Congestion Exposure (ConEx)
Internet-Draft
Intended status: Experimental
Expires: June 23, 2012

M. Kuehlewind, Ed.
University of Stuttgart
R. Scheffenegger
NetApp, Inc.

December 21, 2011

Accurate ECN Feedback in TCP
draft-kuehlewind-tcpm-accurate-ecn-00

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 feedback information in the case where more than one marking is received in one RTT.

Status of this Memo

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

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

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

This Internet-Draft will expire on June 23, 2012.

Copyright Notice

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 feedback information in case when more than one marking is received in one RTT.

This documents discusses and (will in a further version specify) 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]. It provides an extension that requires additional negotiation in the TCP handshake by using the TCP nonce sum (NS) bit which is currently not used when SYN is set.

In the current version of this document there are different coding schemes proposed for discussion. All proposed codings aim to scope with the given bit space. All schemes require the use of the NS bit at least in the TCP handshake. Depending of the coding scheme the accurate ECN feedback extension will or will not include the ECN-Nonce integrity mechanism. A later version of this document will choose between the coding options, and remove the rationale for the choice and the specs of those schemes not chosen. If a scheme will be chosen that does not include ECN Nonce, a mechanism that is requiring a more accurate ECN feedback needs to provide an own method to ensure the integrity of the congestion feedback information or has to scope with the uncertainty of this information.

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

a. A Standard TCP sender with [RFC5681] congestion control algorithm 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.

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

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

ECN feedback mechanism.

d. 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 its best to have one generic feedback mechanism, whether you use it or not.

1.1. Overview ECN and ECN Nonce in 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 (ETC) which results in two signals --- ECT(0) and respectively ECT(1). A network node can set both bits simultaneously when it experiences congestion. When both bits are set the packets is regarded as "Congestion Experienced" (CE).

In the TCP header 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.

ECN-Nonce [RFC3540] is an optional addition to ECN that is used to protects the TCP sender against accidental or malicious concealment of marked or dropped packets. This addition defines the last bit of the 13 byte 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.

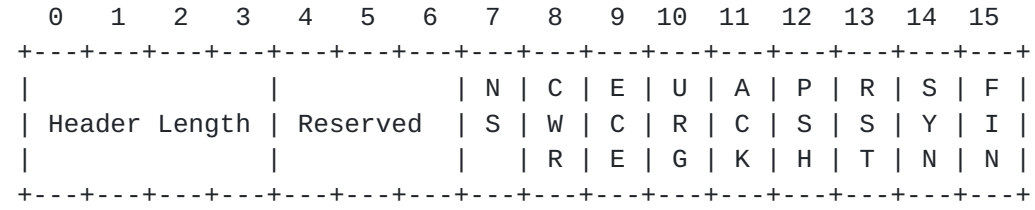

Figure 1: The (post-ECN Nonce) definition of the TCP header flags

1.2. 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 feedback itself on subsequent packets in the flow (with SYN=0).

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 it's own challenges. A seperate documents proposed a new TCP Option for accurate ECN feedback. This option could be used in addition to an more accurate ECN feedback scheme described here or in addtion to the classic ECN, when available and needed.

As seen in Figure 1, there are currently three unused flag bits in the TCP header. Any of the below described schemes could be extended by one or more bits, to add higher resiliency against ACK loss. The relative gains would be proportional to each of the described schemes, while the respective drawbacks would remain identical. Thus the approach in this document is to scope with the given number of bits as they seem to be already sufficient and the accurate ECN feedback scheme will only be used instead of the classic ECN and never in parallel.

1.3. 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)/ECT(1): either one of the two ECN-Capable Transport codepoints.

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 '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 in 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 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 can 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.

These handshakes are 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	1 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 RFC3168 compliant implementation that behaves this way), RFC3168 specifies that the whole connection MUST revert to Not-ECT.

3. Accurate Feedback

In this section we refer the sender to be the on 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.

3.1. Coding

This section proposes three different coding schemes for discussion. First, requirements are listed that will allow to evaluate the proposed schemes against each other. A later version of this document will choose between the coding options, and remove the rationale for the choice and the specs of those schemes not chosen. The next section provides basically a fourth alternative to allow a compatibility mode when a sender needs accurate feedback but has to operate with a legacy [RFC3168] receiver.

3.1.1. 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 implicit carried within the TCP acknowledgment. TCP ACKs can get lost. Moreover, delayed ACK are usually used with TCP. That means in most cases only every second data packets gets acknowledged. 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 this dynamic operates on an time scale of one RTT. Thus the congestion feedback information should be delivered timely (within one RTT).

Integrity

With ECN Nonce, a misbehaving receiver can be detected with a certain probability. As this accurate ECN feedback might reuse 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 RRT 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 a high congestion situation.

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

3.1.2. One bit feedback flag

Remark: In one Acknowledgment all acknowledged bytes are regarded as congested

This option is using a one bit flag, namely the ECE bit, to signal more accurate ECN feedback. Other than classic ECN feedback, a accurate ECN feedback receiver MUST set the ECE bit only in one ACK packets for each one CE received. An more accurate ECN feedback receiver MUST NOT wait for a CWR bit from the sender to reset the ECE bit.

As the CWR would now be unused, the CWR MUST be set in the subsequent ACK after the ECE was set.

CWR(t) = ECE(t-1)

This provides some redundancy in case of ACK loss. If the sender know the ACK'ing scheme of the receiver (e.g. delayed ACKs will send minimum one ACK for every two data packets), the sender can detect ACK loss. If two subsequent ACK or more got lost, the sender SHOULD assume congestion marks for the respective number of ack'ed bytes.

Moreover, when a congestion situation occurs or stops, the receiver MUST immediately acknowledge the data packet and MUST NOT delay the acknowledgment until a further data packet is arrived. A congestion situation occurs when the previous data packet was CE=0 but the current one is CE=1. And a congestion situation stops when the previous data packet was CE=1 and the current one is CE=0.

The following figure shows a simple state machine to describe the receiver behavior.

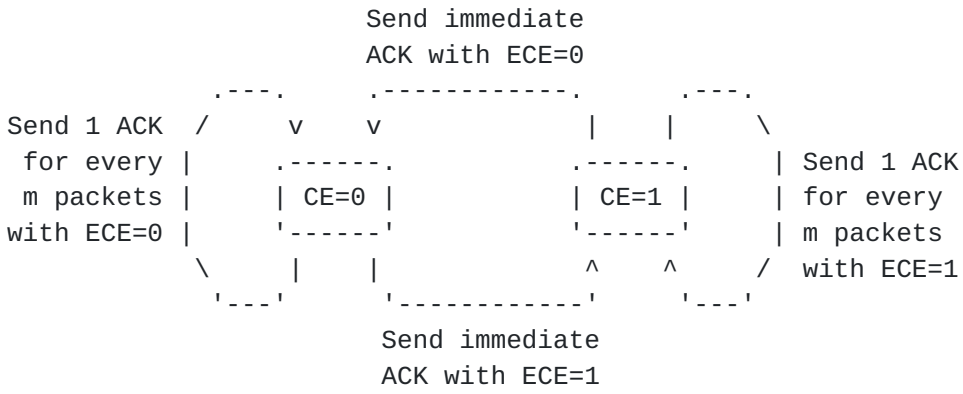

Figure 2: Two state ACK generation state machine

Thus whenever an ACK with the ECE flag set arrives, all acknowledged byte were congestion marked. This scheme provides a byte-wise ECN feedback. The number of CE-marked packet can be estimated by dividing the amount of ack'ed bytes by the Maximum Segment Size (MSS).

When one ACK was lost and the ECN feedback is received based on the CWR set, the sender conservatively SHOULD assume all newly acked bytes as congestion marked.

3.1.2.1. Discussion

ACK loss

In low congestion situations (less than one CE mark per RTT on average), the loss of two subsequent ACKs would result in complete loss of the congestion information. The opposite would be true during high congestion, where the sender can incorrectly assume that all segments were received with the CE codepoint.

One solution would be to carry the same information in a defined number of subsequent ACK packets. This would reduce the number of feedback signals that can be transmitted in one RTT but improve the integrity. More sophisticated solutions based on ACK loss detection might be possible as well.

With DCTCP [Ali10] it was proposed to acknowledge a data packet directly without delay when a congestion situation occurs, as already described above. This scheme allows a more accurate feedback signal in a high congestion/marking situation. However, using delayed ACKs is important for a variety of reasons, including reducing the load on the data sender.

As this heuristic is triggering immediate ACKs whenever the received

CE bit toggles, arbitrarily large ACK ratios are supported. However, the effective ACK ratio is depending on the congestion state of the network. Thus it may collapse to 1 (one ACK for each data segment). More sophisticated solutions based on ACK loss detection might be possible as well, when every other segment is received with CE set.

ECN Nonce

As the ECN Nonce bit is not used otherwise, ECN Nonce [RFC3540] can be used complementary. Network paths not supporting ECN, misbehaving, or malicious receivers withholding ECN information can therefore be detected.

3.1.3. Three bit field with counter feedback

The receiver maintains an unsigned integer counter which we call ECC (echo congestion counter). This counter maintains a count of how many times a CE marked packet has arrived during the half-connection. Once a TCP connection is established, the three TCP option flags (ECE, CWR and NS) are used as a 3-bit field for the receiver to permanently signal the sender the current value of ECC, modulo 8, whenever it sends a TCP ACK. We will call these three bits the echo congestion increment (ECI) field.

This overloaded use of these 3 option flags as one 3-bit ECI field is shown in Figure 3. 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 option flags, it merely overloads them with another definition once a flow with accurate ECN feedback is established.

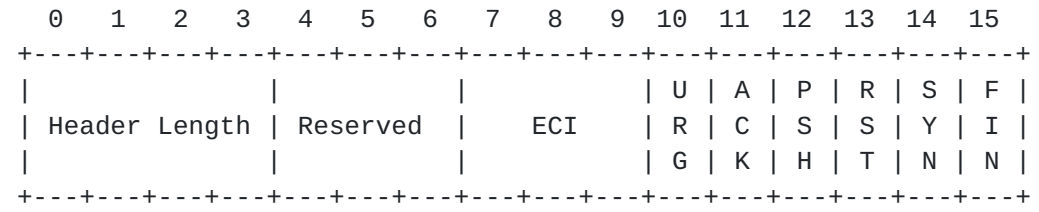

Figure 3: Definition of the ECI field within bytes 13 and 14 of the TCP Header (when SYN=0).

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 ECI field of the SYNACK) MUST NOT be interpreted as the 3-bit ECI value, which is only set as a copy of the local ECC value in non-SYN packets.

This scheme was first proposed in [I-D.briscoe-tsvwg-re-ecn-tcp] for the use with re-ECN.

3.1.3.1. Discussion

ACK loss

As pure ACKs are not protected by TCP reliable delivery, we repeat the same ECI value in every ACK until it changes. Even if many ACKs in a row are lost, as soon as one gets through, the ECI field it repeats from previous ACKs that didn't get through will update the sender on how many CE marks arrived since the last ACK got through.

The sender will only lose a record of the arrival of a CE mark if all the ACKS are lost (and all of them were pure ACKs) for a stream of data long enough to contain 8 or more CE marks. So, if the marking fraction was p, at least 8/p pure ACKs would have to be lost. For example, if p was 5%, a sequence of 160 pure ACKs (without delayed ACKs) would all have to be lost. When ACK are delay this number has to be reduced by 1/m. This would still require a sequence of 80 pure lost ACKs with the usual delay rate of m=2.

Additionally, to protect against such extremely unlikely events, if a re- ECN sender detects a sequence of pure ACKs has been lost it can assume the ECI field wrapped as many times as possible within the sequence. E.g., if a re-ECN sender receives an ACK with an acknowledgement number that acknowledges L (>m) segments since the previous ACK but with a sequence number unchanged from the previously received ACK, it can conservatively assume that the ECI field incremented by D' = L - ((L-D) mod 8), where D is the apparent increase in the ECI field. For example if the ACK arriving after 9 pure ACK losses apparently increased ECI by 2, the assumed increment of ECI would still be 2. But if ECI apparently increased by 2 after 11 pure ACK losses, ECI should be assumed to have increased by 10.

ECN Nonce

ECN Nonce cannot be used in parallel to this scheme. But mechanism that make use of this new scheme might provide stronger incentives to declare congestion honestly when needed. E.g. with ConEx each congestion notification suppressed by the receiver should lead the ConEx audit function to discard an equivalent number of bytes such that the receiver does not gain from suppressing feedback. This mechanism would even provide a stronger integrity mechanism than ECN-Nonce does. Without an external framework to discourage the withholding of ECN information, this scheme is vulnerable to the problems described in [RFC3540].

3.1.4. Codepoints with dual counter feedback

In-line with the definition of the previous section in Figure 3, the ECE, CWR and NS bits are used as one field but instead they are encoding 8 codepoints. These 8 codepoints, as shown below, encode either a "congestion indication" (CI) counter or an ECT(1) counter (E1). These counters maintain the number of CE marks or the number of ECT(1) signals observed at the receiver respectively.

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 2: Codepoint assignment for accurate ECN feedback

By default an accurate ECN receiver MUST echo the CI counter (modulo 5) with the respective codepoints. Whenever an CE occurs 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.

If an ECT(1) mark is receipt and thus E1 increases, the receiver has to convey that updated information to the sender as soon as possible. Thus on the reception of a ECT(1) marked packet, the receiver MUST signal the current value of the E1 counter (modulo 3) in the next ACK, unless a CE mark was receipt which is not echoed yet twice. The receiver MUST also repeat very E1 value. But this repetition does not need to be in the subsequent ACK as the E1 value will only be transmitted when no changes in the CI have occured. Each E1 value will be send excatly 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.

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 should be implemented are given in the next subsection.

3.1.4.1. Implementation

The basic idea is for the receiver to count how many packets carry a congestion notification. This could, in principle, be achieved by increasing a "congestion indication" counter (CI.c) 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 content of this gauge (capped by the maximum number that can be encoded in the ACK, e.g. 4 for CI, and 2 for E1) is copied to the actual counter, and CI.g is reduced by the value that was copied over and transmitted, unless CI.g was zero before. To avoid losing information, it is ensured that an ACK is sent at least after 5 incoming congestion marks (i.e. when CI.g exceeds 5).

For resilience against lost ACKs, an indicator flag (CI.i) ensures that, whether another congestion indication arrives or not, a second ACK transmits the previous counter value again.

The same counter / gauge method is used to count and feed back (using a different mapping) the number of incoming packets marked ECT(1) (called E1 in the algorithm). As fewer 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. Any increase of these counters is added to the sender's internal counters, yielding a precise number of CE-marked and ECT(1) marked packets. Architecturally the counters never decrease during a TCP session. However, any overflow must be modulo 5 for CI, and modulo 3 for E1.

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

3.1.4.2. Discussion

ACK loss

As this scheme sends each codepoint (of the two subsets) at least two times, at least one, and up to two consecutive ACKs can be lost. 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

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 probabilistic detect a receiver that would send false marks or supress accurate ECN feedback, or a path that doesn't properly support ECN.

This approach maintains a balanced selection of properties found in ECN Nonce, Section 3.1.3, and Section 3.1.2. 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. An higher ACK ratios can be sustained even when congestion is low but its need for the E1 feedback.

3.1.5. Short Summary of the Discussions

With the exception of the signaling scheme described in Section 3.1.2, all signaling may fail to work, if middleboxes intervene and check on the semantic of [RFC3168] signals.

The scheme described in Section 3.1.4 is the most complex to implement especially on a receiver, with much additional state to be kept there, compared to the other signaling schemes. With the advances in compute power, many more cycles are available to process TCP than ever before.

Table 4 gives an overview of the relative implications of the different proposed signaling schemes. Further discussion should be included here in the next version of this document.

Section	Resi-liency	Timely	Integrity	Accuracy	Complexity
1-bit-flag	-	+	+	-	+
3-bit-field	++	++	--	++	-
Codepoints	+	+	+	++	--

Table 4: Overview of accurate feedback schemes

Whereas the first scheme is the simplest one (and also provides byte-wise feedback which might be preferable), it has a drawback with respect to reliability. The second one is the most reliable but does not provide an integrity mechanism.

3.2. TCP Sender

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

When the accurate ECN feedback scheme is supported by the receiver, the receiver will maintain an echo congestion counter (ECC). The ECC will hold the number of CE marks received. A sender that is understanding the accurate ECN feedback will be able to reconstruct this ECC value on the sender side by maintaining a counter ECC.r.

On the arrival of every ACK, the sender calculates the difference D between the local ECC.r counter, and the signaled value of the receiver side ECC counter. The value of ECC.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. TCP Receiver

This section will describe the receiver-side action to signal the accurate ECN feedback back to the sender. In any case the receiver will need to maintain a counter of how many CE marking has been seen during a connection. Depending on the chosen coding scheme there will be different action to set the corresponding bits in the TCP header. For all case it might be helpful if the receiver is able to switch form a delayed ACK behavior to send ACKs immediately after the data packet reception in a hight congestion situation.

3.4. Advanced Compatibility Mode

This section describes a possiblity to achieve more accurate feedback even when the receiver is not capable of the new 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 an delayed ACK ratio of one, 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 (l). 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 / (m - l) 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 Michael Welzl and Bob Briscoe for their input and discussion.

5. IANA Considerations

This memo includes no request to IANA.

6. Security Considerations

For coding schemes that increase robustness for the ECN feedback, similar considerations as in RFC3540 apply for the selection of when to sent a ECT(1) codepoint.

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.

Appendix A. Pseudo Code for the Codepoint Coding

Receiver:

Input signals: CE , ECT(1)
TCP Fields: ECI (3-bit field from CWR and ECE). CI.cm and E1.cm map into these 8 codepoints (ie. 5 and 3 codepoints)

```
   These counters get tracked by the following variables:

   CI.c (congestion indication - counter, modulo a multiple of the
   available codepoints to represent CI.c in the ECI field.
   Range[0..n*CI.cp-1])
   CI.g (congestion indication - gauge, [0.."inf"])
   CI.i (congestion indication - iteration, [0,1])
   These are to track CE indications.

   E1.c, E1.g and E1.r (doing the same, but for ECT(1) signals).

   Constants:
   CI.cp (number of codepoints available to signal)
   CI.cm[] (codepoint mapping for CI)
   E1.cp (number of codepoints available for E1 signal)
   E1.cm[0..(E1.cp-1)] (codepoint mappings for E1)

   At session initialization, all these counters are set to 0;

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

   If a CE codepoint is received,
   Increase CI.g by 1
   If a ECT(1) codepoint is received,
   Increase E1.g by 1
   If (CI.g > 5)        # When ACK rate is not sufficient to keep
   or (E1.g > 3)        # gauge close to zero, increase ACK rate
   # works independent of delACK number (ie AckCC)
   Cancel pending delayed ACK (ACK this segment immediately)
   # this increases the ACK rate to a maximum of 1.5 data segments
   # per ACK, with delACK=2,
   # and CE mark rate exceeds 75% for a number
   # of at least 18 segments.
   # 5 codepoints would allow delack=2 indefinitely btw

   When preparing an ACK to be sent:

   If (CI.g > 0) or
   ((E1.i != 0) and (CI.i != 0))  # E1.g = 0 is to skip this
                                  # if only the 2nd CI.c ACK
   # has to be sent - effectively alternating CI.c and E1.c on ACKs
   # should give slightly better resiliency against ack losses
   If CI.i == 0                   # updates to CI.c allowed
   and CI.g > 0                   # update is meaningful
   CI.i = 1                       # may be larger
                                  #if more resiliency is reqd
   CI.c += min(CI.cp-1,CI.g)      # CI.cp-1 is 3 for 4 codepoints,
```

```
                                # 4 for 5 etc
   CI.c = CI.c modulo CI.cp*CI.cp # using modulo the square of
                                # available codepoints,
                                # for convinience (debugging)
   CI.g -= min(CI.cp-1,CI.g)    #
   Else
   CI.i--                       # just in case CI.f was set to
                                # more than 1 for resiliency
   Send next ACK with ECI = CI.cm[CI.c modulo CI.cp]
   Else
   If (E1.g > 0) or (E1.i != 0)

   If (E1.i == 0) and (E1.g > 0)
   E1.i = 1
   E1.c += min(E1.cp-1,E1.g)
   E1.c = E1.c modulo E1.cp*E1.cp
   E1.g -= min(E1.cp-1,E1.g)
   Else
   E1.i--
   Send next ACK with ECI = E1.cm[E1.c modulo E1.cp]
   Else
   Send next ACK with ECI = CI.cm[CI.c modulo CI.cp] # 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.
   # This example is simplified with 4/4 codepoints
   # instead of 5/3

   If ACK with NS=0
   CI.r +=  (ECI + 4 - (CI.r mod CI.cp)) mod CI.cp
   # 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 with NS=1
   E1.r += (ECI + 4 - (E1.r mod E1.cp)) mod E1.c

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

   CI.r = CI.r modulo CI.cp * n_CI
   E1.r = E1.r modulo E1.cp * n_E1

   # (an implementation may choose to use a single constant,
   # ie 3^4*5^4 for 16-bit integers,
   # or 3^8*5^8 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) &lt;= E1.s(t) &lt;= E1.r(t) + CI.r(t)
   # -> receiver lies (or too many ACKs got lost,
   # which can be checked too 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