TCP Maintenance and Minor Extensions (tcpm)
Internet-Draft
Intended status: Experimental
Expires: January 17, 2013
M. Kuehlewind, Ed.
University of Stuttgart
R. Scheffenegger
NetApp, Inc.
July 16, 2012

More Accurate ECN Feedback in TCP
draft-kuehlewind-tcpm-accurate-ecn-01

Abstract

Explicit Congestion Notification (ECN) is an IP/TCP mechanism where network nodes can mark IP packets instead of dropping them to indicate congestion to the end-points. An ECN-capable receiver will feedback 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 ConEx or DCTCP need more accurate ECN feedback information in the case where more than one marking is received in one RTT. This documents specifies a different scheme for the ECN feedback in the TCP header to provide more than one feedback signal per RTT.

Table of Contents

1. Introduction

Explicit Congestion Notification (ECN) [RFC3168] is an IP/TCP mechanism where network nodes can mark IP packets instead of dropping them to indicate congestion to the end-points. An ECN-capable receiver will feedback 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) or DCTCP [Ali10] need more accurate ECN feedback information in case when more than one marking is received in one RTT.

This documents specifies a different scheme for the ECN feedback in the TCP header to provide more than one feedback signal per RTT. This modification does not obsolete [RFC3168]. To avoid confusion we call the ECN specification of [RFC3168] 'classic ECN' in this document. This document provides an extension that requires additional negotiation in the TCP handshake by using the TCP nonce sum (NS) bit, as specified in [RFC3540], which is currently not used when SYN is set. If the more accurate ECN extension has been negotiated successfully, the meaning of ECN TCP bits and the ECN NS bit is different from the specification in [RFC3168] and [RFC3540]. This document specifies the additional negotiation as well as the new coding of the TCP ECN/NS bits.

The proposed coding scheme maintains the given bit space as the ECN feedback information is needed in a timely manner and as such should be reported in every ACK. The reuse will avoid additional network load as the ACK size will not increase. Moreover, the more accurate ECN information will replace the classic ECN feedback if negotiated. Thus those bits are not needed otherwise. But the proposed schemes requires also the use of the NS bit in the TCP handshake as well as for the more accurate ECN feedback itself. The proposed more accurate ECN feedback extension can include the ECN-Nonce integrity mechanism as some coding space is left open. The use of ECN-Nonce is not part of the specification in this document but is discussed in the appendix.

1.1. Use Cases

The following scenarios should briefly show where the accurate feedback is needed or provides additional value:

A Standard (RFC5681) TCP sender that supports ConEx:

In this case the congestion control algorithm still ignores multiple marks per RTT, while the ConEx mechanism uses the extra information per RTT to re-echo more precise congestion information.

A sender using DCTCP congestion control without ConEx:
The congestion control algorithm uses the extra info per RTT to perform its decrease depending on the number of congestion marks.

A sender using DCTCP congestion control and supports ConEx:
Both the congestion control algorithm and ConEx use the accurate ECN feedback mechanism.

A standard TCP sender (using RFC5681 congestion control algorithm) without ConEx:
No accurate feedback is necessary here. The congestion control algorithm still react only on one signal per RTT. But it is best to have one generic feedback mechanism, whether it is used or not.

1.2. Overview ECN and ECN Nonce in IP/TCP

ECN requires two bits in the IP header. The ECN capability of a packet is indicated when either one of the two bits is set. An ECN sender can set one or the other bit to indicate an ECN-capable transport (ECT) which results in two signals, ECT(0) and ECT(1). A network node can set both bits simultaneously when it experiences congestion. When both bits are set the packet is regarded as "Congestion Experienced" (CE).

In the TCP header the first two bits in byte 14 are defined for the use of ECN. The TCP mechanism for signaling the reception of a congestion mark uses the ECN-Echo (ECE) flag in the TCP header. To enable the TCP receiver to determine when to stop setting the ECN-Echo flag, the CWR flag is set by the sender upon reception of the feedback signal. This leads always to a full RTT of ACKs with ECE set. Thus any additional CE markings arriving within this RTT can not signaled back anymore.

ECN-Nonce [RFC3540] is an optional addition to ECN that is used to protect the TCP sender against accidental or malicious concealment of marked or dropped packets. This addition defines the last bit of byte 13 in the TCP header as the Nonce Sum (NS) bit. With ECN-Nonce a nonce sum is maintain that counts the occurrence of ECT(1) packets.

```
  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

1.3. Requirements

The requirements of the accurate ECN feedback protocol for the use of e.g. Conex or DCTCP are to have a fairly accurate (not necessarily perfect), timely and protected signaling. This leads to the following requirements:

Resilience

The ECN feedback signal is carried within the TCP acknowledgment. TCP ACKs can get lost. Moreover, delayed ACK are mostly used with TCP. That means in most cases only every second data packets triggers an ACK. In a high congestion situation where most of the packet are marked with CE, an accurate feedback mechanism must still be able to signal sufficient congestion information. Thus the accurate ECN feedback extension has to take delayed ACK and ACK loss into account.

Timely

The CE marking is induced by a network node on the transmission path and echoed by the receiver in the TCP acknowledgment. Thus when this information arrives at the sender, its naturally already about one RTT old. With a sufficient ACK rate a further delay of a small number of ACK can be tolerated but with large delays this information will be out dated due to high dynamic in the network. TCP congestion control which introduces parts of these dynamics operates on a time scale of one RTT. Thus the congestion feedback information should be delivered timely (within one RTT).

Integrity

With ECN Nonce, a misbehaving receiver or network node can be detected with a certain probability. As this accurate ECN feedback is reusing the NS bit, it is encouraged to ensure integrity as least as good as ECN Nonce. If this is not possible, alternative approaches should be provided how a mechanism using the accurate ECN feedback extension can re-ensure integrity or give strong incentives for the receiver

and network node to cooperate honestly.

Accuracy

Classic ECN feeds back one congestion notification per RTT, as this is supposed to be used for TCP congestion control which reduces the sending rate at most once per RTT. The accurate ECN feedback scheme has to ensure that if a congestion events occurs at least one congestion notification is echoed and received per RTT as classic ECN would do. Of course, the goal of this extension is to reconstruct the number of CE marking more accurately. However, a sender should not assume to get the exact number of congestion marking in all situations.

Complexity

Of course, the more accurate ECN feedback can also be used, even if only one ECN feedback signal per RTT is need. The implementation should be as simple as possible and only a minimum of addition state information should be needed. A proposal fulfilling this for a more accurate ECN feedback can then also be the standard ECN feedback mechanism.

1.4. Design choices

The idea of this document is to use the ECE, CWR and NS bits for additional capability negotiation during the <SYN> / <SYN,ACK> exchange, and then for the more accurate ECN feedback itself on subsequent packets in the flow (where SYN is not set).

Alternatively, a new TCP option could be introduced, to help maintain the accuracy, and integrity of the ECN feedback between receiver and sender. Such an option could provide more information. E.g. ECN for RTP/UDP provides explicit the number of ECT(0), ECT(1), CE, non-ECT marked and lost packets. However, deploying new TCP options has its own challenges. A separate document proposes a new TCP Option for accurate ECN feedback [draft-kuehlewind-tcpm-accurate-ecn-option]. This option could be used in addition to a more accurate ECN feedback scheme described here or in addition to classic ECN, when available and needed.

As seen in Figure 1, there are currently three unused flag bits in the TCP header. The proposed scheme could be extended by one or more bits, to add higher resiliency against ACK loss. The relative gain would be proportionally higher resiliency against ACK loss, while the respective drawbacks would remain identical. Thus the approach in this document is to maintain the scope of the given number of header bits as they seem to be already sufficient. This accurate ECN feedback scheme will only be used instead of the classic ECN and

never in parallel.

1.5. Requirements Language

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

We use the following terminology from [RFC3168] and [RFC3540]:

The ECN field in the IP header:

CE: the Congestion Experienced codepoint, and

ECT(0): the first ECN-Capable Transport codepoint, and

ECT(1): the second ECN-Capable Transport codepoint.

The ECN flags in the TCP header:

CWR: the Congestion Window Reduced flag,

ECE: the ECN-Echo flag, and

NS: ECN Nonce Sum.

In this document, we will call the ECN feedback scheme as specified in [RFC3168] the 'classic ECN' and our new proposal the 'more accurate ECN feedback' scheme. A 'congestion mark' is defined as an IP packet where the CE codepoint is set. A 'congestion event' refers to one or more congestion marks belong to the same overload situation in the network (usually during one RTT).

2. Negotiation during the TCP handshake

During the TCP hand-shake at the start of a connection, an originator of the connection (host A) MUST indicate a request to get more accurate ECN feedback by setting the TCP flags NS=1, CWR=1 and ECE=1 in the initial <SYN>.

A responding host (host B) MUST return a <SYN,ACK> with flags CWR=1 and ECE=0. The responding host MUST NOT set this combination of flags unless the preceding <SYN> has already requested support for

more accurate ECN feedback as above. Normally a server (B) will reply to a client with NS=0, but if the initial <SYN> from client A is marked CE, the sever B SHOULD set the NS flag to 1 to indicate the congestion immediately instead of delaying the signal to the first acknowledgment when the actually data transmission already started. So, server B MAY set the alternative TCP header flags in its <SYN,ACK>: NS=1, CWR=1 and ECE=0.

The addition of ECN to TCP <SYN,ACK> packets is discussed and specified as experimental in [RFC5562]. The addition of ECN to the <SYN> packet is optional. The security implication when using this option are not further discussed here.

This handshake is summarized in Table 1 below, with X indicating NS can be either 0 or 1 depending on whether congestion had been experienced. The handshakes used for the other flavors of ECN are also shown for comparison. To compress the width of the table, the headings of the first four columns have been severely abbreviated, as follows:

Ac: *Ac*curate ECN Feedback

N: ECN-*N*once (RFC3540)

E: *E*CN (RFC3168)

I: Not-ECN (*I*mplicit congestion notification).

Ac	N	E	I	<SYN> A->B	<SYN,ACK> B->A	Mode
				NS CWR ECE	NS CWR ECE	
AB				1 1 1	X 1 0	accurate ECN
A	B			1 1 1	1 0 1	ECN Nonce
A		B		1 1 1	0 0 1	classic ECN
A			B	1 1 1	0 0 0	Not ECN
A			B	1 1 1	X 1 1	Not ECN (broken)

Table 1: ECN capability negotiation between Sender (A) and Receiver (B)

Recall that, if the <SYN,ACK> reflects the same flag settings as the preceding <SYN> (because there is a broken TCP implementation that behaves this way), RFC3168 specifies that the whole connection MUST revert to Not-ECT.

3. More Accurate ECN Feedback

In this section we refer the sender to be the one sending data and the receiver as the one that will acknowledge this data. Of course such a scenario is describing only one half connection of a TCP connection. The proposed scheme, if negotiated, will be used for both half connection as both, sender and receiver, need to be capable to echo and understand the accurate ECN feedback scheme.

This section proposes the new coding of the two ECN TCP bits (ECE/CWR) as well as the TCP NS bit to provide a more accurate ECN feedback. This coding MUST only be used if the more accurate ECN feedback has been negotiated successfully in the TCP handshake.

Section Section 3.4 provides basically another alternative to allow a compatibility mode when a sender needs more accurate ECN feedback but has to operate with a legacy [RFC3168] classic ECN receiver.

3.1. Codepoint Coding

The more accurate ECN feedback coding uses the ECE, CWR and NS bits as one field to encode 8 distinct codepoints. This overloaded use of these 3 header flags as one 3-bit more Accurate ECN (AcE) field is shown in Figure 2. The actual definition of the TCP header, including the addition of support for the ECN Nonce, is shown for comparison in Figure 1. This specification does not redefine the names of these three TCP flags, it merely overloads them with another definition once a flow with more accurate ECN feedback is established.

```
  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 SYN=0).

The 8 possible codepoints are shown below. Five of them are used to encode a "congestion indication" (CI) counter. The other three codepoints are undefined but can be used for some kind of integrity check (see appendix Appendix B). The CI counter maintains the number of CE marks observed at the receiver (see Section 3.3.1).

Also note that, whenever the SYN flag of a TCP segment is set (including when the ACK flag is also set), the NS, CWR and ECE flags

(i.e. the AcE field of the <SYN,ACK>) MUST NOT be interpreted as the 3-bit codepoint, which is only used in non-SYN packets.

AcE	NS	CWR	ECE	CI (base5)
0	0	0	0	0
1	0	0	1	1
2	0	1	0	2
3	0	1	1	3
4	1	0	0	4
5	1	0	1	-
6	1	1	0	-
7	1	1	1	-

Table 2: Codepoint assignment for accurate ECN feedback

By default an accurate ECN receiver MUST echo one of the codepoints encoding the CI counter value. Whenever a CE is received and thus the value of the CI has changed, the receiver MUST echo the CI in the next ACK. Moreover, the receiver MUST repeat the codepoint, that provides the CI counter, directly on the subsequent ACK. Thus every value of CI will be transmitted at least twice. Otherwise the receiver MAY send one of the other, currently undefined, codepoints.

This requirement may conflict with delayed ACK ratios larger than two, using the available number of codepoints. A receiver MUST change the ACK'ing rate such that a sufficient rate of feedback signals can be sent. Details on how the change in the ACK'ing rate can be implemented are given in the section Section 3.3.

3.2. More Accurate ECN TCP Sender

This section specifies the sender-side action describing how to exclude the number of congestion markings from the given receiver feedback signal.

When the more accurate ECN feedback scheme is supported by the sender, the sender will maintain a congestion indication received (CI.r) counter. This CI.r counter will hold the number of CE marks as signaled by the receiver, and reconstructed by the sender.

On the arrival of every ACK, the sender calculates the difference D between the local CI.r value modulo 5, and the signaled CI value of the codepoint in the ACK. The value of CI.r is increased by D, and D is assumed to be the number of CE marked packets that arrived at the receiver since it sent the previously received ACK.

3.3. More Accurate ECN TCP Receiver

This section describes the receiver-side action to signal the accurate ECN feedback back to the sender. The receiver will need to maintain a congestion indication (CI) counter of how many CE marking have been seen during a connection. Thus for each incoming segment with a CE marking, the receiver will increase CI by 1. With each ACK the receiver will calculate CI modulo 5 and set the respective codepoint in the AcE field (see table Table 2). To avoid counter wrap-arounds in a high congestion situation, the receiver SHOULD switch from a delayed ACK behavior to send ACKs immediately after the data packet reception if needed.

3.3.1. Implementation

The receiver counts how many packets carry a congestion notification. This could, in principle, be achieved by directly increasing the CI for every incoming CE marked segment. Since the space for communicating the information back to the sender in ACKs is limited, instead of directly increasing this counter, a "gauge" (CI.g) is increased instead.

When sending an ACK, the CI is increased by either CI.g or at maximum by 4 as a larger increase could cause an overflow in the codepoint counter signaling. Thereafter, CI.g is reduced by the same amount. Then the current CI value (modulo 5) is encoded in the current ACK. To avoid losing information, it must be ensured that an ACK is sent at least after 5 incoming, outstanding congestion marks (i.e. when CI.g exceeds 5). Architecturally the counters never decrease during a TCP session. However, any overflow MUST be modulo a multiple of 5 for CI.

For resilience against lost ACKs, an indicator flag (CI.i) SHOULD be used to ensure that, whether another congestion indication arrives or not, a second ACK transmits the previous counter value again. Thus when a codepoint is transmitted the first time, CI.i will be set to one. Then with the next ACK the same codepoint is transmitted again and the CI.i is reset to zero. Only when CI.i is zero, the counter CI can be increased. In case of heavy congestion (basically all segments are CE marked) the CI.g might grow continuously. In this case the ACK rate should be increased by sending an immediate ACK for an incoming data segment.

The following table provides an example showing an half-connection with a TCP sender A and a TCP receiver B. The sender maintains a counter CI.r to reconstruct the number of CE mark seen at the receiver-side.

	Data	TCP A	IP	TCP B	Data
		SEQ ACK CTL		SEQ ACK CTL	
--		-------------	----------	-------------	
1		0100 SYN	---->		
		CWR,ECE,NS			
2			<----	0300 0101 SYN	
				ACK,CWR	
3		0101 0301 ACK	ECT0 -CE->		
				CI.c=0 CI.g=1	
4	100	0101 0301 ACK	ECT0 ---->		
				CI.c=1 CI.g=0	
5			<----	0301 0201 ACK	
				ECI=CI.1	
		CI.r=1			
6	100	0201 0301 ACK	ECT0 -CE->		
				CI.c=1 CI.g=1	
7	100	0301 0301 ACK	ECT0 -CE->		
				CI.c=1 CI.g=2	
8			XX--	0301 0401 ACK	
				ECI=CI.1	
		CI.r=1			
9	100	0401 0301 ACK	ECT0 -CE->		
				CI.c=1 CI.g=3	
10	100	0501 0301 ACK	ECT0 -CE->		
				CI.c=5 CI.g=0	
11			<----	0301 0601 ACK	
				ECI=CI.0	
		CI.r=5			
12	100	0601 0301 ACK	ECT0 -CE->		
				CI.c=5 CI.g=1	
13	100	0701 0301 ACK	ECT0 -CE->		
				CI.c=5 CI.g=2	
14			<----	0301 0801 ACK	
				ECI=CI.0	
		CI.r=5			

Table 3: Codepoint signal example

3.4. Advanced Compatibility Mode

TBD (more detailed description see draft-ietf-conex-tcp-modifications)

This section describes a possible mechanism to achieve more accurate ECN feedback even when the receiver is not capable of the new more

accurate ECN feedback scheme with the drawback of less reliability.

During initial deployment, a large number of receivers will only support [RFC3168] classic ECN feedback. Such a receiver will set the ECE bit whenever it receives a segment with the CE codepoint set, and clear the ECE bit only when it receives a segment with the CWR bit set. As the CE codepoint has priority over the CWR bit (Note: the wording in this regard is ambiguous in [RFC3168], but the reference implementation of ECN in ns2 is clear), a [RFC3168] compliant receiver will not clear the ECE bit on the reception of a segment, where both CE and CWR are set simultaneously. This property allows the use of a compatibility mode, to extract more accurate feedback from legacy [RFC3168] receivers by setting the CWR permanently.

Assuming a delayed ACK ratio of one (no delayed ACKs), a sender can permanently set the CWR bit in the TCP header, to receive a more accurate feedback of the CE codepoints as seen at the receiver. This feedback signal is however very brittle and any ACK loss may cause congestion information to become lost. Delayed ACKs and ACK loss can both not be accounted for in a reliable way, however. Therefore, a sender would need to use heuristics to determine the current delay ACK ratio M used by the receiver (e.g. most receivers will use M=2), and also the recent ACK loss ratio. Acknowledge Congestion Control (AckCC) as defined in [RFC5690] can not be used, as deployment of this feature is only experimental.

Using a phase locked loop algorithm, the CWR bit can then be set only on those data segments, that will trigger a (delayed) ACK. Thereby, no congestion information is lost, as long as the ACK carrying the ECE bit is seen by the sender.

Whenever the sender sees an ACK with ECE set, this indicates that at least one, and at most M data segments with the CE codepoint set where seen by the receiver. The sender SHOULD react, as if M CE indications where reflected back to the sender by the receiver, unless additional heuristics (e.g. dead time correction) can determine a more accurate value of the "true" number of received CE marks.

4. Acknowledgements

We want to thank Bob Briscoe and Michael Welzl for their input and discussion. Special thanks to Bob Briscoe, who first proposed the use of the ECN bits as one field and the handshake negotiation for more accurate ECN.

5. IANA Considerations

This memo includes no request to IANA.

6. Security Considerations

TBD

ACK loss

This scheme sends each codepoint (of the two subsets) at least two times. In the worst case at least one, and often two or more consecutive   ACKs can be dropped without losing congestion information. Further refinements, such as interleaving ACKs when sending codepoints belonging to the two subsets (e.g. CI, E1), can allow the loss of any two consecutive ACKs, without the sender losing congestion information, at the cost of also reducing the ACK ratio.

At low congestion rates, the sending of the current value of the CI counter by default allows higher numbers of consecutive ACKs to be lost, without impacting the accuracy of the ECN signal.

ECN Nonce

In the proposed scheme there are three more codepoints available that could be used for an integrity check like ECN Nonce. If ECN nonce would be implemented as proposed in Appendix B, even more information would be provided for ECN Nonce than in the original specification.

A delayed ACK ratio of two can be sustained indefinitely even during heavy congestion, but not during excessive ECT(1) marking, which is under the control of the sender. A higher ACK ratio can be sustained when congestion is low, but a low ACK ratio my be needed for the E1 feedback.

7. References

7.1. Normative References

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

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, June 2003.

7.2. Informative References

[Ali10] Alizadeh, M., Greenberg, A., Maltz, D., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and M. Sridharan, "DCTCP: Efficient Packet Transport for the Commoditized Data Center", Jan 2010.

[I-D.briscoe-tsvwg-re-ecn-tcp]
Briscoe, B., Jacquet, A., Moncaster, T., and A. Smith, "Re-ECN: Adding Accountability for Causing Congestion to TCP/IP", draft-briscoe-tsvwg-re-ecn-tcp-09 (work in progress), October 2010.

[RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", RFC 5562, June 2009.

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

[RFC5690] Floyd, S., Arcia, A., Ros, D., and J. Iyengar, "Adding Acknowledgement Congestion Control to TCP", RFC 5690, February 2010.

[draft-kuehlewind-tcpm-accurate-ecn-option]
Kuehlewind, M. and R. Scheffenegger, "Accurate ECN Feedback Option in TCP", draft-kuehlewind-tcpm-accurate-ecn-option-01 (work in progress), Jul 2012.

Appendix A. Estimating CE-marked bytes

TBD (see draft-ietf-conex-tcp-modifications-02 and 'late ACK' scheme of 1 Bit scheme in draft-kuehlewind-tcpm-accurate-ecn-00)

Appendix B. Use with ECN Nonce

In ECN Nonce, by comparing the number of incoming ECT(1) notifications with the actual number of packets that were transmitted with an ECT(1) mark as well as the sum of the sender's two internal counters, the sender can probabilistically detect a receiver that

sends false marks or supresses accurate ECN feedback, or a path that does not properly support ECN.

ECI	NS	CWR	ECE	CI (base5)	E1 (base3)
0	0	0	0	0	-
1	0	0	1	1	-
2	0	1	0	2	-
3	0	1	1	3	-
4	1	0	0	4	-
5	1	0	1	-	0
6	1	1	0	-	1
7	1	1	1	-	2

Table 4: Codepoint assignment for accurate ECN feedback and ECN Nonce

If an ECT(1) mark is received, an ETC(1) counter (E1) is incremented. The receiver has to convey that updated information to the sender with the next possible ACK using the three remaining codepoints as show in table Table 4. Thus on the reception of a ECT(1) marked packet, the receiver should signal the current value of the E1 counter (modulo 3) in the next ACK. If a CE mark was received before sending the next ACK (e.g. delayed ACKs) sending that update MUST take precedence. The receiver should also repeat sending every E1 value. But this repetition does not need to be in the consecutive ACK as the E1 value will only be transmitted when no changes in the CI have occurred. Each E1 value will therefore be sent exactly twice. The repetition of every signal will provide further resilience against lost ACKs.

As only a limited number of E1 codepoints exist and the receiver might not acknowledge every single data packet immediately (delayed ACKs), a sender SHOULD NOT mark more than 1/m of the packets with ECT(1), where m is the ACK ratio (e.g. 50% when every second data packet triggers an ACK). This constraint will avoid a permanent feedback of E1 only, and must be maintained also on short timescales. A sender SHOULD send no more than 3 consecutive packets marked with ECT(1).

The same counter / gauge method as described in Section 3.3.1 can be used to count and return (using a different mapping) the number of incoming packets marked ECT(1) (called E1 in the algorithm). As few codepoints are available for conveying the E1 counter value, an immediate ACK MUST be triggered whenever the gauge E1.g exceeds a threshold of 3. The sender receives the receiver's counter values and compares them with the locally maintained counter.

B.1. Pseudo Code for the Codepoint Coding

```
IP signals: CE
TCP Fields: AcE

Counters:

CI    Congestion Indication - counter [0..(n*5-1)]
CI.g  Congestion Indication - Gauge [0.."inf"])
CI.i  Congestion Indication - indicator flag [0,1]

At session initialization, all these counters are initialized to zero.

When a segment (Data, ACK) is received, perform the following steps:

If (CE)                   # When a CE codepoint is received,
  CI.g++                  # Increase CI.g by 1
If (ECT(1))               # When a ECT(1) codepoint is received,
  E1.g++                  # Increase E1.g by 1
If (CI.g > 5) or          # When ACK rate is not sufficient to keep
   (E1.g > 3)             # gauges close to zero,
  Send ACK immediately    # increase ACK rate


When preparing an ACK to be sent:

If (CI.g > 0) or          # When there is a unsent change in CI
   ( (E1.i != 0) and      # this check is to in effect alternate
   (CI.i != 0) )          # sending CI and E1 codepoints
  If (CI.i == 0) and      # updates to CI allowed
     (CI.g > 0)           # update is meaningful
    CI.i = 1              # set flag to repeat CI value
    CI += min(4,CI.g)     # 4 for 5 codepoints
    CI %= 5               # using modulo the available codepoints
    CI.g -= min(4,CI.g)   # reduce the holding gauge accordingly
  Else
    CI.i--                # just in case CI.f was set to
                          # more than 1 for resiliency
  Send ACK with AcE set to CI
Else
  If (E1.g > 0) or
     (E1.i != 0)
    If (E1.i == 0) and
       (E1.g > 0)
      E1.i = 1
      E1 += min(2, E1.g)
```

```
        E1 %= 3
        E1.g -= min(2, E1.g)
      Else
        E1.i--
      Send ACK with AcE set to E1
    Else
      Send ACK with AcE set to CI  # default action
```

Sender:

Counters:

```
CI.r - current value of CEs seen by receiver
E1.s - sum of all sent ECT(1) marked packets (up to snd.nxt)
E1.s(t) - value of E1.s at time (in sequence space) t
E1.r - value signaled by receiver about received ECT(1) segments
E1.r(t) - value of E1.r at time (in sequence space) t
CI.r(t) - ditto

# Note: With a codepoint implementation,
# a reverse table ECI[n] -> CI.r / E1.r is needed.
# The wire protocol transports the absolute value
# of the receiver-side counter.
# Thus the (positive only) delta needs to be calculated,
# and added to the sender-side counter.

If ACK AcE in the set of CI values
  D = (AcE.CI + 5 - (CI.r mod 5)) mod 5
  CI.r += D
If ACK AcE in the set of E1 values
  D = (Ace.E1 + 3 - (E1.r mod 3)) mod 3
  E1.r += D

# Before CI.r or E1.r reach a (binary) rollover,
# they need to roll over some multiple of 5
# and 3 respectively.

CI.r = CI.r modulo 255   # 5 * 51
E1.r = E1.r modulo 255   # 3 * 85

# (an implementation may choose to use another constant,
# ie 3^4*5^4 (50625) for 16-bit integers,
# or 3^8*5^8 (2562890625) for 32-bit integers)

# The following test can (probabilistically) reveal,
# if the receiver or path is not properly
# handling ECN (CE, E1) marks
```

```
If not E1.r(t) <= E1.s(t) <= E1.r(t) + CI.r(t)

# -> receiver or path do not properly reflect ECN
# (or too many ACKs got lost, which can be checked
# also by the sender).
```

Authors' Addresses

Mirja Kuehlewind (editor)
University of Stuttgart
Pfaffenwaldring 47
Stuttgart 70569
Germany

Email: mirja.kuehlewind@ikr.uni-stuttgart.de

Richard Scheffenegger
NetApp, Inc.
Am Euro Platz 2
Vienna, 1120
Austria

Phone: +43 1 3676811 3146
Email: rs@netapp.com