responses
include the server not acknowledging the data in the SYN, ICMP error
messages, and (most importantly) no response (SYN-ACK) from the
server at all, i.e., connection timeout."). TFOinfo is cached when a
connection is established.

Other TCP option state might not be as readily cached. E.g., TCP-AO
[RFC5925] success or failure between a host pair for a single SYN
destination port might be usefully cached. TCP-AO success or failure
to other SYN destination ports on that host pair is never useful to
cache because TCP-AO security parameters can vary per service.

The table below gives an overview of option-specific information
that is considered safe to share.

        TEMPORAL SHARING - Option info

        Cached   New
        ----
        old_TFO_Cookie   old_TFO_Cookie

        old_TFO_Failure   old_TFO_Failure

TEMPORAL SHARING - Cache Updates

Safe?	Cached TCB	Current TCB	when?	New Cached TCB
yes	old_MMS_S	curr_ MMS_S	OPEN	curr MMS_S
yes	old_MMS_R	curr_ MMS_R	OPEN	curr_MMS_R
yes	old_sendMSS	curr_sendMSS	MSSopt	curr_sendMSS
yes	old_PMTU	curr_PMTU	PMTUD	curr_PMTU
TBD	old_RTT	curr_RTT	CLOSE	merge(curr,old)
TBD	old_RTTvar	curr_RTTvar	CLOSE	merge(curr,old)
varies	old_option	curr option	ESTAB	(depends on option)
TBD	old_ssthresh	curr_ssthresh	CLOSE	merge(curr,old)
TBD	old_snd_cwnd	curr_snd_cwnd	CLOSE	merge(curr,old)

Caching PMTU and sendMSS is trivial; reported values are cached, and
the most recent values are used. The cache is updated when the MSS
option is received in a SYN or after PMTUD (i.e., when an ICMPv4
Fragmentation Needed [RFC1191] or ICMPv6 Packet Too Big message is
received [RFC8201] or the equivalent is inferred, e.g. as from
PLPMTUD [RFC4821]), respectively, so the cache always has the most
recent values from any connection. For sendMSS, the cache is
consulted only at connection establishment and not otherwise
updated, which means that MSS options do not affect current
connections. The default sendMSS is never saved; only reported MSS
values update the cache, so an explicit override is required to
reduce the sendMSS. There is no particular benefit to caching MMS_S
and MMS_R as these are reported by the local IP stack.

TCP options are copied or merged depending on the details of each
option. E.g., TFO state is updated when a connection is established
and read before establishing a new connection.

RTT values are updated by a more complicated mechanism
[RFC1644][Ja86]. Dynamic RTT estimation requires a sequence of RTT
measurements. As a result, the cached RTT (and its variance) is an
average of its previous value with the contents of the currently
active TCB for that host, when a TCB is closed. RTT values are
updated only when a connection is closed. The method for merging old
and current values needs to attempt to reduce the transient for new

connections. [THESE MERGE FUNCTIONS NEED TO BE SPECIFIED, considering e.g. [DM16] - TBD].

The updates for RTT, RTTvar and ssthresh rely on existing information, i.e., old values. Should no such values exist, the current values are cached instead.

```
              TEMPORAL SHARING - Option info Updates

Cached           Current         when?   New Cached
------------------------------------------------------------
old_TFO_Cookie   old_TFO_Cookie   ESTAB   old_TFO_Cookie

old_TFO_Failure old_TFO_Failure  ESTAB   old_TFO_Failure
```

7. An Example of Ensemble Sharing

   Sharing cached TCB data across concurrent connections requires attention to the aggregate nature of some of the shared state. For example, although MSS and RTT values can be shared by copying, it may not be appropriate to copy congestion window or ssthresh information (see section 8 for a discussion of congestion window or ssthresh sharing).

```
           ENSEMBLE SHARING - TCB Initialization

           Safe?   Cached TCB       New TCB
           -----------------------------------------
           yes     old_MMS_S        old_MMS_S

           yes     old_MMS_R        old_MMS_R

           yes     old_sendMSS      old_sendMSS

           yes     old_PMTU         old_PMTU           TBD
old_RTT        old_RTT

           TBD     old_RTTvar       old_RTTvar

           TBD     old_option       (option-specific)
```

   Table entries indicate which are considered to be safe to share across an ensemble. The other entries are discussed in section 8.

The table below gives an overview of option-specific information
that is considered safe to share.

```
            ENSEMBLE SHARING - Option info

      Cached                 New
      ----------------------------------------
      old_TFO_Cookie         old_TFO_Cookie

      old_TFO_Failure        old_TFO_Failure
```

```
            ENSEMBLE SHARING - Cache Updates
```

Safe?	Cached TCB	Current TCB	when?	New Cached TCB
yes	old_MMS_S	curr_MMS_S	OPEN	curr_MMS_S
yes	old_MMS_R	curr_MMS_R	OPEN	curr_MMS_R
yes	old_sendMSS	curr_sendMSS	MSSopt	curr_sendMSS
yes	old_PMTU	curr_PMTU	PMTUD /PLPMTUD	curr_PMTU
TBD	old_RTT	curr_RTT	update	rtt_update(old,cur)
TBD	old_RTTvar	curr_RTTvar	update	rtt_update(old,cur)
varies	old_option	curr option	(depends)	(option specific)

For ensemble sharing, TCB information should be cached as early as
possible, sometimes before a connection is closed. Otherwise,
opening multiple concurrent connections may not result in TCB data
sharing if no connection closes before others open. The amount of
work involved in updating the aggregate average should be minimized,
but the resulting value should be equivalent to having all values
measured within a single connection. The function "rtt_update" in
the ensemble sharing table indicates this operation, which occurs
whenever the RTT would have been updated in the individual TCP
connection. As a result, the cache contains the shared RTT
variables, which no longer need to reside in the TCB [Ja86].

Congestion window size and ssthresh aggregation are more complicated
in the concurrent case. When there is an ensemble of connections, we

need to decide how that ensemble would have shared these variables, in order to derive initial values for new TCBs.

```
                 ENSEMBLE SHARING - Option info Updates

    Cached          Current        when?   New Cached
    ----------------------------------------------------------------
    old_TFO_Cookie  old_TFO_Cookie   ESTAB   old_TFO_Cookie

    old_TFO_Failure old_TFO_Failure  ESTAB   old_TFO_Failure
```

Any assumption of this sharing can be incorrect, including this one, because identical endpoint address pairs may not share network paths. In current implementations, new congestion windows are set at an initial value of 4-10 segments [RFC3390][RFC6928], so that the sum of the current windows is increased for any new connection. This can have detrimental consequences where several connections share a highly congested link.

There are several ways to initialize the congestion window in a new TCB among an ensemble of current connections to a host, as shown below. Current TCP implementations initialize it to four segments as standard [rfc3390] and 10 segments experimentally [RFC6928] and T/TCP hinted that it should be initialized to the old window size [RFC1644]. In the former cases, the assumption is that new connections should behave as conservatively as possible. In the latter T/TCP case, no accommodation is made for concurrent aggregate behavior.

In either case, the sum of window sizes can increase, rather than remain constant. A different approach is to give each pending connection its "fair share" of the available congestion window, and let the connections balance from there. The assumption we make here is that new connections are implicit requests for an equal share of available link bandwidth, which should be granted at the expense of current connections. [TBD - a new method for safe congestion sharing will be described]

8. Compatibility Issues

For the congestion and current window information, the initial values computed by TCB interdependence may not be consistent with the long-term aggregate behavior of a set of concurrent connections between the same endpoints. Under conventional TCP congestion control, if a single existing connection has converged to a congestion window of 40 segments, two newly joining concurrent

connections assume initial windows of 10 segments [RFC6928], and the
current connection's window doesn't decrease to accommodate this
additional load and connections can mutually interfere. One example
of this is seen on low-bandwidth, high-delay links, where concurrent
connections supporting Web traffic can collide because their initial
windows were too large, even when set at one segment.

[TBD - this paragraph needs to be revised based on new
recommendations] Under TCB interdependence, all three connections
could change to use a congestion window of 12 (rounded down to an
even number from 13.33, i.e., 40/3). This would include both
increasing the initial window of the new connections (vs. current
recommendations [RFC6928]) and decreasing the congestion window of
the current connection (from 40 down to 12). This gives the new
connections a larger initial window than allowed by [RFC6928], but
maintains the aggregate. Depending on whether the previous
connections were in steady-state, this can result in more bursty
behavior, e.g., when previous connections are idle and new
connections commence with a large amount of available data to
transmit. Additionally, reducing the congestion window of an
existing connection needs to account for the number of packets that
are already in flight.

Because this proposal attempts to anticipate the aggregate steady-
state values of TCB state among a group or over time, it should
avoid the transient effects of new connections. In addition, because
it considers the ensemble and temporal properties of those
aggregates, it should also prevent the transients of short-lived or
multiple concurrent connections from adversely affecting the overall
network performance. There have been ongoing analysis and
experiments to validate these assumptions. For example, [Ph12]
recommends to only cache ssthresh for temporal sharing when flows
are long. Sharing ssthresh between short flows can deteriorate the
overall performance of individual connections[Ph12, Nd16], although
this may benefit overall network performance. [TBD - the details of
this issue need to be summarized and clarified herein].

[TBD - placeholder for corresponding RTT discussion]

Due to mechanisms like ECMP and LAG [RFC7424], TCP connections
sharing the same host-pair may not always share the same path. This
does not matter for host-specific information such as RWIN and TCP
option state, such as TFOinfo. When TCB information is shared across
different SYN destination ports, path-related information can be
incorrect; however, the impact of this error is potentially
diminished if (as discussed here) TCB sharing affects only the
transient event of a connection start or if TCB information is

shared only within connections to the same SYN destination port. In case of Temporal Sharing, TCB information could also become invalid over time. Because this is similar to the case when a connection becomes idle, mechanisms that address idle TCP connections (e.g., [RFC7661]) could also be applied to TCB cache management.

There may be additional considerations to the way in which TCB interdependence rebalances congestion feedback among the current connections, e.g., it may be appropriate to consider the impact of a connection being in Fast Recovery [RFC5861] or some other similar unusual feedback state, e.g., as inhibiting or affecting the calculations described herein.

TCP is sometimes used in situations where packets of the same host-pair always take the same path. Because ECMP and LAG examine TCP port numbers, they may not be supported when TCP segments are encapsulated, encrypted, or altered - for example, some Virtual Private Networks (VPNs) are known to use proprietary UDP encapsulation methods. Similarly, they cannot operate when the TCP header is encrypted, e.g., when using IPsec ESP. TCB interdependence among the entire set sharing the same endpoint IP addresses should work without problems under these circumstances. Moreover, measures to increase the probability that connections use the same path could be applied: e.g., the connections could be given the same IPv6 flow label. TCB interdependence can also be extended to sets of host IP address pairs that share the same network path conditions, such as when a group of addresses is on the same LAN (see Section 9).

It can be wrong to share TCB information between TCP connections on the same host as identified by the IP address if an IP address is assigned to a new host (e.g., IP address spinning, as is used by ISPs to inhibit running servers). It can be wrong if Network Address (and Port) Translation (NA(P)T) [RFC2663] or any other IP sharing mechanism is used. Such mechanisms are less likely to be used with IPv6. Other methods to identify a host could also be considered to make correct TCB sharing more likely. Moreover, some TCB information is about dominant path properties rather than the specific host. IP addresses may differ, yet the relevant part of the path may be the same.

9. Implications

There are several implications to incorporating TCB interdependence in TCP implementations. First, it may reduce the need for application-layer multiplexing for performance enhancement [RFC7231]. Protocols like HTTP/2 [RFC7540] avoid connection reestablishment costs by serializing or multiplexing a set of per-

host connections across a single TCP connection. This avoids TCP's
per-connection OPEN handshake and also avoids recomputing MSS, RTT,
and congestion windows. By avoiding the so-called, "slow-start
restart," performance can be optimized. TCB interdependece can
provide the "slow-start restart avoidance" of multiplexing, without
requiring a multiplexing mechanism at the application layer.

TCB interdependence pushes some of the TCP implementation from the
traditional transport layer (in the ISO model), to the network
layer. This acknowledges that some state is in fact per-host-pair or
can be per-path as indicated solely by that host-pair. Transport
protocols typically manage per-application-pair associations (per
stream), and network protocols manage per-host-pair and path
associations (routing). Round-trip time, MSS, and congestion
information could be more appropriately handled in a network-layer
fashion, aggregated among concurrent connections, and shared across
connection instances [RFC3124].

An earlier version of RTT sharing suggested implementing RTT state
at the IP layer, rather than at the TCP layer [Ja86]. Our
observations are for sharing state among TCP connections, which
avoids some of the difficulties in an IP-layer solution. One such
problem is determining the associated prior outgoing packet for an
incoming packet, to infer RTT from the exchange. Because RTTs are
still determined inside the TCP layer, this is simpler than at the
IP layer. This is a case where information should be computed at the
transport layer, but could be shared at the network layer.

Per-host-pair associations are not the limit of these techniques. It
is possible that TCBs could be similarly shared between hosts on a
subnet or within a cluster, because the predominant path can be
subnet-subnet, rather than host-host. Additionally, TCB
interdependence can be applied to any protocol with congestion
state, including SCTP [RFC4960] and DCCP [RFC4340], as well as for
individual subflows in Multipath TCP [RFC6824].

There may be other information that can be shared between concurrent
connections. For example, knowing that another connection has just
tried to expand its window size and failed, a connection may not
attempt to do the same for some period. The idea is that existing
TCP implementations infer the behavior of all competing connections,
including those within the same host or subnet. One possible
optimization is to make that implicit feedback explicit, via
extended information associated with the endpoint IP address and its
TCP implementation, rather than per-connection state in the TCB.

Like its initial version in 1997, this document's approach to TCB interdependence focuses on sharing a set of TCBs by updating the TCB state to reduce the impact of transients when connections begin or end. Other mechanisms have since been proposed to continuously share information between all ongoing communication (including connectionless protocols), updating the congestion state during any congestion-related event (e.g., timeout, loss confirmation, etc.) [RFC3124]. By dealing exclusively with transients, TCB interdependence is more likely to exhibit the same behavior as unmodified, independent TCP connections.

10. Implementation Observations

The observation that some TCB state is host-pair specific rather than application-pair dependent is not new and is a common engineering decision in layered protocol implementations. A discussion of sharing RTT information among protocols layered over IP, including UDP and TCP, occurred in [Ja86]. Although now deprecated, T/TCP was the first to propose using caches in order to maintain TCB states (see Appendix A for more information).

The table below describes the current implementation status for some TCB information in Linux kernel version 4.6, FreeBSD 10 and Windows (as of October 2016). In the table, "shared" only refers to temporal sharing.

```
TCB data      Status
------------------------------------------------------------
old MMS_S     Not shared

old MMS_R     Not shared

old_sendMSS   Cached and shared in Linux (MSS)

old PMTU      Cached and shared in FreeBSD and Windows (PMTU)

old_RTT       Cached and shared in FreeBSD and Linux

old_RTTvar    Cached and shared in FreeBSD

old TFOinfo   Cached and shared in Linux and Windows

old_snd_cwnd  Not shared

old_ssthresh  Cached and shared in FreeBSD and Linux:
              FreeBSD: arithmetic
              mean of ssthresh and previous value if
              a previous value exists;
              Linux: depending on state,
              max(cwnd/2, ssthresh) in most cases
```

11. Security Considerations

   These suggested implementation enhancements do not have additional
   ramifications for explicit attacks. These enhancements may be
   susceptible to denial-of-service attacks if not otherwise secured.
   For example, an application can open a connection and set its window
   size to zero, denying service to any other subsequent connection
   between those hosts.

   TCB sharing may be susceptible to denial-of-service attacks,
   wherever the TCB is shared, between connections in a single host, or
   between hosts if TCB sharing is implemented within a subnet (see
   Implications section). Some shared TCB parameters are used only to
   create new TCBs, others are shared among the TCBs of ongoing
   connections. New connections can join the ongoing set, e.g., to
   optimize send window size among a set of connections to the same
   host.

   Attacks on parameters used only for initialization affect only the
   transient performance of a TCP connection. For short connections,
   the performance ramification can approach that of a denial-of-

service attack. E.g., if an application changes its TCB to have a
false and small window size, subsequent connections would experience
performance degradation until their window grew appropriately.

The solution is to limit the effect of compromised TCB values. TCBs
are compromised when they are modified directly by an application or
transmitted between hosts via unauthenticated means (e.g., by using
a dirty flag). TCBs that are not compromised by application
modification do not have any unique security ramifications. Note
that the proposed parameters for TCB sharing are not currently
modifiable by an application.

All shared TCBs MUST be validated against default minimum parameters
before used for new connections. This validation would not impact
performance, because it occurs only at TCB initialization. This
limits the effect of attacks on new connections to reducing the
benefit of TCB sharing, resulting in the current default TCP
performance. For ongoing connections, the effect of incoming packets
on shared information should be both limited and validated against
constraints before use. This is a beneficial precaution for existing
TCP implementations as well.

TCBs modified by an application SHOULD NOT be shared, unless the new
connection sharing the compromised information has been given
explicit permission to use such information by the connection API.
No mechanism for that indication currently exists, but it could be
supported by an augmented API. This sharing restriction SHOULD be
implemented in both the host and the subnet. Sharing on a subnet
SHOULD utilize authentication to prevent undetected tampering of
shared TCB parameters. These restrictions limit the security impact
of modified TCBs both for connection initialization and for ongoing
connections.

Finally, shared values MUST be limited to performance factors only.
Other information, such as TCP sequence numbers, when shared, are
already known to compromise security.

12. IANA Considerations

There are no IANA implications or requests in this document.

This section should be removed upon final publication as an RFC.

13. References

13.1. Normative References

   [RFC793]   Postel, Jon, "Transmission Control Protocol," Network
              Working Group RFC-793/STD-7, ISI, Sept. 1981.

   [RFC1191] Mogul, J., Deering, S., "Path MTU Discovery," RFC 1191,
              Nov. 1990.

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
              Requirement Levels", BCP 14, RFC 2119, March 1997.

   [RFC4821] Mathis, M., Heffner, J., "Packetization Layer Path MTU
              Discovery," RFC 4821, Mar. 2007.

   [RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., Jain, A., "TCP Fast
              Open", RFC 7413, Dec. 2014.

   [RFC8201] McCann, J., Deering. S., Mogul, J., Hinden, R. (Ed.),
              "Path MTU Discovery for IP version 6," RFC 8201, Jul.
              2017.

13.2. Informative References

   [Br02]   Brownlee, N. and K. Claffy, "Understanding Internet
              Traffic Streams: Dragonflies and Tortoises", IEEE
              Communications Magazine p110-117, 2002.

   [Be94]   Berners-Lee, T., et al., "The World-Wide Web,"
              Communications of the ACM, V37, Aug. 1994, pp. 76-82.

   [Br94]   Braden, B., "T/TCP -- Transaction TCP: Source Changes for
              Sun OS 4.1.3,", Release 1.0, USC/ISI, September 14, 1994.

   [Co91]   Comer, D., Stevens, D., Internetworking with TCP/IP, V2,
              Prentice-Hall, NJ, 1991.

   [FreeBSD] FreeBSD source code, Release 2.10, http://www.freebsd.org/

   [Ja86]   Jacobson, V., (mail to public list "tcp-ip", no archive
              found), 1986.

   [Nd16]   Dukkipati, N., Yuchung C., and Amin V., "Research
              Impacting the Practice of Congestion Control." ACM SIGCOMM
              CCR (editorial).

   [DM16]   Matz, D., "Optimize TCP's Minimum Retransmission Timeout
              for Low Latency Environments", Master's thesis, Technical
              University Munich, 2016.

   [Ph12]   Hurtig, P., Brunstrom, A., "Enhanced metric caching for
              short TCP flows," 2012 IEEE International Conference on
              Communications (ICC), Ottawa, ON, 2012, pp. 1209-1213.

   [RFC1122] Braden, R. (ed), "Requirements for Internet Hosts --
              Communication Layers", RFC-1122, Oct. 1989.

   [RFC1644] Braden, R., "T/TCP -- TCP Extensions for Transactions
              Functional Specification," RFC-1644, July 1994.

   [RFC1379] Braden, R., "Transaction TCP -- Concepts," RFC-1379,
              September 1992.

   [RFC2663] Srisuresh, P., Holdrege, M., "IP Network Address
              Translator (NAT) Terminology and Considerations", RFC-
              2663, August 1999.

   [RFC3390] Allman, M., Floyd, S., Partridge, C., "Increasing TCP's
              Initial Window," RFC 3390, Oct. 2002.

   [RFC7231] Fielding, R., J. Reshke, Eds., "HTTP/1.1 Semantics and
              Content," RFC-7231, June 2014.

   [RFC3124] Balakrishnan, H., Seshan, S., "The Congestion Manager,"
              RFC 3124, June 2001.

   [RFC4340] Kohler, E., Handley, M., Floyd, S., "Datagram Congestion
              Control Protocol (DCCP)," RFC 4340, Mar. 2006.

   [RFC4960] Stewart, R., (Ed.), "Stream Control Transmission
              Protocol," RFC4960, Sept. 2007.

   [RFC5861] Allman, M., Paxson, V., Blanton, E., "TCP Congestion
              Control," RFC 5861, Sept. 2009.

   [RFC5925] Touch, J., Mankin, A., Bonica, R., "The TCP Authentication
              Option," RFC 5925, June 2010.

   [RFC6824] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., "TCP
              Extensions for Multipath Operation with Multiple
              Addresses," RFC 6824, Jan. 2013.

   [RFC6928] Chu, J., Dukkipati, N., Cheng, Y., Mathis, M., "Increasing
             TCP's Initial Window," RFC 6928, Apr. 2013.

   [RFC7424] Krishnan, R., Yong, L., Ghanwani, A., So, N., Khasnabish,
             B., "Mechanisms for Optimizing Link Aggregation Group
             (LAG) and Equal-Cost Multipath (ECMP) Component Link
             Utilization in Networks", RFC 7424, Jan. 2015

   [RFC7540] Belshe, M., Peon, R., Thomson, M., "Hypertext Transfer
             Protocol Version 2 (HTTP/2)", RFC 7540, May 2015.

   [RFC7661] Fairhurst, G., Sathiaseelan, A., Secchi, R., "Updating TCP
             to Support Rate-Limited Traffic", RFC 7661, Oct. 2015

14. Acknowledgments

   This document was prepared using 2-Word-v2.0.template.dot.

15. Change log

   03:

      - Updated Touch's affiliation and address information

   02:

      - Stated that our OS implementation overview table only covers
      temporal sharing.

      - Correctly reflected sharing of old_RTT in Linux in the
   implementation overview table.

      - Marked entries that are considered safe to share with an
   asterisk (suggestion was to split the table)

      - Discussed correct host identification: NATs may make IP
   addresses the wrong input, could e.g. use HTTP cookie.

   - Included MMS_S and MMS_R from RFC1122; fixed the use of MSS and
   MTU

   - Added information about option sharing, listed options in the
   appendix

Authors' Addresses

   Joe Touch

   Manhattan Beach, CA 90266
   USA

   Phone: +1 (310) 560-0334
   Email: touch@strayalpha.com

   Michael Welzl
   University of Oslo
   PO Box 1080 Blindern
   Oslo N-0316
   Norway

   Phone: +47 22 85 24 20
   Email: michawe@ifi.uio.no

   Safiqul Islam
   University of Oslo
   PO Box 1080 Blindern
   Oslo N-0316
   Norway

   Phone: +47 22 84 08 37
   Email: safiquli@ifi.uio.no

Jianjie You
Huawei
101 Software Avenue, Yuhua District
Nanjing 210012
China

Email: youjianjie@huawei.com

16. Appendix A: TCB sharing history

   T/TCP proposed using caches to maintain TCB information across
   instances (temporal sharing), e.g., smoothed RTT, RTT variance,
   congestion avoidance threshold, and MSS [RFC1644]. These values were
   in addition to connection counts used by T/TCP to accelerate data
   delivery prior to the full three-way handshake during an OPEN. The
   goal was to aggregate TCB components where they reflect one
   association - that of the host-pair, rather than artificially
   separating those components by connection.

   At least one T/TCP implementation saved the MSS and aggregated the
   RTT parameters across multiple connections, but omitted caching the
   congestion window information [Br94], as originally specified in
   [RFC1379]. Some T/TCP implementations immediately updated MSS when
   the TCP MSS header option was received [Br94], although this was not
   addressed specifically in the concepts or functional specification
   [RFC1379][RFC1644]. In later T/TCP implementations, RTT values were
   updated only after a CLOSE, which does not benefit concurrent
   sessions.

   Temporal sharing of cached TCB data was originally implemented in
   the SunOS 4.1.3 T/TCP extensions [Br94] and the FreeBSD port of same
   [FreeBSD]. As mentioned before, only the MSS and RTT parameters were
   cached, as originally specified in [RFC1379]. Later discussion of
   T/TCP suggested including congestion control parameters in this
   cache [RFC1644].

17. Appendix B: Options

   In addition to the options that can be cached and shared, this memo
   also lists all options for which state should *not* be kept. This
   list is meant to avoid work duplication and should be removed upon
   publication.

Obsolete (MUST NOT keep state):

ECHO

ECHO REPLY

PO Conn permitted

PO service profile

CC

CC.NEW

CC.ECHO

Alt CS req

Alt CS data

No state to keep:

EOL

NOP

WS

SACK

TS

MD5

TCP-AO

EXP1

EXP2

MUST NOT keep state:

Skeeter (DH exchange - might be obsolete, though)

Bubba (DH exchange - might really be obsolete, though)

Trailer CS

SCPS capabilities

S-NACK

Records boundaries

Corruption experienced

SNAP

TCP Compression

Quickstart response

UTO

MPTCP (can we cache when this fails?)

TFO success

MAY keep state:

MSS

TFO failure (so we don't try again, since it's optional)

MUST keep state:

TFP cookie (if TFO succeeded in the past)

                          TCP Low Latency Option
                     draft-wang-tcpm-low-latency-opt-00

Abstract

   This document specifies the TCP Low Latency option, which TCP
   connections can use during the connection establishment handshake to
   communicate extra parameters that can improve performance in low-
   latency environments. With the first such parameter, a TCP data
   receiver can advertise a hint about the Maximum ACK Delay (MAD) it
   will schedule for its own delayed ACK mechanism. This enables the
   TCP data sender to achieve lower latencies during loss recovery by
   using the Maximum ACK Delay advertised by the remote receiver to help
   compute retransmission timeouts that are potentially much lower than
   would otherwise be feasible. The Low Latency option is extensible,
   and later versions of this draft will introduce other mechanisms,
   including TCP timestamps with a finer granularity than those
   supported by RFC 7323.

Copyright Notice

1. Introduction

   TCP receivers typically implement a delayed ACK algorithm, as
   specified in [RFC1122] Sec 4.2.3.2; as summarized in [RFC5681] sec
   4.2, "an ACK SHOULD be generated for at least every second full-sized
   segment, and MUST be generated within 500 ms of the arrival of the
   first unacknowledged packet." In practice, many widely-deployed
   implementations have tended to delay ACKs by up to roughly 200ms.
   This is probably a historical artifact inherited from the 200ms "fast
   timeout" mechanism in the BSD TCP implementation from the late 1980s
   [WS95].

   As a result, to avoid spurious timeouts due to delayed ACKs, widely-
   deployed TCP sender implementations have adapted to this delayed ACK
   behavior by constraining retransmission timeout (RTO) values to be at
   least 200ms.

   Unfortunately, this 200ms value is 2000x the typical RTT of today's
   commodity datacenter networks (which are typically below 100
   microseconds). So senders constraining RTOs to be at least 200ms are
   paying a latency penalty much higher than the RTT in such
   environments.

   The TCP Low Latency option enables a TCP data receiver to advertise a
   hint about the Maximum ACK Delay (MAD) it will schedule for its own
   delayed ACK mechanism. The receiver specifies the MAD value in the
   Low Latency option because the value that is feasible can be quite
   different for different receivers, based on the CPU's speed, CPU and
   network workloads, and OS-specific constraints on minimum supported
   timer granularity.

   This Low Latency option enables the TCP data sender to achieve lower
   latencies during loss recovery by using the Maximum ACK Delay

advertised by the remote receiver to help compute retransmission
timeouts that are potentially much lower than would otherwise be
feasible.

2. Terminology

The keywords "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].

In this document, "MAD" refers to the Maximum Ack Delay used by the
data receiver to delay TCP acknowledgments, and "minRTO" refers to
the Minimum Retransmit Timeout.

3. Detailed Protocol

3.1. TCP Low Latency Option

The Low Latency option is only valid in SYN or SYN/ACK packets during
the three way handshake. It MUST be ignored in other cases.

The format of the TCP Low Latency option is as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Kind       |     Length      |M u|        MAD       |     |
|                 |                 |A n|       Value      | Res |
|                 |                 |D i|      (10 bits)   |     |
|                 |                 |  t|                  |     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                     ...  Reserved  ...                        ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
Kind:       1 byte: value = IANA-assigned option number
Length:     1 byte: value = 4 (or longer in later versions)
MAD unit:   2 bits: indicates time unit for MAD value:
                    0: reserved
                    1: milliseconds
                    2: microseconds
                    3: nanoseconds
MAD value:  10 bits: indicates MAD value set on the host:
                     1 ... 1023: MAD value in the given units
                     0: no MAD value is specified
Reserved:   N>=4 bits: value = 0
```

   In order to support future extensions, the option is variable-length.
   Bits beyond those defined so far in IETF standards should be
   considered "reserved". TCP implementations MUST (a) set to zero any
   reserved bits they add for padding, and (b) ignore any reserved bits
   (whether they are set or not).

3.2. Overview

   The communication, starting from the TCP connection handshake, looks
   like the following:

```
TCP A (Active)                                  TCP B (Passive)
==============                                  ===============
CLOSED                                          LISTEN
#1 SYN-SENT        ----- <SYN,MAD=10ms>  ------> SYN-RCVD
                                                (Adjust RTO accordingly)
#2 ESTABLISHED     <---- <SYN,ACK,MAD=5ms> ----- SYN-RCVD
   (Adjust RTO accordingly)
#3 ESTABLISHED     -------<ACK>----------------> ESTABLISHED
#4 Send()          --------<DATA-1>------------> -
                                                 |
                                                 | Delay Ack < 5ms
                                                 |
                   <-------<ACK-1>------------   -
#5                                              Recv()

#6 Send()          ---------<DATA-2>-------------->
                       |
   RTO >= 5ms          |
                       |
                   ---------<DATA-2 retransmit>--->
                   <-------<ACK-2>----------------
#7                                              Recv()
```
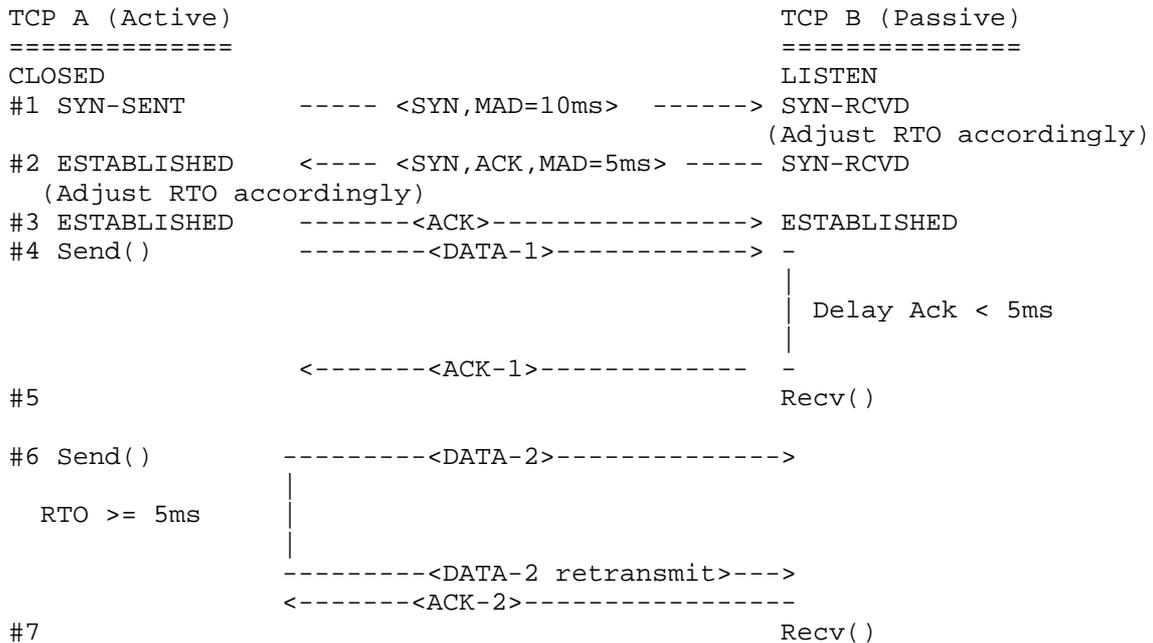
3.3. Configuring maximum ACK delay

   An implementation that supports the maximum ACK delay parameter MUST
   provide a user API to configure the maximum ACK delay for a specific
   connection or all TCP connections.

   o If the user does not specify a MAD value, then the implementation
      SHOULD NOT specify a MAD value in the Low Latency option.

   o If the user specifies a MAD value outside the range of ACK delay
      values supported by the implementation, then the implementation
      SHOULD allow the request to succeed, but SHOULD silently constrain
      the MAD value to be within the valid range (between the minimum
      and maximum ACK delay for the implementation). This is intended

          to allow applications to portably request a MAD value without
          needing special logic to search for a valid value.

     o If the specified connections are not in CLOSED or LISTEN states,
        the API SHOULD return an error and ignore the request to specify a
        MAD value.

     o Otherwise the implementation SHOULD use the user-specified value
        as the maximum timeout for the delayed ACK and the MAD value in
        the Low Latency option of the specified TCP connections.

     The exact design and implementation of such an API is intentionally
     left to the implementation. We discuss some examples in the
     appendix.

3.4. Announcing the maximum ACK delay

     o The maximum ACK delay is announced to the remote TCP endpoint by
        including a Low Latency option with a non-zero MAD value in the
        SYN or SYN/ACK packet. A "MAD value" field of 0 in the Low
        Latency option indicates that the sender is not specifying a MAD
        value.

     o If specified, then the MAD value in the Low Latency option MUST be
        set, as close as possible, to the implementation's actual delayed
        ACK timeout for the connection. Note that the actual maximum
        delayed ACK timeout of the connection may be larger than the
        actual user specified value because of implementation constraints
        (e.g. timer granularity limitations).

     o If the user has specified a MAD value for an active connection,
        then the active open side SHOULD include a Low Latency option with
        a MAD value in the SYN packet.

     o If the user has specified a MAD value for a passive connection,
        and the passive side has received at least one SYN packet with a
        Low Latency option with a valid MAD value, then the passive open
        side SHOULD return its MAD value in the Low Latency option.

3.5. Adjusting TCP retransmission timeouts

     If the MAD value advertised in a received Low Latency option is 0, or
     greater than the default maximum ACK delay of 200ms, then the option
     SHOULD be ignored and no further action is needed.

     Otherwise the (data) sender MAY use the maximum delayed ACK
     advertised by the receiver to adjust the sender's RTO calculation.
     Specifically, if the sender implements an RTO calculation based on

[RFC6298], it MAY replace the 1 second lower-bound specified in step
2.4 in Section 2 with the value of the maximum ACK delay advertised
in the Low Latency option, so that the calculation becomes:

RTO <- SRTT + max(G, K*RTTVAR) + max(G, max_ACK_delay)

instead of

RTO <- max(SRTT + max(G, K*RTTVAR), 1 second) /* [RFC6298] */

Here we use the notation of [RFC6298], including SRTT (smoothed
round-trip time), RTTVAR (round-trip time variation), and G (clock
granularity).

Also, if the sender also implements [draft-ietf-tcpm-rack] then it
SHOULD replace the maximum delayed ACK parameter (WCDelAckT) with the
max_ACK_delay specified in the Low Latency option.

Using the MAD value in the RTO calculation helps senders reduce the
RTO significantly while still avoiding spurious retransmissions due
to delayed acks. With this new algorithm, the RTO can be drastically
shortened in most environments where the receiver advertises a MAD.
In particular, in data center environments the RTO can often be
reduced from more than one second to single-digit milliseconds.
Using the MAD to reduce the RTO can improve performance and thus
mitigate TCP incast issues. More details are provided in the
following Related work section.

4. Related work

   Several research papers have shown that reducing the minimum
   retransmission timeout (minRTO) significantly improves the
   performance of TCP in the datacenter, by mitigating the effect of TCP
   timeouts. As a result, this can mitigate TCP incast issues.

   o In "Attaining the Promise and Avoiding the Pitfalls of TCP in the
      Datacenter" [JS15], the authors show that reducing minRTO from
      200ms to 5ms greatly reduced the impact of TCP incast issues.

   o In "Understanding TCP incast throughput collapse in datacenter
      networks" [CG09], the authors show significant improvement in
      goodput when reducing minRTO.

   o In "Measurement and Analysis of TCP Throughput Collapse in
      Cluster-based Storage Systems" [PK07], the authors show that
      reducing minRTO from 200 milliseconds to 200 microseconds improved
      goodput by an order of magnitude in some data center scenarios
      they evaluated.

o In "Safe and Effective Fine-grained TCP Retransmissions for
   Datacenter Communication" [VP09], the authors point out that the
   imbalance between the TCP minRTO and datacenter latencies can
   result in poor performance for applications sensitive to
   millisecond-scale delays in query response times. In simulations
   of datacenter scenarios they show that goodput drops when
   increasing minRTO above 1ms. Moreover, in some data center
   scenarios the default minRTO of 200ms results in nearly 2 orders
   of magnitude lower throughput compared to a minRTO of 1ms.

o In Google data centers a TCP option mechanism equivalent to the
   Low Latency option's MAD parameter has been used since 2005, and
   the TCP minRTO has been set to 5ms by default since 2013 [CC16].

5. Middlebox Considerations

   The new Low Latency option might expose some middlebox issues:

o Middleboxes could drop SYNs with a Low Latency option in the case
   where it treats the Low Latency option as an unknown option.
   However, this happens fairly rarely according to "Is it still
   possible to extend TCP?" [HN11], table 3.

o In case middleboxes alter the content in the Low Latency option,
   the receiver SHOULD do a sanity check on the MAD value included in
   the Low Latency option to verify it is less than or equal to the
   default maximum ACK delay of 200ms. As explained earlier, it is
   not practical for users to set MAD value greater than default. So
   it is safe to consider a MAD value greater than default as a
   result of a bad user configuration or a malfunctioning middlebox
   and ignore the Low Latency option completely in such cases.

6. Security Considerations

   TBD

7. IANA Considerations

   As no official option number has been issued for the new Low Latency
   option by IANA yet, experimental option 254 per [RFC6994] with magic
   number 0xF990 (16 bits) is used for now.

   The option format with experimental ID is as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Kind       |      Length     |   RFC 6994 Experiment ID   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|M u|       MAD       |       |
|A n|      Value      |  Res  |            ...
|D i|    (10 bits)    |       |
|   t|                |       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
Kind:          1 byte: value = 254
Length:        1 byte: value = 6 (or longer in later versions)
Experiment ID: 2 bytes: value = 0xF990
MAD unit:      2 bits: indicates time unit for MAD value:
                        0: reserved
                        1: milliseconds
                        2: microseconds
                        3: nanoseconds
MAD value:     10 bits: indicates MAD value set on the host:
                        1 ... 1023: MAD value in the given units
                        0: no MAD value is specified
Reserved:      N>=4 bits: value = 0
```

We will migrate to using the official option number for the Low
Latency option after IANA has assigned one.

8. Appendix

8.1. Example user API in Linux to configure maximum ACK delay

8.1.1. Per-route MAD configuration API

   A new configuration option called "mad" will be added to the "ip"
   command line tool in the iproute2 package. Users can use this to
   configure a per-route MAD value like the following:

   ip route add 10.1.2.0/24 dev eth0 scope link src 10.1.2.123 mad 5ms

   This configures all connections destined to 10.1.2.0/24 to have a MAD
   value of 5ms. When implementing this new MAD option field, the "ip"
   command line tool will verify that the provided MAD parameter is less
   than or equal to the default MAD value of 200ms. If the MAD is
   invalid then the ip route command will ignore the command and report
   an error to user.

   Newly-created TCP sockets have the default 200ms MAD value. When a
   TCP connection is opened, it SHOULD consult the ip routing table to
   check if there is any configured MAD value for the route. If so, the

implementation copies the route's MAD value to the connection's MAD value.

This per-route configuration will mostly be used by network administrators when configuring routes on the host.

8.1.2. MAD Socket option API

Socket options provide per-connection configuration parameters. To allow per-connection configuration of the MAD value in the Low Latency option, a new TCP socket option called TCP_MAD will be added to the TCP implementation. This will allow applications to request a MAD value on a finer granularity than the per-route configuration, depending on the application's requirements.

The API will look like the following example:

int mad_val = 5 * 1000 * 1000; // in ns unit: 5ms

err = setsockopt(fd, SOL_TCP, TCP_MAD, &mad_val, sizeof(mad_val));

The socket option implementation will sanitize the MAD value provided by the user. Per the specification above, in the "Configuring maximum ACK delay" section, if the user specifies a MAD value outside the range of ACK delay values supported by the implementation, then the implementation will allow the request to succeed, but will silently constrain the MAD value to be within the valid range (between the minimum and maximum ACK delay for the implementation). This is intended to allow applications to portably request a MAD value without needing special logic to search for a valid value.

Once the implementation has sanitized the provided MAD value, it will record the value in the socket as the socket's own MAD value.

Note: the MAD value set by the socket option SHOULD always override the per-route MAD value if there is one.

9. References

9.1. Normative References

   [draft-ietf-tcpm-rack]
              Cheng, Y., Cardwell, N., and N. Dukkipati, "RACK: a time-
              based fast loss detection algorithm for TCP", draft-ietf-
              tcpm-rack-02 (work in progress), March 2017.

   [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion
              Control", RFC 5681, September 2009.

   [RFC6298] Paxson, V., "Computing TCP's Retransmission Timer",
              RFC 6298, June 2011.

   [RFC6994] Touch, J., "Shared Use of Experimental TCP Options",
              RFC 6994, August 2013.

9.2. Informative References

   [CC16]   Cardwell, N., Cheng, Y., and E. Dumazet, "TCP Options for
              Low Latency: Maximum ACK Delay and Microsecond
              Timestamps", IETF 97 , November 2016.

   [CG09]   Chen, Y., Griffith, R., Liu, J., and R. Katz,
              "Understanding TCP incast throughput collapse in
              datacenter networks", WREN 09 , August 2009.

   [HN11]   Honda, M., Nishida, Y., Raiciu, C., Greenhalgh, A.,
              Handley, M., and H. Tokuda, "Is it Still Possible to
              Extend TCP?", IMC 11 , November 2011.

   [JS15]   Judd, G. and M. Stanley, "Attaining the Promise and
              Avoiding the Pitfalls of TCP in the Datacenter", NSDI 15 ,
              May 2015.

   [PK07]   Phanishayee, A., Krevat, E., Vasudevan, V., Andersen, D.,
              Ganger, G., Gibson, G., and S. Seshan, "Measurement and
              Analysis of TCP Throughput Collapse in Cluster-based
              Storage Systems", September 2007.

   [VP09]   Vasudevan, V., Phanishayee, A., Shah, H., Krevat, E.,
              Andersen, D., Ganger, G., Gibson, G., and B. Mueller,
              "Safe and Effective Fine-grained TCP Retransmissions for
              Datacenter Communication", SIGCOMM 09 , August 2009.

   [WS95]   Wright, G. and W. Stevens, "TCP/IP Illustrated, Volume 2:
              The Implementation", 1995.

Authors' Addresses

   Wei Wang
   Google, Inc
   1600 Amphitheater Parkway
   Mountain View, California 94043
   USA

   Email: weiwan@google.com

   Neal Cardwell
   Google, Inc
   76 Ninth Avenue
   New York, NY 10011
   USA

   Email: ncardwell@google.com

   Yuchung Cheng
   Google, Inc
   1600 Amphitheater Parkway
   Mountain View, California 94043
   USA

   Email: ycheng@google.com

   Eric Dumazet
   Google, Inc
   1600 Amphitheater Parkway
   Mountain View, California 94043

   Email: edumazet@google.com