Network Working Group
Internet-Draft
Intended status: Experimental
Expires: April 2, 2018

M. Bagnulo
UC3M
Y. Nishida
GE Global Research

September 29, 2017

TCP ESN: Extended Sequence Numbers for TCP
draft-bagnulo-tcpm-esn-00.txt

Abstract

This note defines the Extended Sequence Number (ESN) experimental modification to TCP to increase TCP’s sequence number using the TimeStamp (TS) option. It also modifies the Window Scale (WS) option to support larger receiver window enable by the extended sequence number space. At this stage, the purpose of this document is to discuss different design choices to generate discussion about the approach to follow.

Table of Contents

1. Overview

The proposed Extended Sequence Number (ESN) mechanism re-purposes the TS option [RFC7323] to carry a prefix for the sequence number and a prefix for the Acknowledgement number, increasing the sequence number used in TCP connections.

As currently defined, the TS option contains two 32-bit fields, TSval and TSecr. The current ESN proposal re-defines TSval to carry a prefix for the sequence number and TSecr to carry a prefix for the Acknowledgment number. In this way, the actual sequence number corresponding to the first data byte contained in the segment would the the concatenation of the value contained in the TSval and the value of the Sequence Number field of the TCP header. The Acknowledgment sequence number would be the concatenation of the value contained in the TSecr and the value of the Acknowledgment Number field of the TCP header.

The proposed ESN mechanism also modifies the WS option as follows: First, values up to 46 are allowed (enabling a RCV window up to 2^62). These are encoded in the 6 less significant bits of the shift.count. Second, the remaining two (most significant) bits are turned into flags. In particular, the most significant bit is used as the ESN flag to indicate the ESN support in the connection. Specifically, when the ESN bit is set to 1 in the WS carried in a SYN or a SYN/ACK, it means that: i) the TS option is being used for extended sequence numbers, as defined above, and ii) that the sender of the WS option with the ESN bit set supports receiver window up to 2^62 in this connection. The ESN flag defined this way allows endpoints to express and negotiate ESN support during the TCP 3-way handshake.

The sequence number of a TCP segment using ESN is the result of prepending the prefix carried in the TS Value and the sequence number contained in the Sequence Number field of the TCP header. Similarly, the ACK number is the result of prepending the value in the TS Echo Reply value and the value in the ACK field of the TCP header.

When a client wants to use the extended sequence number for a new connection, it sends a SYN with both the TS and the WS options. In the WS option, it sets the ESN flag to inform that it wants to use ESN for this connection. It encodes the most significant bits of the sequence number in the TS Value and the remaining bits of the extended sequence number in the sequence number field in the TCP header. Since the ACK flag is not set in the TCP header of the SYN packet, the TS Echo Value is set to zero (as defined in [RFC7323]).

If the server also supports the extended sequence number mechanism, the server replies with a SYN/ACK carrying both the TS and WS options. In the WS option it sets the ESN flag to confirm the ESN support. It encodes the prefix of its own extended sequence number in the TS Value and the prefix of the ACK in the TS Echo Reply.

If the server does not support ESN, it will respond with a SYN/ACK containing a WS option carrying a value lower then 14 i.e. with the most significant bit set to 0. It may also include the TS option indicating its willingness to use timestamps as defined in RFC7323 in this connection. Upon the reception of the SYN/ACK, the client can gracefully fall back to use TS are defined in RFC7323, in particular, PAWS can be used.

2. Design rationale

Our proposal is to re-utilize the TCP TS option to carry a sequence number offset in addition to the existing 32 bits sequence number. This approach is similar to [I-D.looney-tcpm-64-bit-seqnos] although it has distinct difference. while [I-D.looney-tcpm-64-bit-seqnos] proposes to allocate a new TCP option, we propose to utilize existing TS option instead. We believe this approach will have the following advantages.

2.1. Reduced option space consumption in the SYN and graceful fallback

The maximum size of the TCP header (including options) is 60 bytes (this is because the Data Offset field of the TCP header is 4 bits and can expresses the offset in 32-bit words). Since the TCP basic header is 20 bytes, a segment can carry 40 bytes of options at most. This is particularly pressing for the TCP SYN and TCP SYN/ACK packets. Currently, there is a fair number of options that are frequently carried in SYN packets, especially in high performance communications. In particular, the MSS option (2 bytes) [RFC0793], the SACK permitted option (2 bytes)[RFC2018], the Window Scale option (3 bytes) and the TimeStamp option (used for PAWS) (10 bytes) [RFC7323]. All these options account for 17 bytes. The are other options that are becoming increasingly popular. For instance, The option length of TCP Fast Open (TFO) [RFC7413] is 6 bytes or 18 bytes depending on the length of the cookie used. There are other options that require SYN and SYN/ACK option space such as MP_CAPABLE in [RFC6824], or TCP-AO [RFC5925].

This means that for instance, a TCP client that would like to initiate a connection including the MSS option, SACK permitted option the WS and TS options and also carry a TFO option would not have room to carry an additional 10 byte long option for the extended sequence number. Since our approach utilizes TS option, additional option space for extended sequence number is not needed.

The proposed ESN approach allows for using the extended sequence number if both endpoints support it while enabling graceful fall-back. A client supporting ESN would include the TS option and set the flag in the WS option indicating the ESN support. If the server does not support ESN, the connection can still be established using 32 bit sequence numbers and the TS and WS options as defined in RFC7323 (in particular PAWS can be used in the connection).

2.2. Deployability

[HONDA11] reported that unknown options in the SYN prone to be removed with higher probability than known options. Hence, we believe utilizing existing options will have better chances to avoid unwanted middleboxes’ interferences. Although it would be useful to perform some other measurements specifically about how frequently the TS option is removed.

3. RTTM With Extended Sequence Number Prefix

[RFC7323] defined two uses for the TS option: PAWS and RTTM. When re-purposing the TS option for ESN, we argue that the use of TS for carrying extended sequence number subsumes the uses of PAWS.

However, this is not the case for RTTM. We identify the following alternatives in order to archive RTTM when re-purposing the TS option for ESN.

Option 1:
This approach uses the most significant bit (MSB) of both TSval and TSecr as a flag as depicted in Figure 1. If the MSB is set to 1, it means the field contained a sequence number prefix. If it is reset, it means that it contains a timestamp. This means that we use 31 bits for the extended sequence number prefix, resulting in 63 bit long sequence numbers. The main problem here is that the segments containing the timestamp lack the sequence number prefix information. So, for instance, it is not possible to have more that 2^32 bytes in flight if any of the segments in flight is carrying and actual timestamp, since there is the possibility of confusion (in particular is the receive window is large enough to accommodate two packets with the same 32 bit sequence number, then the receiver would not be able to figure out the right place for the packet that carries the timestamp and does not carry the sequence number prefix). So, if we want to use this option, the receiver window cannot be larger than 2^32. However, this restriction does not address all the problems. If a duplicated packet carrying a timestamp in the TS option gets delay one RTT or more and the 32 bit sequence number wraps around, then the receiver can potentially take this old duplicated packet for a new packet with the same sequence number suffix. It would be possible to rely on PAWS for detecting and eliminating this packets. However, in order for PAWS to be used, it is necessary to keep the timestamp information stored in TS.recent updated. This requires that at least a few actual timestamps are exchanged every 2^31 sequence numbers. Summarizing, the constraints to use this option are first that the light-size is less than 2^32 and that at least n (n=4?) timestamps are exchanged every 2^32 bytes of data. We believe this is poor alternative, especially due to the flight-size constraint.

```
+-------+-------+-+---------------------+-+----------------------+
|Kind=8 |  10   |F|   TSval or Prefix   |F| TSecr or Prefix      |
+-------+-------+-+---------------------+-+----------------------+
    8       8    1          31           1           31
```

Figure 1: Time Stamp Option format for Option 1

Option 2:

This approach uses the TSecr in some packets to exchange timestamps. The idea here is that all data segments carry the extended sequence number prefix in the TSval but that some packets do not carry ACK information, which is acceptable because we use cumulative ACKs as long as this only affects a few packets (e.g. one packet per RTT do not carry ACK information). In order to enable both uses of the TSecr (timestamp or sequence number prefix), we need to use 2 bits to encode whether the TSecr carries either an extended sequence number prefix for the ACK, a timestamp or a timestamp echo. This implies that there are 30 bits left in TSecr for the actual value, resulting in 30 bit timestamps and 62 bit sequence numbers The receiver of a packet carrying the TS option carrying an actual timestamp or timestamp echo should discard the ACK information since it cannot know the the prefix of the seq number carried in the ACK field. This option seems a reasonable trade-off. If this option is adopted, RTTM could only be used sporadically. However, this may not be a concern, since it is likely that it would be possible to measure the RTT at least once every RTT which is likely to be enough for estimating the RTT for the RTO calculation (see [RFC7323] for further details).

```
+-------+-------+--+--------------------+--+---------------------+
|Kind=8 |  10   |F |   TSval or Prefix  |F |   TSecr or Prefix   |
+-------+-------+--+--------------------+--+---------------------+
    8       8    2          30            2          30
```

Figure 2: Time Stamp Option format for Option 2

Option 3:
This approach splits the TSval and the TSecr into two 16-bit fields resulting in 16 bit timestamps and 48 bit sequence numbers. 48 bit sequence numbers are a significant improvement from the current 32 bit sequence numbers, so it is probably enough. It is possible to encode the timestamp information using 16 bits. For example, [I-D.trammell-tcpm-timestamp-interval] proposes to encode timestamp information using 16 bits, which could be used in this option.

```
+-------+-------+-----------+-----------+------------+-----------+
|Kind=8 |  10   |  Prefix   |   TSval   |   Prefix   |   TSecr   |
+-------+-------+-----------+-----------+------------+-----------+
    8       8        16          16           16           16
```

Figure 3: Time Stamp Option format for Option 3

Option 4:
This approach Only uses the TS for one single purpose per connection either the original purpose or ESN. This will be less attractive because the RTTM cannot be used with ESN in the same connection.

```
+-------+-------+-----------------------+------------------------+
|Kind=8 |  10   |        Prefix         |         Prefix         |
+-------+-------+-----------------------+------------------------+
    8       8              32                         32
```

Figure 4: Time Stamp Option format for Option 4

Based on the observations above, we believe option 2 and 3 would be worth for further discussions while option 1 and 4 can be discarded due to major drawbacks.

4. Middleboxes Implications

It has been observed in [HONDA11] that some middleboxes insert the TS Option. Also, there may be boxes out there that modify the sequence number, while not terminating the connection. In order to detect these cases that would break the proposed mechanism, it would be beneficial to add an extra safety measure requiring that the prefix encoded in the TS Option replicates the most significant bits of the value included in the Sequence number field. In this way, a server supporting the extended sequence number mechanism cannot only verify the flag in the WS option, but also check if the TS value matches with the 31 most significant bits in the Sequence Number field in the TCP header. If they do not match, the server should not negotiate the use of the extended sequence number mechanism (i.e. it replies with the WS option resetting the flag for the extended sequence number mechanism). This is adopted from [I-D.looney-tcpm-64-bit-seqnos].

In case that the server is a legacy server, it will reply without the WS option or with the WS option with a shift.count value lower than

15. In this case, the client falls back to regular TCP without the extended sequence number and regular timestamps.

5. SACK for Extended Sequence Number

In the case of SACK blocks, there are two possible complementary approaches:

1. we use the currently defined SACK options identifying bits using 32 bit sequence numbers. These are used in a connection that has successfully negotiated ESN, the prefix carried in the TSecr of the message applies also to the sequence numbers identifying the SACK blocks. The limitation of such approach is that all SACK blocks in a single SACK option must use to the same prefix, which prevents from SACKing older blocks. However, it is not certain that if we really need to report wide range of SACK blocks in a single SACK option. Another issue would be the case where a SACK option is detached from the original packet and attached to a different one. One possible mitigation for this would be discarding SACK info in case of suspicious as SACK is optional info and a SACK info usually is carried in multiple ACKs.

2. define a new SACK block option for extended sequence numbers as proposed in [I-D.looney-tcpm-64-bit-seqnos].

There are a couple of observations regarding the last option using the new SACK block option. First, note that the currently SACK permitted option could still be used. Hence, if a connection negotiated both SACK and ESN, we may presume that it supports the new SACK block option. If the ESN negotiation fails, it means that 32-bit SACK are to be used for that connection, providing graceful fallback.

6. Impacts On Other TCP Extensions

Since this proposal repurpose the existing use of timestamp option, some other proposals that use the option will be affected. We investigated the impacts on the following TCP extensions and propose modifications to make them work with the proposal.

6.1. PAWS

In order to perform PAWS, receives need to check if the timestamp option in an arrived packet contains sequence number prefix or timestamp info by checking the most significant bit. If it contains timestamp info, it process the timestamp info as described Section 5.3 in [RFC7323]. If it contains sequence number prefix, it can know the extended sequence number of the packet based on the into. If the extended sequence number is outside of the window, the packet will be discarded as PAWS.

6.2. Eifel Detection Algorithm

If Eifel detection algorithm [RFC3522] is activated, senders performs the logics described in Section 3.2 of [RFC3522] with the following two modifications. First, TCP sender MUST set timestamp info when it retransmit packets. Second, if TCP sender receives the ACK with sequence number prefix for the retransmitted packet, it should treat as if the timestamp is smaller than the value of RetransmitTS.

7. Acknowledgments

8. Security Considerations

9. IANA Considerations

10. References

10.1. Normative References

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

[RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, DOI 10.17487/RFC7323, September 2014, <https://www.rfc-editor.org/info/rfc7323>.

10.2. Informative References

[HONDA11] Honda, M., Nishida, Y., Raiciu, C., Greenhalgh, A., Handley, M., and H. Tokuda, "Is it still possible to extend TCP?", ACM IMC 2011, 2011.

[I-D.looney-tcpm-64-bit-seqnos] jlooney@juniper.net, j., "64-bit Sequence Numbers for TCP", draft-looney-tcpm-64-bit-seqnos-00 (work in progress), March 2017.

[I-D.trammell-tcpm-timestamp-interval] Scheffenegger, R., Kuehlewind, M., and B. Trammell, "Encoding of Time Intervals for the TCP Timestamp Option", draft-trammell-tcpm-timestamp-interval-01 (work in progress), July 2013.

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

[RFC3522] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC 3522, DOI 10.17487/RFC3522, April 2003, <https://www.rfc-editor.org/info/rfc3522>.

[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>.

Authors' Addresses

Marcelo Bagnulo
UC3M

Email: marcelo@it.uc3m.es

Yoshifumi Nishida
GE Global Research
2623 Camino Ramon
San Ramon, CA 94583
USA

Email: nishida@wide.ad.jp

TCP Maintenance & Minor Extensions (tcpm)
Internet-Draft
Intended status: Experimental
Expires: May 3, 2018

B. Briscoe
CableLabs
M. Kuehlewind
ETH Zurich
R. Scheffenegger

October 30, 2017

More Accurate ECN Feedback in TCP
draft-ietf-tcpm-accurate-ecn-04

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, new 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.

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 whenever more than one marking is received in one RTT. 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 ECN feedback, not a fork in the design of TCP. Thus, the applicability of AccECN is intended to include all public and private IP networks (and even any non-IP networks over which TCP is used today). Until the AccECN experiment succeeds, [RFC3168] will remain as the 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 flags and fields 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 [I-D.ietf-tsvwg-ecn-experimentation], 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.

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 RFC 2119 [RFC2119].

1.5. Recap of Existing ECN feedback in IP/TCP

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

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

Table 1: The ECN Field in the IP Header

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

The last bit in byte 13 of the TCP header was defined as the Nonce Sum (NS) for the ECN Nonce [RFC3540]. RFC 3540 was never deployed so it is being reclassified as historic, 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:

- 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;

- 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:

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

- a single upgrade path for the TCP protocol is preferable to a fork in the design;

- 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;

- 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 [I-D.ietf-tsvwg-ecn-experimentation]. 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, [I-D.ietf-tsvwg-ecn-experimentation] 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] is being reclassified as historic, 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: An AccECN implementation, whether client or server, sender or receiver, does not need to implement the ECN Nonce feedback mode [RFC3540], which is being reclassified as historic [I-D.ietf-tsvwg-ecn-experimentation]. 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.

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.

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

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

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

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.

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 also 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 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. AccECN Compliance by TCP Proxies, Offload Engines and other Middleboxes

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 MUST attempt to preserve the timing of each ACK (for example, if it coalesced ACKs it would not be AccECN-compliant). 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 MUST attempt to 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:

- the TCP client must have requested AccECN support on the SYN
- 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:

- 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 is being 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 [I-D.ietf-tsvwg-ecn-experimentation]. 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 and Michael Welzl 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). The views expressed here are solely those of the authors.

This work is 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>.

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-02 (work in progress), October 2017.

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

[I-D.ietf-tsvwg-ecn-experimentation]
Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", draft-ietf-tsvwg-ecn-experimentation-07 (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-00 (work in progress), May 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>.

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
Internet-Draft
Intended status: Experimental
Expires: April 23, 2018

N. Khademi
M. Welzl
University of Oslo
G. Armitage
Swinburne University of Technology
G. Fairhurst
University of Aberdeen

October 20, 2017

TCP Alternative Backoff with ECN (ABE)
draft-ietf-tcpm-alternativebackoff-ecn-02

Abstract

Recent Active Queue Management (AQM) mechanisms instantiate shallow buffers with burst tolerance to minimise the time that packets spend enqueued at a bottleneck. However, shallow buffering can cause noticeable performance degradation when TCP is used over a network path with a large bandwidth-delay-product. Traditional methods rely on detecting network congestion through reported loss of transport packets. Explicit Congestion Notification (ECN) instead allows a router to directly signal incipient congestion. A sending endpoint can distinguish when congestion is signalled via ECN, rather than by packet loss. An ECN signal indicates that an AQM mechanism has done its job, and therefore the bottleneck network queue is likely to be shallow. This document therefore proposes an update to 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 loss. This document also recommends this approach to be adopted by any other transport protocol that implements a congestion control reduction to an ECN congestion signal.

Status of This Memo

This Internet-Draft will expire on April 23, 2018.

Copyright Notice

Table of Contents

1. 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].

2. 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. There are also significant other benefits from deploying ECN [RFC8087], including reduced end-to-end network latency.

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 a packet loss [RFC3168].

Research has demonstrated the benefits of reducing network delays due to excessive buffering [BUFFERBLOAT]; this has led to the creation of new AQM mechanisms like PIE [RFC8033] and CoDel [CODEL2012] [I-D.CoDel], which avoid causing bloated queues that are common with a simple tail-drop behaviour (also known as a First-In First-Out, FIFO, queue).

These AQM mechanisms instantiate short queues that are designed to tolerate packet bursts. However, congestion control mechanisms cannot always utilise a bottleneck link well where there are short queues. For example, to allow a single TCP connection to fully utilise a network path, the queue at the bottleneck link must be able to compensate for TCP halving the "cwnd" and "ssthresh" variables in response to a lost packet [RFC5681]. This requires the bottleneck queue to be able to store at least an end-to-end bandwidth-delay product (BDP) of data, which effectively doubles both the amount of data that can be in flight and the round-trip time (RTT) experience 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 packet loss [I-D.ECN-exp]. 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 the performance when routers use shallow buffered AQM mechanisms.

3. Specification

This specification describes an update to the congestion control algorithm of an ECN-capable TCP transport protocol. It allows a TCP stack to update the TCP sender response when it receives feedback indicating reception of a CE-marked packet. It RECOMMENDS that a TCP sender multiplies the cwnd by 0.8 and reduces the slow start threshold (ssthresh) in congestion avoidance following reception of a TCP segment that sets the ECN-Echo flag (defined in [RFC3168]). While this specification concerns TCP, other transports also support a per-RTT response to ECN. The method defined in this document is also applicable for such transports.

4. Discussion

Much of the technical background to this congestion control response can be found in a research paper [ABE2017]. This paper used a mix of experiments, theory and simulations with standard NewReno and CUBIC to evaluate the technique. It examined the impact of enabling ECN and letting individual TCP senders back off by a reduced amount in reaction to the receiver that reports ECN CE-marks from AQM-enabled bottlenecks. The technique was shown to present "...significant performance gains in lightly-multiplexed 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 by multiplying cwnd and ssthresh with a value in the range [0.7..0.85].

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

The classic rule-of-thumb dictates that a network path needs to provide a BDP of bottleneck buffering if a TCP connection wishes to optimise path utilisation. A single TCP bulk transfer running through such a bottleneck will have increased its congestion window (cwnd) up to 2*BDP by the time that packet loss occurs. When packet loss is detected (regarded as a notification of congestion), Standard TCP halves the cwnd and ssthresh [RFC5681], which causes the TCP congestion control to go back to allowing only a BDP of packets in flight -- just sufficient to maintain 100% utilisation of the bottleneck on the network path.

AQM mechanisms such as CoDel [I-D.CoDel] 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 shallow buffered bottleneck (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 connections). However, it interacts badly for a lightly-multiplexed case (few concurrent connections) over a path with a large BDP. Conventional TCP backoff in such cases 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 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 shallow queue. Reacting differently to an ECN CE-mark than to packet loss can then yield the benefit of a reduced back-off, as with CUBIC [I-D.CUBIC], when queues are short, yet it can avoid generating excessive delay when queues are long. Using ECN can also be advantageous for several other reasons [RFC8087].

The idea of reacting differently to loss and detection of an ECN CE-mark pre-dates this document. For example, previous research proposed using ECN CE-marks 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 shallow queue ([RFC7567] notes the current status of RED as an AQM method.)

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 example, Congestion Exposure (ConEx) [RFC7713] and Datacenter TCP (DCTCP) [I-D.ietf-tcpm-dctcp] need more accurate ECN information than that offered by the original feedback method. Other mechanisms (e.g., [I-D.ietf-tcpm-accurate-ecn]) allow the sender to adjust the rate more frequently than once each path RTT. Use of these mechanisms is out of the scope of the current document.

4.3. Discussion: Choice of ABE Multiplier

ABE decouples the reaction of a TCP sender to loss and ECN CE-marks when in the congestion avoidance phase by differentiating the scaling factor used 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 packet loss, and in response to a receiver indicating that an ECN CE-mark was received on an ECN-enabled TCP connection. For non-ECN-enabled

TCP connections, no ECN CE-marks are received and only beta_{loss} applies.

In other words, in response to detected loss:

ssthresh_(t+1) = max (FlightSize_t * beta_{loss}, 2 * SMSS)

and in response to an indication of a received ECN CE-mark:

ssthresh_(t+1) = max (FlightSize_t * beta_{ecn}, 2 * SMSS)

and

cwnd_(t+1) = ssthresh_(t+1)

where FlightSize is the amount of outstanding data in the network, upper-bounded by 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 [I-D.CUBIC] 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.

beta_{ecn} depends on how the response of a TCP connection to shallow AQM marking thresholds is optimised. beta_{loss} reflects the preferred response of each congestion control algorithm when faced with exhaustion of buffers (of unknown depth) signalled by packet loss. Consequently, for any given TCP congestion control algorithm the choice of beta_{ecn} is likely to be algorithm-specific, 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. Status of the Update

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.

The currently published ECN specification requires that the congestion control response to a CE-marked packet is the same as the response to a dropped packet [RFC3168]. The specification is currently being updated to allow for specifications that do not follow this rule [I-D.ECN-exp]. 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 (except to enable an ECN-marking mechanism [RFC3168] [RFC7567]) 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 loss indications, the new method cannot lead to congestion collapse.

The result of this Internet experiment will be reported by presentation to the TCPM WG (or IESG) or an implementation report at the end of the experiment.

6. Acknowledgements

Authors N. Khademi, M. Welzl and G. Fairhurst were part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The views expressed are solely those of the authors.

The authors would like to thank Stuart Cheshire for many suggestions when revising the draft, and the following people for their contributions to [ABE2017]: Chamil Kulatunga, David Ros, Stein

Gjessing, Sebastian Zander. Thanks also to (in alphabetical order) Bob Briscoe, Markku Kojo, John Leslie, Dave Taht and the TCPM working group for providing valuable feedback on this document.

The authors would finally like to thank everyone who provided feedback on the congestion control behaviour specified in this update received from the IRTF Internet Congestion Control Research Group (ICCRG).

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]. An evolved version of the patch for FreeBSD is currently under review for potential inclusion in the mainline kernel [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 [I-D.CUBIC]). 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

-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 implements 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

[I-D.ECN-exp]
Black, D., "Explicit Congestion Notification (ECN) Experimentation", Internet-draft, IETF work-in-progress draft-ietf-tsvwg-ecn-experimentation-06, September 2017.

[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>.

11.2. Informative References

[ABE-FreeBSD] "ABE patch review in FreeBSD", <https://reviews.freebsd.org/D11616>.

[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] "Bufferbloat project", <https://www.bufferbloat.net/projects/bloat/wiki/Introduction/>.

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

[I-D.CoDel] Nichols, K., Jacobson, V., McGregor, V., and J. Iyengar, "Controlled Delay Active Queue Management", Internet-draft, IETF work-in-progress draft-ietf-aqm-codel-09, September 2017.

[I-D.CUBIC] Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", Internet-draft, IETF work-in-progress draft-ietf-tcpm-cubic-06, September 2017.

[I-D.ietf-tcpm-accurate-ecn] Briscoe, B., Kuehlewind, M., and R. Scheffenegger, "More Accurate ECN Feedback in TCP", draft-ietf-tcpm-accurate-ecn-03 (work in progress), May 2017.

[I-D.ietf-tcpm-dctcp]
Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters", draft-ietf-tcpm-dctcp-10 (work in progress), August 2017.

[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>.

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

Network Working Group
Internet-Draft
Obsoletes: 5562 (if approved)
Intended status: Experimental
Expires: May 3, 2018
October 30, 2017

M. Bagnulo
UC3M
B. Briscoe
CableLabs

ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control Packets

draft-ietf-tcpm-generalized-ecn-02

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

Copyright Notice

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

This document defines an experimental modification to ECN [RFC3168] that shall be called ECN++. It enables ECN support on all the aforementioned types of TCP packet.

ECN++ is a sender-side change. It works whether the two ends of the TCP connection use classic ECN feedback [RFC3168] or experimental Accurate ECN feedback (AccECN [I-D.ietf-tcpm-accurate-ecn]). Nonetheless, if the client does not implement AccECN, it cannot use ECN++ on the one packet that offers most benefit from it - the initial SYN. Therefore, implementers of ECN++ are RECOMMENDED to also implement AccECN.

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

1.1. Motivation

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

Non-ECN control packets particularly harm performance in environments where the ECN marking level is high. For example, [judd-nsdi] shows that in a controlled private data centre (DC) environment where ECN is used (in conjunction with DCTCP [RFC8257]), the probability of being able to establish a new connection using a non-ECN SYN packet drops to close to zero even when there are only 16 ongoing TCP flows transmitting at full speed. The issue is that DCTCP exhibits a much more aggressive response to packet marking (which is why it is only applicable in controlled environments). This leads to a high marking probability for ECN-capable packets, and in turn a high drop probability for non-ECN packets. Therefore non-ECN SYNs are dropped aggressively, rendering it nearly impossible to establish a new connection in the presence of even mild traffic load.

Finally, there are ongoing experimental efforts to promote the adoption of a slightly modified variant of DCTCP (and similar congestion controls) over the Internet to achieve low latency, low loss and scalable throughput (L4S) for all communications [I-D.ietf-tsvwg-l4s-arch]. In such an approach, L4S packets identify themselves using an ECN codepoint [I-D.ietf-tsvwg-ecn-l4s-id]. With L4S 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 packets 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.ietf-tsvwg-ecn-l4s-id].

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

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

3. Specification

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 and independent of any network behaviour.

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

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	ECT	ECT	Reduce IW
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.

3.2.1.2. Caching Lack of AccECN 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 entry per server to record that the server does not support AccECN and therefore has no logic for congestion markings on the SYN. Mobile hosts MAY maintain a cache entry per access network to record lack of AccECN support by proxies (see Section 4.2.1).

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 ECT has been set on the SYN and 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. Some evidence of this was found in a 2014 study [ecn-pam], but in a more recent study using 2017 data [Mandalari18] extensive measurements found no case where ECT on TCP control packets was treated any differently from ECT on TCP data packets. Loss is commonplace for numerous other reasons, e.g. congestion loss at a non-ECN queue on the forward or reverse path, transmission errors, etc. Alternatively, the cause of the loss might be the attempt to negotiate AccECN, or possibly other unrelated options on the SYN.

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

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

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

3.2.2. SYN-ACK

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

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

The server MAY cache failed connection attempts, e.g. per client access network. An client-based alternative to caching at the server is given in Section 4.3.2. If the TCP server is caching failed connection attempts, it SHOULD NOT give up using ECT on the first

SYN-ACK of subsequent connection attempts until it is clear that the blockage persistently and specifically affects ECT on SYN-ACKs. This is because loss is so commonplace for other reasons (see Section 3.2.1.4). If a cache is used, it SHOULD be arranged to expire so that the server will infrequently attempt to check whether the problem has been resolved.

3.2.3. Pure ACK

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. {ToDo: Write-up reconsideration of this requirement in the light of WG comments.} 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 RFC 3168 servers react to pure ACKs marked with the ECT(0)/ECT(1)/CE codepoints, i.e. whether they are dropped, codepoint cleared or processed and the congestion indication fed back on a subsequent packet.

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

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

3.2.5. FIN

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.

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

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

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

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

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.

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

(S3): ECT by configuration: In a controlled environment, the administrator can make sure that servers support ECN-capable SYN packets. Examples of controlled environments are single-tenant DCs, and possibly multi-tenant DCs if it is assumed that each tenant mostly communicates with its own VMs.

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

- 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.

- 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%.

In a study using 2017 data from a wider range of fixed and mobile vantage points to the top 500k Alexa servers, no case was found where adding the ECN capability to a SYN increased the likelihood of connection establishment failure [Mandalari18].

MEASUREMENTS NEEDED: More investigation is needed to understand the different outcomes of the 2014 and 2017 studies.

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, in the 2014 study, perhaps some responder implementations were checking that the SYN complied with RFC 3168, then silently ignoring non-compliant SYNs (or perhaps returning a RST). Also some middleboxes (e.g. firewalls) might have been 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, if there is any discard of ECN-capable SYNs 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 third connection attempt after the more careful ("pessimistic") first and second attempts.

If so, it might be necessary to send a not-ECT SYN a short delay after an ECT SYN and only accept the non-ECT connection if it returned first. This would reduce the performance penalty for those deploying ECT SYN support.

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

4.3.1. Response to Congestion on a SYN-ACK

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

The present ECN experiment obsoletes RFC 5562 because it uses the ECN+ congestion response, not ECN+/TryOnce. First we argue against the rationale for ECN+/TryOnce given in sections 4.4 and 6.2 of [RFC5562]. It starts with a rather too literal interpretation of the requirement in RFC 3168 that says TCP's response to a single CE mark has to be "essentially the same as the congestion control response to a *single* dropped packet." TCP's response to a dropped initial (SYN or SYN-ACK) packet is to wait for the retransmission timer to expire (currently 1s). However, this long delay assumes the worst case between two possible causes of the loss: a) heavy overload; or b) the normal capacity-seeking behaviour of other TCP flows. When the network is still delivering CE-marked packets, it implies that there is an AQM at the bottleneck and that it is not overloaded. This is because an AQM under overload will disable ECN (as recommended in section 7 of RFC 3168 and repeated in section 4.2.1 of RFC 7567). So scenario (a) can be ruled out. Therefore, TCP's response to a CE-marked SYN-ACK can be similar to its response to the loss of _any_ packet, rather than backing off as if the special _initial_ packet of a flow has been lost.

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

Therefore, we have to work out a reasoned argument for what to do. If an AQM is CE-marking packets, it implies there is already a queue and it is probably already somewhere around the AQM's operating point - it is unlikely to be well below and it might be well above. So, 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.

4.3.2. Fall-Back if ECT SYN-ACK Fails

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

4.4. Pure ACKs

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

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

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

Later on, in section 6.1.4 it reads:

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

We next address each of the arguments presented above.

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

The second argument says that ECN ought not to be enabled unless there is a mechanism to respond to it. 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. Note that, while a host has no application data to send, any congestion window it has attained might also be reduced by the congestion window validation mechanism [RFC7661].

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.

4.9. General Fall-back for any Control Packet

Extensive experiments have found no evidence of any traversal problems with ECT on any TCP control packet [Mandalari18]. Nonetheless, Sections 3.2.1.4 and 3.2.2.3 specify fall-back measures if ECT on the first packet of each half-connection (SYN or SYN-ACK) appears to be blocking progress. Here, the question of fall-back measures for ECT on other control packets is explored. It supports the advice given in Section 3.2.8; until there's evidence that something's broken, don't fix it.

If an implementation has had to disable ECT to ensure the first packet of a flow (SYN or SYN-ACK) gets through, the question arises whether it ought to disable ECT on all subsequent control packets within the same TCP connection. Without evidence of any such problems, this seems unnecessarily cautious. Particularly given it would be hard to detect loss of most other types of TCP control packets that are not ACK'd. And particularly given that unnecessarily removing ECT from other control packets could lead to performance problems, e.g. by directing them into an inferior queue [I-D.ietf-tsvwg-ecn-l4s-id] or over a different path, because some broken multipath equipment (erroneously) routes based on all 8 bits of the Diffserv field.

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

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

5. Interaction with popular variants or derivatives of TCP

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

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

5.1. IW10

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

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

As specified in Section 3.2.1.1, a TCP initiator 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.

5.2. TFO

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

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

- o The handling of ECN marking on a SYN is no different whether or not it carries data.

- o In response to any CE-marking on the SYN-ACK, the responder adopts the normal response to congestion, as discussed in Section 7.2 of [RFC7413].

5.3. TCP Derivatives

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

QUIC [I-D.ietf-quic-transport] is another standards track transport protocol offering similar services to TCP but intended to exploit some of the benefits of running over UDP. A way to add ECN support to QUIC has been proposed [I-D.johansson-quic-ecn].

Experience from experiments on adding ECN support to all TCP packets ought to be directly transferable to derivatives of TCP, like SCTP or QUIC.

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, David Black, Padma Bhooma and Gorry Fairhurst 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 consortium's view, but the consortium is not liable for any use that may be made of any of the information contained therein.

Bob Briscoe's contribution was partly funded by the Research Council of Norway through the TimeIn project. The views expressed here are solely those of the authors.

9. References

9.1. Normative References

[I-D.ietf-tcpm-accurate-ecn]
Briscoe, B., Kuehlewind, M., and R. Scheffenegger, "More Accurate ECN Feedback in TCP", draft-ietf-tcpm-accurate-ecn-03 (work in progress), May 2017.

[I-D.ietf-tsvwg-ecn-experimentation]
Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", draft-ietf-tsvwg-ecn-experimentation-07 (work in progress), October 2017.

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

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <https://www.rfc-editor.org/info/rfc3168>.

[RFC5961] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", RFC 5961, DOI 10.17487/RFC5961, August 2010, <https://www.rfc-editor.org/info/rfc5961>.

9.2. Informative References

[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.

[I-D.ietf-quic-transport]
Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-07 (work in progress), October 2017.

[I-D.ietf-tsvwg-ecn-l4s-id]
Schepper, K. and B. Briscoe, "Identifying Modified Explicit Congestion Notification (ECN) Semantics for Ultra-Low Queuing Delay", draft-ietf-tsvwg-ecn-l4s-id-00 (work in progress), May 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-00 (work in progress), May 2017.

[I-D.johansson-quic-ecn]
Johansson, I., "ECN support in QUIC", draft-johansson-quic-ecn-03 (work in progress), May 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.

[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)

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://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, <https://www.rfc-editor.org/info/rfc1122>.

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

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/info/rfc4960>.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <https://www.rfc-editor.org/info/rfc4987>.

[RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", RFC 5562, DOI 10.17487/RFC5562, June 2009, <https://www.rfc-editor.org/info/rfc5562>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/info/rfc5681>.

[RFC5690] Floyd, S., Arcia, A., Ros, D., and J. Iyengar, "Adding Acknowledgement Congestion Control to TCP", RFC 5690, DOI 10.17487/RFC5690, February 2010, <https://www.rfc-editor.org/info/rfc5690>.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <https://www.rfc-editor.org/info/rfc6298>.

[RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <https://www.rfc-editor.org/info/rfc6928>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.

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

[RFC7661] Fairhurst, G., Sathiaseelan, A., and R. Secchi, "Updating TCP to Support Rate-Limited Traffic", RFC 7661, DOI 10.17487/RFC7661, October 2015, <https://www.rfc-editor.org/info/rfc7661>.

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

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
CableLabs
UK

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

TCP Maintenance Working Group Y. Cheng
Internet-Draft N. Cardwell
Intended status: Experimental N. Dukkipati
Expires: May 4, 2017 Google, Inc
October 31, 2016

RACK: a time-based fast loss detection algorithm for TCP
draft-ietf-tcpm-rack-01

Abstract

This document presents a new TCP loss detection algorithm called RACK ("Recent ACKnowledgment"). RACK uses the notion of time, instead of packet or sequence counts, to detect losses, for modern TCP implementations that can support per-packet timestamps and the selective acknowledgment (SACK) option. It is intended to replace the conventional DUPACK threshold approach and its variants, as well as other nonstandard approaches.

Status of This Memo

Copyright Notice

1. Introduction

This document presents a new loss detection algorithm called RACK ("Recent ACKnowledgment"). RACK uses the notion of time instead of the conventional packet or sequence counting approaches for detecting losses. RACK deems a packet lost if some packet sent sufficiently later has been delivered. It does this by recording packet transmission times and inferring losses using cumulative acknowledgments or selective acknowledgment (SACK) TCP options.

In the last couple of years we have been observing several increasingly common loss and reordering patterns in the Internet:

1. Lost retransmissions. Traffic policers [POLICER16] and burst losses often cause retransmissions to be lost again, severely increasing TCP latency.

2. Tail drops. Structured request-response traffic turns more losses into tail drops. In such cases, TCP is application-limited, so it cannot send new data to probe losses and has to rely on retransmission timeouts (RTOs).

3. Reordering. Link layer protocols (e.g., 802.11 block ACK) or routers' internal load-balancing can deliver TCP packets out of order. The degree of such reordering is usually within the order of the path round trip time.

Despite TCP stacks (e.g. Linux) that implement many of the standard and proposed loss detection algorithms [RFC3517][RFC4653][RFC5827][RFC5681][RFC6675][RFC7765][FACK][THIN-STREAM][TLP], we've found that together they do not perform well. The main reason is that many of them are based on the classic rule of counting duplicate acknowledgments [RFC5681]. They can either detect loss quickly or accurately, but not both, especially when the sender is application-limited or under reordering that is unpredictable. And under these conditions none of them can detect lost retransmissions well.

Also, these algorithms, including RFCs, rarely address the interactions with other algorithms. For example, FACK may consider a packet is lost while RFC3517 may not. Implementing N algorithms while dealing with N^2 interactions is a daunting task and error-prone.

The goal of RACK is to solve all the problems above by replacing many of the loss detection algorithms above with one simpler, and also more effective, algorithm.

2. Overview

The main idea behind RACK is that if a packet has been delivered out of order, then the packets sent chronologically before that were either lost or reordered. This concept is not fundamentally different from [RFC5681][RFC3517][FACK]. But the key innovation in RACK is to use a per-packet transmission timestamp and widely deployed SACK options to conduct time-based inferences instead of inferring losses with packet or sequence counting approaches.

Using a threshold for counting duplicate acknowledgments (i.e., dupthresh) is no longer reliable because of today's prevalent reordering patterns. A common type of reordering is that the last "runt" packet of a window's worth of packet bursts gets delivered first, then the rest arrive shortly after in order. To handle this effectively, a sender would need to constantly adjust the dupthresh to the burst size; but this would risk increasing the frequency of RTOs on real losses.

Today's prevalent lost retransmissions also cause problems with packet-counting approaches [RFC5681][RFC3517][FACK], since those approaches depend on reasoning in sequence number space. Retransmissions break the direct correspondence between ordering in sequence space and ordering in time. So when retransmissions are lost, sequence-based approaches are often unable to infer and quickly repair losses that can be deduced with time-based approaches.

Instead of counting packets, RACK uses the most recently delivered packet's transmission time to judge if some packets sent previous to that time have "expired" by passing a certain reordering settling window. On each ACK, RACK marks any already-expired packets lost, and for any packets that have not yet expired it waits until the reordering window passes and then marks those lost as well. In either case, RACK can repair the loss without waiting for a (long) RTO. RACK can be applied to both fast recovery and timeout recovery, and can detect losses on both originally transmitted and retransmitted packets, making it a great all-weather recovery mechanism.

3. Requirements

The reader is expected to be familiar with the definitions given in the TCP congestion control [RFC5681] and selective acknowledgment

[RFC2018] RFCs. Familiarity with the conservative SACK-based recovery for TCP [RFC6675] is not expected but helps.

RACK has three requirements:

1. The connection MUST use selective acknowledgment (SACK) options [RFC2018].

2. For each packet sent, the sender MUST store its most recent transmission time with (at least) millisecond granularity. For round-trip times lower than a millisecond (e.g., intra-datacenter communications) microsecond granularity would significantly help the detection latency but is not required.

3. For each packet sent, the sender MUST remember whether the packet has been retransmitted or not.

We assume that requirement 1 implies the sender keeps a SACK scoreboard, which is a data structure to store selective acknowledgment information on a per-connection basis. For the ease of explaining the algorithm, we use a pseudo-scoreboard that manages the data in sequence number ranges. But the specifics of the data structure are left to the implementor.

RACK does not need any change on the receiver.

4. Definitions of variables

A sender needs to store these new RACK variables:

"Packet.xmit_ts" is the time of the last transmission of a data packet, including retransmissions, if any. The sender needs to record the transmission time for each packet sent and not yet acknowledged. The time MUST be stored at millisecond granularity or finer.

"RACK.packet". Among all the packets that have been either selectively or cummulatively acknowledged, RACK.packet is the one that was sent most recently (including retransmission).

"RACK.xmit_ts" is the latest transmission timestamp of RACK.packet.

"RACK.end_seq" is the ending TCP sequence number of RACk.packet.

"RACK.RTT" is the associated RTT measured when RACK.xmit_ts, above, was changed. It is the RTT of the most recently transmitted packet that has been delivered (either cumulatively acknowledged or selectively acknowledged) on the connection.

"RACK.reo_wnd" is a reordering window for the connection, computed in the unit of time used for recording packet transmission times. It is used to defer the moment at which RACK marks a packet lost.

"RACK.min_RTT" is the estimated minimum round-trip time (RTT) of the connection.

"RACK.ack_ts" is the time when all the sequences in RACK.packet were selectively or cumulatively acknowledged.

Note that the Packet.xmit_ts variable is per packet in flight. The RACK.xmit_ts, RACK.RTT, RACK.reo_wnd, and RACK.min_RTT variables are to keep in TCP control block per connection. RACK.packet and RACK.ack_ts are used as local variables in the algorithm.

5. Algorithm Details

5.1. Transmitting a data packet

Upon transmitting a new packet or retransmitting an old packet, record the time in Packet.xmit_ts. RACK does not care if the retransmission is triggered by an ACK, new application data, an RTO, or any other means.

5.2. Upon receiving an ACK

Step 1: Update RACK.min_RTT.

Use the RTT measurements obtained in [RFC6298] or [RFC7323] to update the estimated minimum RTT in RACK.min_RTT. The sender can track a simple global minimum of all RTT measurements from the connection, or a windowed min-filtered value of recent RTT measurements. This document does not specify an exact approach.

Step 2: Update RACK.reo_wnd.

To handle the prevalent small degree of reordering, RACK.reo_wnd serves as an allowance for settling time before marking a packet lost. By default it is 1 millisecond. We RECOMMEND implementing the reordering detection in [REORDER-DETECT][RFC4737] to dynamically adjust the reordering window. When the sender detects packet reordering RACK.reo_wnd MAY be changed to RACK.min_RTT/4. We discuss more about the reordering window in the next section.

Step 3: Advance RACK.xmit_ts and update RACK.RTT and RACK.end_seq

Given the information provided in an ACK, each packet cumulatively ACKed or SACKed is marked as delivered in the scoreboard. Among all the packets newly ACKed or SACKed in the connection, record the most recent Packet.xmit_ts in RACK.xmit_ts if it is ahead of RACK.xmit_ts. Ignore the packet if any of its TCP sequences has been retransmitted before and either of two condition is true:

1. The Timestamp Echo Reply field (TSecr) of the ACK's timestamp option [RFC7323], if available, indicates the ACK was not acknowledging the last retransmission of the packet.

2. The packet was last retransmitted less than RACK.min_rtt ago. While it is still possible the packet is spuriously retransmitted because of a recent RTT decrease, we believe that our experience suggests this is a reasonable heuristic.

If this ACK causes a change to RACK.xmit_ts then record the RTT and sequence implied by this ACK:

```
RACK.RTT = Now() - RACK.xmit_ts
RACK.end_seq = Packet.end_seq
```

Exit here and omit the following steps if RACK.xmit_ts has not changed.

Step 4: Detect losses.

For each packet that has not been fully SACKed, if RACK.xmit_ts is after Packet.xmit_ts + RACK.reo_wnd, then mark the packet (or its corresponding sequence range) lost in the scoreboard. The rationale is that if another packet that was sent later has been delivered, and the reordering window or "reordering settling time" has already passed, the packet was likely lost.

If a packet that was sent later has been delivered, but the reordering window has not passed, then it is not yet safe to deem the given packet lost. Using the basic algorithm above, the sender would wait for the next ACK to further advance RACK.xmit_ts; but this risks a timeout (RTO) if no more ACKs come back (e.g, due to losses or application limit). For timely loss detection, the sender MAY install a "reordering settling" timer set to fire at the earliest moment at which it is safe to conclude that some packet is lost. The earliest moment is the time it takes to expire the reordering window of the earliest unacked packet in flight.

This timer expiration value can be derived as follows. As a starting point, we consider that the reordering window has passed if the RACK.packet was sent sufficiently after the packet in question, or a sufficient time has elapsed since the RACK.packet was S/ACKed, or some combination of the two. More precisely, RACK marks a packet as lost if the reordering window for a packet has elapsed through the sum of:

1. delta in transmit time between a packet and the RACK.packet

2. delta in time between when RACK.ack_ts and now

So we mark a packet as lost if:

```
RACK.xmit_ts > Packet.xmit_ts
        AND
(RACK.xmit_ts - Packet.xmit_ts) + (now - RACK.ack_ts) > RACK.reo_wnd
```

If we solve this second condition for "now", the moment at which we can declare a packet lost, then we get:

```
now > Packet.xmit_ts + RACK.reo_wnd + (RACK.ack_ts - RACK.xmit_ts)
```

Then (RACK.ack_ts - RACK.xmit_ts) is just the RTT of the packet we used to set RACK.xmit_ts, so this reduces to:

```
now > Packet.xmit_ts + RACK.RTT + RACK.reo_wnd
```

The following pseudocode implements the algorithm above. When an ACK is received or the RACK timer expires, call RACK_detect_loss(). The algorithm includes an additional optimization to break timestamp ties by using the TCP sequence space. The optimization is particularly useful to detect losses in a timely manner with TCP Segmentation Offload, where multiple packets in one TSO blob have identical timestamps. It is also useful when the timestamp clock granularity is close to or longer than the actual round trip time.

```
RACK_detect_loss():
min_timeout = 0

For each packet, Packet, in the scoreboard:
    If Packet is already SACKed, ACKed,
       or marked lost and not yet retransmitted:
        Skip to the next packet

    If Packet.xmit_ts > RACK.xmit_ts:
        Skip to the next packet
    /*  Timestamp tie breaker */
    If Packet.xmit_ts == RACK.xmit_ts AND
       Packet.end_seq > RACK.end_seq:
        Skip to the next packet

    timeout = Packet.xmit_ts + RACK.RTT + RACK.reo_wnd + 1
    If Now() >= timeout:
        Mark Packet lost
    Else If (min_timeout == 0) or (timeout is before min_timeout):
        min_timeout = timeout

If min_timeout != 0
    Arm a timer to call RACK_detect_loss() after min_timeout
```

6. Tail Loss Probe: fast recovery on tail losses

This section describes a supplemental algorithm, Tail Loss Probe (TLP), which leverages RACK to further reduce RTO recoveries. TLP triggers fast recovery to quickly repair tail losses that can otherwise only be recoverable by RTOs. After an original data transmission, TLP sends a probe data segment within one to two RTTs. The probe data segment can either be new, previously unsent data, or a retransmission. In either case the goal is to elicit more feedback from the receiver, in the form of an ACK (potentially with SACK blocks), to allow RACK to trigger fast recovery instead of an RTO.

An RTO occurs when the first unacknowledged sequence number is not acknowledged after a conservative period of time has elapsed [RFC6298 [1]]. Common causes of RTOs include:

1. Tail losses at the end of an application transaction.

2. Lost retransmits, which can halt fast recovery if the ACK stream completely dries up. For example, consider a window of three data packets (P1, P2, P3) that are sent; P1 and P2 are dropped. On receipt of a SACK for P3, RACK marks P1 and P2 as lost and retransmits them as R1 and R2. Suppose R1 and R2 are lost as

well, so there are no more returning ACKs to detect R1 and R2 as lost. Recovery stalls.

3. Tail losses of ACKs.

4. An unexpectedly long round-trip time (RTT). This can cause ACKs to arrive after the RTO timer expires. The F-RTO algorithm [RFC5682 [2]] is designed to detect such spurious retransmission timeouts and at least partially undo the consequences of such events (though F-RTO cannot be used in many situations).

6.1. Tail Loss Probe: An Example

Following is an example of TLP. All events listed are at a TCP sender.

(1) Sender transmits segments 1-10: 1, 2, 3, ..., 8, 9, 10. There is no more new data to transmit. A PTO is scheduled to fire in 2 RTTs, after the transmission of the 10th segment. (2) Sender receives acknowledgements (ACKs) for segments 1-5; segments 6-10 are lost and no ACKs are received. The sender reschedules its PTO timer relative to the last received ACK, which is the ACK for segment 5 in this case. The sender sets the PTO interval using the calculation described in step (2) of the algorithm. (3) When PTO fires, sender retransmits segment 10. (4) After an RTT, a SACK for packet 10 arrives. The ACK also carries SACK holes for segments 6, 7, 8 and 9. This triggers RACK-based loss recovery. (5) The connection enters fast recovery and retransmits the remaining lost segments.

6.2. Tail Loss Probe Algorithm Details

We define the terminology used in specifying the TLP algorithm:

FlightSize: amount of outstanding data in the network, as defined in [RFC5681 [3]].

RTO: The transport’s retransmission timeout (RTO) is based on measured round-trip times (RTT) between the sender and receiver, as specified in [RFC6298 [4]] for TCP. PTO: Probe timeout is a timer event indicating that an ACK is overdue. Its value is constrained to be smaller than or equal to an RTO.

SRTT: smoothed round-trip time, computed as specified in [RFC6298 [5]].

Open state: the sender has so far received in-sequence ACKs with no SACK blocks, and no other indications (such as retransmission timeout) that a loss may have occurred.

The TLP algorithm has three phases, which we discuss in turn.

6.2.1. Phase 1: Scheduling a loss probe

Step 1: Check conditions for scheduling a PTO.

A sender should schedule a PTO after transmitting new data or receiving an ACK if the following conditions are met:

(a) The connection is in Open state. (b) The connection is either cwnd-limited (the data in flight matches or exceeds the cwnd) or application-limited (there is no unsent data that the receiver window allows to be sent). (c) SACK is enabled for the connection.

(d) The most recently transmitted data was not itself a TLP probe (i.e. a sender MUST NOT send consecutive or back-to-back TLP probes).

(e) TLPRtxOut is false, indicating there is no TLP retransmission episode in progress (see below).

Step 2: Select the duration of the PTO.

A sender SHOULD use the following logic to select the duration of a PTO:

```
If an SRTT estimate is available:
     PTO = 2 * SRTT
Else:
     PTO = initial RTO of 1 sec
If FlightSize == 1:
     PTO = max(PTO, 1.5 * SRTT + WCDelAckT)
     PTO = max(10ms, PTO)
     PTO = min(RTO, PTO)
```

Aiming for a PTO value of 2*SRTT allows a sender to wait long enough to know that an ACK is overdue. Under normal circumstances, i.e. no losses, an ACK typically arrives in one SRTT. But choosing PTO to be exactly an SRTT is likely to generate spurious probes given that network delay variance and even end-system timings can easily push an ACK to be above an SRTT. We chose PTO to be the next integral multiple of SRTT. Similarly, current end-system processing latencies and timer granularities can easily push an ACK beyond 10ms, so senders SHOULD use a minimum PTO value of 10ms. If RTO is smaller than the computed value for PTO, then a probe is scheduled to be sent at the RTO time.

WCDelAckT stands for worst case delayed ACK timer. When FlightSize is 1, PTO is inflated additionally by WCDelAckT time to compensate for a potential long delayed ACK timer at the receiver. The RECOMMENDED value for WCDelAckT is 200ms, or the delayed ACK interval value explicitly negotiated by the sender and receiver, if one is available.

6.2.2. Phase 2: Sending a loss probe

When the PTO fires, transmit a probe data segment:

```
    If a previously unsent segment exists AND
       the receive window allows new data to be sent:
        Transmit that new segment
        FlightSize += SMSS
        The cwnd remains unchanged
        Record Packet.xmit_ts
    Else:
        Retransmit the last segment
        The cwnd remains unchanged
```

6.2.3. Phase 3: ACK processing

On each incoming ACK, the sender should ancel any existing loss probe timer. The timer will be re-scheduled if appropriate.

6.3. TLP recovery detection

If the only loss in an outstanding window of data was the last segment, then a TLP loss probe retransmission of that data segment might repair the loss. TLP loss detection examines ACKs to detect when the probe might have repaired a loss, and thus allows congestion control to properly reduce the congestion window (cwnd) [RFC5681 [6]].

Consider a TLP retransmission episode where a sender retransmits a tail packet in a flight. The TLP retransmission episode ends when the sender receives an ACK with a SEG.ACK above the SND.NXT at the time the episode started. During the TLP retransmission episode the sender checks for a duplicate ACK or D-SACK indicating that both the original segment and TLP retransmission arrived at the receiver, meaning there was no loss that needed repairing. If the TLP sender does not receive such an indication before the end of the TLP retransmission episode, then it MUST estimate that either the original data segment or the TLP retransmission were lost, and congestion control MUST react appropriately to that loss as it would any other loss.

Since a significant fraction of the hosts that support SACK do not support duplicate selective acknowledgments (D-SACKs) [RFC2883 [7]] the TLP algorithm for detecting such lost segments relies only on basicRFC 2018 [8] SACK support [RFC2018 [9]].

Definitions of variables

TLPRtxOut: a boolean indicating whether there is an unacknowledged TLP retransmission.

TLPHighRxt: the value of SND.NXT at the time of sending a TLP retransmission.

6.3.1. Initializing and resetting state

When a connection is created, or suffers a retransmission timeout, or enters fast recovery, it should reset TLPRtxOut to false

6.3.2. Recording loss probe states

Senders must only send a TLP loss probe retransmission if TLPRtxOut is false. This ensures that at any given time a connection has at most one outstanding TLP retransmission. This allows the sender to use the algorithm described in this section to estimate whether any data segments were lost.

Note that this condition only restricts TLP loss probes that are retransmissions. There may be an arbitrary number of outstanding unacknowledged TLP loss probes that consist of new, previously-unsent data, since the retransmission timeout and fast recovery algorithms are sufficient to detect losses of such probe segments.

Upon sending a TLP probe that is a retransmission, the sender set TLPRtxOut to true and TLPHighRxt to SND.NXT

Detecting recoveries done by loss probes

Step 1: Track ACKs indicating receipt of original and retransmitted segments

A sender considers both the original segment and TLP probe retransmission segment as acknowledged if either (i) or (ii) are true:

(i) This is a duplicate acknowledgment (as defined in [RFC5681 [10]], section 2), and all of the following conditions are met:

(a) TLPRtxOut is true

(b) SEG.ACK == TLPHighRxt

(c) SEG.ACK == SND.UNA

(d) the segment contains no SACK blocks for sequence ranges above TLPHighRxt

(e) the segment contains no data

(f) the segment is not a window update

(ii) This is an ACK acknowledging a sequence number at or above TLPHighRxt and it contains a D-SACK; i.e. all of the following conditions are met:

(a) TLPRtxOut is true

(b) SEG.ACK >= TLPHighRxt and

(c) the ACK contains a D-SACK block

If either conditions (i) or (ii) are met, then the sender estimates that the receiver received both the original data segment and the TLP probe retransmission, and so the sender considers the TLP episode to be done, and records that fact by setting TLPRtxOut to false.

Step 2: Mark the end of a TLP retransmission episode and detect losses

If the sender receives a cumulative ACK for data beyond the TLP loss probe retransmission then, in the absence of reordering on the return path of ACKs, it should have received any ACKs for the original segment and TLP probe retransmission segment. At that time, if the TLPRtxOut flag is still true and thus indicates that the TLP probe retransmission remains unacknowledged, then the sender should presume that at least one of its data segments was lost, so it SHOULD invoke a congestion control response equivalent to the response to any other loss.

More precisely, on each ACK, after executing step (5a) the sender SHOULD reset the TLPRtxOut to false, and invoke the congestion control about the loss event that TLP has successfully repaired.

7. RACK and TLP discussions

7.1. Advantages

The biggest advantage of RACK is that every data packet, whether it is an original data transmission or a retransmission, can be used to detect losses of the packets sent prior to it.

Example: tail drop. Consider a sender that transmits a window of three data packets (P1, P2, P3), and P1 and P3 are lost. Suppose the transmission of each packet is at least RACK.reo_wnd (1 millisecond by default) after the transmission of the previous packet. RACK will mark P1 as lost when the SACK of P2 is received, and this will trigger the retransmission of P1 as R1. When R1 is cumulatively acknowledged, RACK will mark P3 as lost and the sender will retransmit P3 as R3. This example illustrates how RACK is able to repair certain drops at the tail of a transaction without any timer. Notice that neither the conventional duplicate ACK threshold [RFC5681], nor [RFC6675], nor the Forward Acknowledgment [FACK] algorithm can detect such losses, because of the required packet or sequence count.

Example: lost retransmit. Consider a window of three data packets (P1, P2, P3) that are sent; P1 and P2 are dropped. Suppose the transmission of each packet is at least RACK.reo_wnd (1 millisecond by default) after the transmission of the previous packet. When P3 is SACKed, RACK will mark P1 and P2 lost and they will be retransmitted as R1 and R2. Suppose R1 is lost again (as a tail drop) but R2 is SACKed; RACK will mark R1 lost for retransmission again. Again, neither the conventional three duplicate ACK threshold approach, nor [RFC6675], nor the Forward Acknowledgment [FACK] algorithm can detect such losses. And such a lost retransmission is very common when TCP is being rate-limited, particularly by token bucket policers with large bucket depth and low rate limit. Retransmissions are often lost repeatedly because standard congestion control requires multiple round trips to reduce the rate below the policed rate.

Example: (small) degree of reordering. Consider a common reordering event: a window of packets are sent as (P1, P2, P3). P1 and P2 carry a full payload of MSS octets, but P3 has only a 1-octet payload due to application-limited behavior. Suppose the sender has detected reordering previously (e.g., by implementing the algorithm in [REORDER-DETECT]) and thus RACK.reo_wnd is min_RTT/4. Now P3 is reordered and delivered first, before P1 and P2. As long as P1 and P2 are delivered within min_RTT/4, RACK will not consider P1 and P2 lost. But if P1 and P2 are delivered outside the reordering window, then RACK will still falsely mark P1 and P2 lost. We discuss how to reduce the false positives in the end of this section.

The examples above show that RACK is particularly useful when the sender is limited by the application, which is common for interactive, request/response traffic. Similarly, RACK still works when the sender is limited by the receive window, which is common for applications that use the receive window to throttle the sender.

For some implementations (e.g., Linux), RACK works quite efficiently with TCP Segmentation Offload (TSO). RACK always marks the entire TSO blob lost because the packets in the same TSO blob have the same transmission timestamp. By contrast, the counting based algorithms (e.g., [RFC3517][RFC5681]) may mark only a subset of packets in the TSO blob lost, forcing the stack to perform expensive fragmentation of the TSO blob, or to selectively tag individual packets lost in the scoreboard.

7.2. Disadvantages

RACK requires the sender to record the transmission time of each packet sent at a clock granularity of one millisecond or finer. TCP implementations that record this already for RTT estimation do not require any new per-packet state. But implementations that are not yet recording packet transmission times will need to add per-packet internal state (commonly either 4 or 8 octets per packet) to track transmission times. In contrast, the conventional approach requires one variable to track number of duplicate ACK threshold.

7.3. Adjusting the reordering window

RACK uses a reordering window of min_rtt / 4. It uses the minimum RTT to accommodate reordering introduced by packets traversing slightly different paths (e.g., router-based parallelism schemes) or out-of-order deliveries in the lower link layer (e.g., wireless links using link-layer retransmission). Alternatively, RACK can use the smoothed RTT used in RTT estimation [RFC6298]. However, smoothed RTT can be significantly inflated by orders of magnitude due to congestion and buffer-bloat, which would result in an overly conservative reordering window and slow loss detection. Furthermore, RACK uses a quarter of minimum RTT because Linux TCP uses the same factor in its implementation to delay Early Retransmit [RFC5827] to reduce spurious loss detections in the presence of reordering, and experience shows that this seems to work reasonably well.

One potential improvement is to further adapt the reordering window by measuring the degree of reordering in time, instead of packet distances. But that requires storing the delivery timestamp of each packet. Some scoreboard implementations currently merge SACKed packets together to support TSO (TCP Segmentation Offload) for faster scoreboard indexing. Supporting per-packet delivery timestamps is difficult in such implementations. However, we acknowledge that the current metric can be improved by further research.

7.4. Relationships with other loss recovery algorithms

The primary motivation of RACK is to ultimately provide a simple and general replacement for some of the standard loss recovery algorithms [RFC5681][RFC6675][RFC5827][RFC4653] and nonstandard ones [FACK][THIN-STREAM]. While RACK can be a supplemental loss detection on top of these algorithms, this is not necessary, because the RACK implicitly subsumes most of them.

[RFC5827][RFC4653][THIN-STREAM] dynamically adjusts the duplicate ACK threshold based on the current or previous flight sizes. RACK takes a different approach, by using only one ACK event and a reordering window. RACK can be seen as an extended Early Retransmit [RFC5827] without a FlightSize limit but with an additional reordering window. [FACK] considers an original packet to be lost when its sequence range is sufficiently far below the highest SACKed sequence. In some sense RACK can be seen as a generalized form of FACK that operates in time space instead of sequence space, enabling it to better handle reordering, application-limited traffic, and lost retransmissions.

Nevertheless RACK is still an experimental algorithm. Since the oldest loss detection algorithm, the 3 duplicate ACK threshold [RFC5681], has been standardized and widely deployed, we RECOMMEND TCP implementations use both RACK and the algorithm specified in Section 3.2 in [RFC5681] for compatibility.

RACK is compatible with and does not interfere with the the standard RTO [RFC6298], RTO-restart [RFC7765], F-RTO [RFC5682] and Eifel algorithms [RFC3522]. This is because RACK only detects loss by using ACK events. It neither changes the timer calculation nor detects spurious timeouts.

Furthermore, RACK naturally works well with Tail Loss Probe [TLP] because a tail loss probe solicit seither an ACK or SACK, which can be used by RACK to detect more losses. RACK can be used to relax TLP’s requirement for using FACK and retransmitting the the highest-sequenced packet, because RACK is agnostic to packet sequence numbers, and uses transmission time instead. Thus TLP can be modified to retransmit the first unacknowledged packet, which can improve application latency.

7.5. Interaction with congestion control

RACK intentionally decouples loss detection from congestion control. RACK only detects losses; it does not modify the congestion control algorithm [RFC5681][RFC6937]. However, RACK may detect losses earlier or later than the conventional duplicate ACK threshold approach does. A packet marked lost by RACK SHOULD NOT be retransmitted until congestion control deems this appropriate (e.g. using [RFC6937]).

RACK is applicable for both fast recovery and recovery after a retransmission timeout (RTO) in [RFC5681]. The distinction between fast recovery or RTO recovery is not necessary because RACK is purely based on the transmission time order of packets. When a packet retransmitted by RTO is acknowledged, RACK will mark any unacked packet sent sufficiently prior to the RTO as lost, because at least one RTT has elapsed since these packets were sent.

7.6. TLP recovery detection with delayed ACKs

Delayed ACKs complicate the detection of reparies done by TLP, since with a delayed ACK the sender receives one fewer ACK than would normally be expected. To mitigate this complication, before sending a TLP loss probe retransmission, the sender should attempt to wait long enough that the receiver has sent any delayed ACKs that it is withholding. The sender algorithm described above features such a delay, in the form of WCDelAckT. Furthermore, if the receiver supports duplicate selective acknowledgments (D-SACKs) [RFC2883] then in the case of a delayed ACK the sender’s TLP loss detection algorithm (in step (4)(a)(ii), above) can use the D-SACK information to infer that the original and TLP retransmission both arrived at the receiver.

If there is ACK loss or a delayed ACK without a D-SACK, then this algorithm is conservative, because the sender will reduce cwnd when in fact there was no packet loss. In practice this is acceptable, and potentially even desirable: if there is reverse path congestion then reducing cwnd is prudent.

However, in practice sending a single byte of data turned out to be problematic to implement and more fragile than necessary. Instead we use a full segment to probe but have to add complexity to compensate for the probe itself masking losses.

7.7. RACK for other transport protocols

RACK can be implemented in other transport protocols. The algorithm can skip step 3 and simplify if the protocol can support unique transmission or packet identifier (e.g. TCP echo options). For example, the QUIC protocol implements RACK [QUIC-LR] .

8. Experiments and Performance Evaluations

RACK and TLP have been deployed at Google including the connections to the users in the Internet and internally. We conducted an performance evaluation experiment on RACK and TLP on a small set of Google Web servers in western-europe that serve most European and some African countries. The length of the experiments was five days (one weekend plus 3 weekdays) in October 2016, where the servers were divided evenly into three groups.

Group 1 (control): RACK off, TLP off

Group 2: RACK on, TLP off

Group 3: RACK on, TLP on

All groups use Linux using the Cubic congestion control with an initial window of 10 packets and fq/pacing qdisc. In term of specific recovery features, all of them enable RFC3517 (Conservative SACK-based recovery) and RFC5682 (F-RTO) but disable FACK because it is not an IETF RFC. The goal of this setup is to compare RACK and TLP to RFC-based loss recoveries instead of Linux-based recoveries.

The servers sit behind a load-balancer that distributes the connections evenly across the three groups.

Each group handles similar amount of connections and send and receive similar amount of data. We compare total amount of time spent in loss recovery across groups. The recovery time is from when the recovery and retransmit starts, till the remote has acknowledge beyond the highest sequence at the time the recovery starts. Therefore the recovery includes both fast recoveries and timeout recoveries. Our data shows that Group 2 recovery latency is only 2% lower than the Group 1 recovery latency. But Group 3 recovery latency is 25% lower than Group 1 by reducing 40% of the RTOs triggered recoveries! Therefore it is very important to implement both TLP and RACK for performance.

We want to emphasize that the current experiment is limited in terms of network coverage. The connectivities in western-europe is fairly good therefore loss recovery is not a performance bottleneck. We plan to expand our experiments in regions with worse connectivities, in particular on networks with strong traffic policing. We also plan to add the fourth group to disable RFC3517 to use solely RACK and TLP only to see if RACK plus TLP can completely replace all other SACK based recoveries.

9. Security Considerations

RACK does not change the risk profile for TCP.

An interesting scenario is ACK-splitting attacks [SCWA99]: for an MSS-size packet sent, the receiver or the attacker might send MSS ACKs that SACK or acknowledge one additional byte per ACK. This would not fool RACK. RACK.xmit_ts would not advance because all the sequences of the packet are transmitted at the same time (carry the same transmission timestamp). In other words, SACKing only one byte of a packet or SACKing the packet in entirety have the same effect on RACK.

10. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

11. Acknowledgments

The authors thank Matt Mathis for his insights in FACK and Michael Welzl for his per-packet timer idea that inspired this work. Eric Dumazet, Randy Stewart, Van Jacobson, Ian Swett, and Jana Iyengar contributed to the algorithm and the implementations in Linux, FreeBSD and QUIC.

12. References

12.1. Normative References

[RFC793] Postel, J., "Transmission Control Protocol", September 1981.

[RFC2018] Mathis, M. and J. Mahdavi, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.

[RFC6937] Mathis, M., Dukkipati, N., and Y. Cheng, "Proportional Rate Reduction for TCP", May 2013.

[RFC4737] Morton, A., Ciavattone, L., Ramachandran, G., Shalunov, S., and J. Perser, "Packet Reordering Metrics", RFC 4737, November 2006.

[RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, August 2012.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.

[RFC5827] Allman, M., Ayesta, U., Wang, L., Blanton, J., and P. Hurtig, "Early Retransmit for TCP and Stream Control Transmission Protocol (SCTP)", RFC 5827, April 2010.

[RFC5682] Sarolahti, P., Kojo, M., Yamamoto, K., and M. Hata, "Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP", RFC 5682, September 2009.

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

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

[RFC2883] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, July 2000.

[RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, "TCP Extensions for High Performance", September 2014.

12.2. Informative References

[FACK] Mathis, M. and M. Jamshid, "Forward acknowledgement: refining TCP congestion control", ACM SIGCOMM Computer Communication Review, Volume 26, Issue 4, Oct. 1996. , 1996.

[TLP] Dukkipati, N., Cardwell, N., Cheng, Y., and M. Mathis, "Tail Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Drops", draft-dukkipati-tcpm-tcp-loss-probe-01 (work in progress), August 2013.

[RFC7765] Hurtig, P., Brunstrom, A., Petlund, A., and M. Welzl, "TCP and SCTP RTO Restart", February 2016.

[REORDER-DETECT]
Zimmermann, A., Schulte, L., Wolff, C., and A. Hannemann, "Detection and Quantification of Packet Reordering with TCP", draft-zimmermann-tcpm-reordering-detection-02 (work in progress), November 2014.

[QUIC-LR] Iyengar, J. and I. Swett, "QUIC Loss Recovery And Congestion Control", draft-tsvwg-quic-loss-recovery-01 (work in progress), June 2016.

[THIN-STREAM]
Petlund, A., Evensen, K., Griwodz, C., and P. Halvorsen, "TCP enhancements for interactive thin-stream applications", NOSSDAV , 2008.

[SCWA99] Savage, S., Cardwell, N., Wetherall, D., and T. Anderson, "TCP Congestion Control With a Misbehaving Receiver", ACM Computer Communication Review, 29(5) , 1999.

[POLICER16]
Flach, T., Papageorge, P., Terzis, A., Pedrosa, L., Cheng, Y., Karim, T., Katz-Bassett, E., and R. Govindan, "An Analysis of Traffic Policing in the Web", ACM SIGCOMM , 2016.

12.3. URIs

[1] https://tools.ietf.org/html/rfc6298

[2] https://tools.ietf.org/html/rfc5682

[3] https://tools.ietf.org/html/rfc5681

[4] https://tools.ietf.org/html/rfc6298

[5] https://tools.ietf.org/html/rfc6298

[6] https://tools.ietf.org/html/rfc5681

[7] https://tools.ietf.org/html/rfc2883

[8] https://tools.ietf.org/html/rfc2018

[9] https://tools.ietf.org/html/rfc2018

[10] https://tools.ietf.org/html/rfc5681

Authors' Addresses

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

Email: ycheng@google.com

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

Email: ncardwell@google.com

Nandita Dukkipati
Google, Inc
1600 Amphitheater Parkway
Mountain View, California 94043
USA

Email: nanditad@google.com