Transport Area Working Group
Internet-Draft
Intended status: Experimental
Expires: January 3, 2015

B. Briscoe
BT
R. Scheffenegger
NetApp, Inc.
M. Kuehlewind
University of Stuttgart
July 02, 2014

More Accurate ECN Feedback in TCP
draft-kuehlewind-tcpm-accurate-ecn-03

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. Also, to improve robustness it uses 15 more bits if available. For initial experiments it places these in a TCP option. However, if the Urgent flag is cleared, zero header overhead could be achieved by reusing the Urgent Pointer opportunistically. Therefore this document reserves space in the Urgent Pointer to be used if the protocol progresses to the standards track.

This Internet-Draft will expire on January 3, 2015.

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 [I-D.ietf-conex-abstract-mech]) or DCTCP [I-D.bensley-tcpm-dctcp] 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 [I-D.ietf-tcpm-accecn-reqs].

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 we call the ECN specification of [RFC3168] 'classic ECN' in this document.

AccECN is solely an (experimental) change to the TCP wire protocol. It is completely independent of how TCP might respond to congestion feedback. This specification 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.

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, as well as discussing additional security-related issues.

The following aspects are relegated to appendices:

- Appendix A: Pseudocode examples for the various algorithms that AccECN uses;

- Then three appendices for use during document development that will be deleted before publication {ToDo: Delete this list before publication}:
  - Appendix B: Protocol design alternatives that could be considered for inclusion in the main specification;
  - Appendix C: a 'To Do' list of open protocol design issues;
  - Appendix D: Document change log.

1.2. Goals

[I-D.ietf-tcpm-accecn-reqs] 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 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).

Success criteria: The experimental protocol will be considered successful if it satisfies the requirements of [I-D.ietf-tcpm-accecn-reqs] 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, whether or not they comply with standards.

Duration: To be credible, the experiment will need to last at least 12 months from publication of the present specification. At that time, a report on the experiment will be written up. If successful, it would then be appropriate to work on a standards track specification that adds more accurate ECN feedback to TCP.

1.4. Terminology

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

Classic ECN: the ECN scheme as specified in [RFC3168].

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

Pure ACK: A TCP acknowledgement without a data payload.

SupAccECN: The Supplementary Accurate ECN field that provides additional resilience as well as information about the ordering of ECN markings covered by a delayed ACK.

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.

Outgoing AccECN Protocol Handler (or, Outgoing Protocol Handler): The protocol handler at the Data Receiver that marshals the AccECN fields when sending an ACK.

Incoming AccECN Protocol Handler (or, Incoming Protocol Handler): The protocol handler at the Data Sender that reads the AccECN fields when receiving an ACK.

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] requires two bits in the IP header. Prior to the specification of ECN, these two bits were always zero, which is called Not-ECT. An ECN sender can set two possible codepoints (ECT(0) or ECT(1)) to indicate an ECN-capable transport (ECT). It is

prohibited from doing so unless it has checked that the receiver will understand ECN and be able to feed it back. A network node can set both bits simultaneously when it experiences congestion, which is termed 'Congestion Experienced' (CE), or loosely a 'congestion mark'. Table 1 summarises these codepoints.

IP-ECN codepoint (binary)	Codepoint name	Abbrev-iation	Description
00	Not-ECT	N	Not ECN-Capable Transport
01	ECT(1)	1	ECN-Capable Transport (1)
10	ECT(0)	0	ECN-Capable Transport (0)
11	CE	C	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). 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 ECN Nonce [RFC3540] is an optional experimental addition to ECN that the TCP sender can use to protect against accidental or malicious concealment of marked or dropped packets. The sender can send an ECN nonce, which is a continuous pseudo-random pattern of ECT(0) and ECT(1) codepoints in the ECN field. The receiver is required to feed back a 1-bit nonce sum that counts the occurrence of ECT(1) packets using the last bit of byte 13 in the TCP header, which is defined as the Nonce Sum (NS) flag.

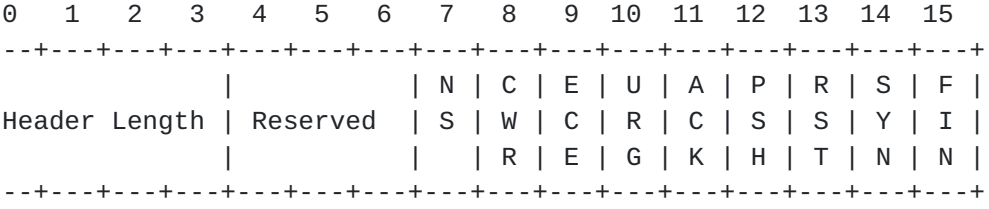


Figure 1: The (post-ECN Nonce) definition of the TCP header flags

2. AccECN Protocol Overview

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

2.1. Essential and Supplementary Parts

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 AccECN protocol has had to be designed in two parts:

- an essential part that provides more accurate ECN feedback than classic ECN with limited resilience against ACK loss;
- a supplementary part that serves three functions:
  - it greatly improves the resilience of AccECN feedback information against loss of ACKs;
  - it provides information about the order in which ECN markings in the IP header arrived at the Data Receiver;
  - it improves the timeliness of AccECN feedback when a delayed ACK covers multiple congestion signals.

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 intact and adding more accurate feedback separately:

- because this efficiently reuses scarce TCP header space, given TCP option space is approaching saturation;
- because a single upgrade path for the TCP protocol is preferable to a fork in the design;
- because 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;
- because 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. The

supplementary part is carried in a field called Supplementary Accurate ECN (SupAccECN).

It is eventually intended that the SupAccECN field would be placed within the main TCP header, by overloading the Urgent Pointer in any segment with URG = 0. However, it would be presumptuous to reassign bits in the main TCP header on an experimental basis. Therefore, this specification reserves sufficient bits within the Urgent Pointer (when URG = 0) for use by AccECN if it reaches the standards track. For the present AccECN experiments, this specification defines an experimental TCP option to carry SupAccECN instead.

When URG = 0, the Urgent Pointer field cannot be used as an Urgent Pointer. Therefore, this specification gives it a new name when URG = 0, defining it as the Non-Urgent field. This specification also establishes an IANA registry for future standards actions to assign values in this newly defined Non-Urgent field.

In order to ease a future transition from experiment to standards track, the Incoming Protocol Handler of all AccECN implementations is required to be able to read the SupAccECN field whether it arrives in a TCP Option or within the Non-Urgent field. However, for the present experimental specification, an AccECN implementation is forbidden from writing into the Non-Urgent field.

Reserving the Non-Urgent field for future use by AccECN is justified, because the Non-Urgent field cannot always be guaranteed to be available. AccECN is unusual in that it is designed to work reasonably well even if the supplementary part is sometimes missing. Therefore, on the rare segments when the Urgent Pointer is needed for its original purpose, URG=1 can still be set and AccECN will still work. However, a future standards action can overload part of the Non-Urgent field for use by AccECN, whenever URG=0.

2.2. 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 client signals support for AccECN on the initial SYN of a connection and the 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 server will interpret them as a request to support the most advanced variant of ECN that it supports. Then the client falls back to the same ECN variant.

The above negotiation uses the three ECN-related flags in the TCP header and determines if both ends support the essential part of

AccECN. On segments after the SYN/ACK, the SupAccECN field is used to determine whether the supplementary part of AccECN is usable over each half-connection. No supplementary part is needed on the initial SYN. A proposal to include a supplementary AccECN field on the SYN/ACK is included in Appendix B.1.

2.3. Two Complementary Feedback Methods

Each AccECN half-connection uses two complementary methods to feed back ECN markings:

Cumulative Counters: A Data Receiver maintains three counters for the number of CE, ECT(1) and Not-ECT codepoints received since the start of the half-connection. In each ACK it places one of these counters, reduced in size by a suitable modulo operation. The Data Sender reads each counter in order to update its own three respective counters, which it uses to track the three counters at the Data Receiver. Of course, each endpoint takes the role of both Data Receiver and Data Sender, so each will maintain three counters as a receiver and three as a sender. AccECN does not provide an explicit count of ECT(0) marks, but this can be inferred from the other feedback;

Sequence List: A list of the codepoints in the IP-ECN field of all the segments covered by a delayed ACK, in the order that they arrived at the Data Receiver. This list also provides timely feedback of any congestion information other than the one covered by the single counter selected.

TCP's traditional feedback is byte-based, whereas AccECN feedback is packet-based, which was a pragmatic choice to reduce feedback overhead, given each packet carries only one ECN mark. AccECN aims to act as a sufficiently generic feedback reflector that can be applied for different uses by different TCP sender behaviours, both existing and in the future.

If a particular sender behaviour needed to associate AccECN's feedback of each ECN marking with the size of the original packet that picked up the marking, there is enough information in AccECN feedback to do so, although perhaps imperfectly. Similarly, if a sender behaviour needed to associate the feedback of each ECN marking with the timing of each packet it originally sent, that too ought to be possible. Of course, the order of arrival at the receiver is not necessarily the order in which packets were sent, and the order in which ACKs return might be different again. So, to apply AccECN to these more challenging tasks, the Data Sender would probably have to record the sizes and/or timings of packets in flight and combine

AccECN feedback with the cumulative acknowledgement numbers on each ACK as well as selective ACK (SACK) information [RFC2018].

Whether such calculations are required or not is outside the scope of the present AccECN specification. The role of AccECN is merely to ensure it would be possible for a Data Sender to reconstruct which segment carried which marking, not to mandate whether it should. As long as AccECN reflects sufficient feedback information without excessive overhead, it fulfils its role. One reason for the experimental status of the present specification is to establish whether the trade-off between accuracy and overhead has been pitched at the right level.

2.4. Resilience Against ACK Loss

Because the counter method repeats one of the accumulating counters on each ACK, if ACKs are lost, a counter in a subsequent ACK will still recover the lost information in a fairly timely fashion.

There is very little space in the 3 bits available for the essential part of an AccECN acknowledgement, so each of the three counters can wrap fairly frequently. Therefore, even if the counter appears to have incremented by one (say), the counter might have actually wrapped completely then incremented by one. This is a possibility because the whole sequence of ACKs carrying the intervening values of the counter might all have been lost or delayed. To be able to tell if a counter has wrapped, AccECN feeds back more significant bits of the counter within the supplementary part, making it resilient to ACK loss.

The supplementary part includes the sequence of ECN codepoints covered by a delayed ACK (see below). As well as providing ordering information, this provides more timely feedback when more than one counter has changed within the time covered by one delayed ACK. It also provides resilience against the loss of a counter in a future ACK.

2.5. Order of Arrival of IP-ECN Markings

[RFC5681] recommends using delayed ACKs, so one acknowledgement will often carry feedback about the ECN markings on more than one segment. Therefore, ideally, AccECN is required to provide ordering information [I-D.ietf-tcpm-accecn-reqs]. However, a counter in each ACK only says how many more IP-ECN markings arrived since the last ACK, not the order in which they arrived.

This might seem an unnecessary level of precision given [RFC5681] currently advises against delaying acknowledgement for more than two

full-sized segments. However, a delayed ACK could cover multiple segments that are smaller than full-size. Also, in practice one delayed ACK can cover many tens of packets that have all been coalesced into one large segment by large receive offload (LRO) hardware before being passed to the Data Receiver. Therefore, the design of AccECN allows for future expansion of the number of segments that can be covered by one delayed ACK.

Once the connection is in progress, in each ACK the Data Receiver encodes the sequence of IP-ECN markings covered by that ACK, which includes the number of segments covered by the delayed ACK. The sequence does not need to include the last segment to arrive, because there is already sufficient information in the essential part of the feedback to infer that marking (by subtracting the markings in the list from the increment of the cumulative counter).

AccECN uses a fixed size (10b) field for the sequence encoding. This can communicate a sequence of up to 14 codepoints, not including the last segment. The encoding is optimised for a selection of simple but common patterns. If the pattern of arriving codepoints becomes too complex to encode in 10b, the Data Receiver has to emit an ACK and start a new sequence for the next ACK. The scheme can always encode all the theoretically possible combinations of arriving codepoints in a delayed ACK covering 3 segments or less.

3. AccECN Protocol Specification

3.1. Negotiation during the TCP handshake

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

If a responding host (B) that implements AccECN receives a SYN with the above three flags set, it MUST set both its half connections into AccECN mode. Then it MUST set the flags NS=0, CWR=1 and ECE=0 on its response in the SYN/ACK segment to confirm that it supports AccECN. The responding host MUST NOT set this combination of flags unless the preceding SYN requested support for AccECN as above.

Once an originating host (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 responding host supports AccECN, the originating host MUST set both its half connections into AccECN mode.

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. To compress the width of the table, the headings of the first four columns have been severely abbreviated, as follows:

Ac: More *Ac*curate ECN Feedback

N: ECN-*N*once [RFC3540]

E: *E*CN [RFC3168]

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

Ac	N	E	I	SYN A->B	SYN/ACK B->A	Mode
				NS CWR ECE	NS CWR ECE	
AB				1 1 1	0 1 0	AccECN
A	B			1 1 1	1 0 1	classic ECN
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)
B	A			0 1 1	0 0 1	classic ECN
B		A		0 1 1	0 0 1	classic ECN
B			A	0 0 0	0 0 0	Not ECN
A				1 1 1	0 1 1	AccECN (Rsvd)
A				1 1 1	1 0 0	AccECN (Rsvd)
A				1 1 1	1 1 0	AccECN (Rsvd)

Table 2: ECN capability negotiation between Originator (A) and Responder (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.

2. The second block shows the cases where the originating host (A) supports AccECN but the responding host (B) supports some earlier variant of TCP, indicated in its SYN/ACK. Therefore, as soon as an originating AccECN-capable host (A) receives the SYN/ACK shown it MUST set both its half connections into the mode shown in the rightmost column.

3. The third block shows the cases where the responding host (B) supports AccECN but the originating host (A) supports some earlier variant of TCP, indicated in its SYN. Therefore, as soon as responding AccECN-capable host (B) receives the SYN shown it MUST set both its half connections into the mode shown in the rightmost column.

4. Forward Compatibility: The fourth block enumerates the remaining combinations of AccECN-related flags that are Reserved for future use by AccECN ('Rsvd').

   * If an originating AccECN host (A) sends NS=1, CWR=1 and ECE=1 in the initial SYN segment and if it receives any of these Reserved values in a SYN/ACK response, it MUST set both its half connections into AccECN mode.

     {ToDo: Can we think of anything now that an AccECN server could use any of these Reserved combinations of flags for, to signal something extra for the whole connection? If not, rather than Reserved, we need to decide whether to make these combinations Rsvd and therefore not switch to AccECN mode.}

   * To comply with the present AccECN protocol, middleboxes MUST forward these Rsvd combinations of flags unaltered (see also Section 3.7).

The table is self-explanatory in most respects, but the following exceptional cases need some explanation.

Not ECN (broken): [RFC3168] points out that broken TCP server implementations exist that reflect the 'reserved' flags [RFC0793] back to the originator. If the SYN/ACK reflects the same flag settings as the preceding SYN, an AccECN client implementation MUST revert to Not-ECT.

ECN Nonce: An AccECN implementation, whether client or server, sender or receiver, does not need to implement the ECN Nonce behaviour [RFC3540]. AccECN is compatible with a sender-only ECN feedback integrity approach that does not use up the ECT(1) codepoint (see Section 3.3.5).

Simultaneous Open: An originating AccECN Host (A), having sent a SYN with NS=1, CWR=1 and ECE=1, might receive another SYN from host B. Host A MUST then enter the same 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 (see the third block above).

3.2. Essential AccECN Feedback

This section specifies the essential part of AccECN feedback, including its placement and the encoding of the counters.

3.2.1. The ACE Field

Once AccECN has been negotiated for a connection, it overloads the three TCP flags ECE, CWR and NS in the main TCP header as one 3-bit field to encode 8 distinct codepoints. Then the field is given a new name, ACE, as shown in Figure 2. 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, it merely overloads them with another name and definition once an AccECN connection has been established.

A host MUST interpret the ECE, CWR and NS flags as the 3-bit ACE counter on a segment with SYN=0 that it sends or receives after it has set both its half-connections 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).

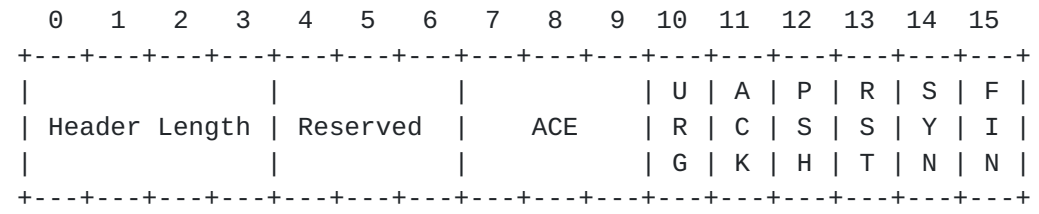


Figure 2: Definition of the ACE field within bytes 13 and 14 of the TCP Header (when AccECN has been negotiated and SYN=0).

The Data Receiver maintains three counters, r.ci, r.e1 and r.ni, to count the number of packets it receives with respectively the CE, ECT(1) and Not-ECT codepoint in the IP-ECN field. When a Data Receiver first enters AccECN mode, it MUST initialise its counters to zero. The Outgoing Protocol Handler at the Data Receiver uses the ACE field to encode one of these counters at a time into each ACK.

How it determines which counter to signal on any particular ACK is specified later (Section 3.2.3).

The 8 possible codepoints of the ACE field are shown in Table 3. A Data Receiver uses four of them to encode a 'Congestion Indication' (CI) counter for CE markings and three to encode E1 for ECT(1) markings. It uses the eighth codepoint to feed back the arrival of Not-ECT in the IP-ECN field using a codepoint termed NI (Not-ECT Indication). We will now use an example to explain how ACE is encoded by the Outgoing Protocol Handler and decoded by the Incoming Protocol Handler.

ACE (base 2)	CI (base 4) for CE	E1 (base 3) for ECT(1)	NI (base 1) for Not-ECT
000	0	-	-
001	1	-	-
010	2	-	-
011	3	-	-
100	-	0	-
101	-	1	-
110	-	2	-
111	-	-	0

Table 3: Codepoint assignments in the ACE field for feedback of congestion counters

Encode: Imagine that the E1 counter is the next to be signalled and r.e1 = 17. Then, because the E1 counter is base 3, the Data Receiver calculates

```
E1 = 17 % 3
   = 2
```

So it looks up E1=2 in Table 3 to get the codepoint to set in ACE, which is 0b110.

Decode: The Data Sender maintains three counters, s.ci, s.e1 and s.ni and it uses the incoming codepoints in ACE to ensure these track the equivalent counters at the receiver. Imagine the s.e1 counter at the Data Sender has currently reached 16 when the 0b110 codepoint arrives via the ACE field. The Data Sender looks up 0b110 in Table 3 to get E1 = 2. It finds the difference between s.e1 and E1 using modulo 3 arithmetic, then adds the difference to s.e1, as follows:

```
delta_s.e1 = (E1 + 3 - s.e1 % 3) % 3
           = (2 + 3 - 16 % 3) % 3
           = 1
=>  s.e1 = s.e1 + delta_s.e1
         = 16 + 1
         = 17
```

3.2.2. Safety against Ambiguity of the ACE Field

Clearly, the CI, E1 and NI counters will frequently wrap given the size of the space available to encode them is so small. If a number of ACKs in a row are lost, the Data Sender might not be able to tell whether one of these counters has wrapped or not.

The supplementary part of AccECN provides more space to signal higher bits of these counters, which gives resilience against ACK loss (Section 3.3.3). However, the supplementary part of the AccECN protocol might be unavailable (perhaps due to middlebox interference).

Therefore, if the Data Sender detects that these fields could have wrapped, it SHOULD behave conservatively. That is, if the AccECN sender detects that the supplementary part of the AccECN protocol is unavailable, and it detects a jump in the acknowledgement number that implies that so many ACKs are missing that a counter could have wrapped under the prevailing conditions, it SHOULD decode the counter assuming that the counter did wrap. If missing acknowledgement numbers arrive later (reordering) and prove that the counter did not wrap, 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.

An example algorithm to implement this policy is given in Appendix A.1. An implementer MAY develop an alternative algorithm as long as it satisfies these requirements.

3.2.3. ACE Counter Selection

If the Data Receiver implements ACK-withholding as recommended in [RFC5681], more than one counter could have incremented before sending each ACK. It follows the steps below to determine which counter to encode in the ACE field:

1. If the last IP-ECN field that arrived was CE, ECT(1) or Not-ECT, the Data Receiver MUST encode the associated counter in the ACE field, i.e. respectively CI, E1 or Not-ECT;

2. If the last IP-ECN field that arrived was ECT(0), the Data Receiver can signal either the CI or the E1 counter:

   * The choice of which to signal SHOULD be based on the principle that the more one counter has changed recently the more it SHOULD be signalled;

   * If there is a tie between CI and E1, CI MUST take precedence.

Appendix A.2 suggests two possible algorithms that could be used to determine which counter to encode in ACE. An implementer MAY develop an alternative algorithm as long as it meets the requirements in the three steps above.

If an AccECN Data Sender has to retransmit a packet due to a suspected loss, in its role as a Data Receiver it will piggy-back AccECN feedback on the retransmitted packet. On a retransmitted packet, a Data Receiver MUST select which counter to send using the rules in the above three steps and encode the latest prevailing value of the selected counter, which will not necessarily be the same counter that the packet carried originally, nor the original value of that counter.

There is no standards track end-to-end definition of the ECT(1) codepoint of the IP-ECN field. Nonetheless, to comply with this specification, an AccECN Data Receiver MUST implement and reflect the ECT(1) counter as specified here. Then, a standards track definition of the ECT(1) codepoint can be defined in future and be deployed unilaterally in Data Senders, without having to wait for associated receivers to be deployed. The above rules ensure that a Data Receiver will only feed back the ECT(1) counter if some packets marked with ECT(1) are arriving.

At the Data Sender, the Incoming AccECN Protocol Handler MUST be able to receive feedback of E1 codepoints, but the Data Sender MAY discard them (it might not have any logic to understand what to do with them). However, if an Incoming AccECN Protocol Handler is running back-to-back with an Outgoing AccECN Protocol handler (e.g. to implement a split TCP connection), it MUST forward the values of all AccECN counters including E1, and not discard any.

{ToDo: Refer if necessary to Section 3.4).

3.3. The Supplementary AccECN Field (SupAccECN)

This section defines the size, placement and internal structure of the Supplementary AccECN field (SupAccECN), as well as the semantics of the sub-fields within it. The internal structure of the SupAccECN

field is agnostic to where it is placed in the TCP header, so that it can be moved during planned evolution of the protocol. The protocol overview in Section 2 explains that the field is placed in a TCP option for initial experiments, but if it progresses to the standards track, it is planned to place it in the main TCP header, using some of the bits in the Urgent Pointer (when URG=0).

3.3.1. Placement of the SupAccECN Field

The Outgoing AccECN Protocol Handler at a Data Receiver MUST place the SupAccECN field in a SupAccECN TCP option (Section 3.3.1.1).

Forward compatibility: If the SupAccECN TCP option (Section 3.3.1.1) is absent, the Incoming AccECN Protocol Handler at a Data Sender MUST attempt to read the SupAccECN field from within the Non-Urgent field (Section 3.3.1.2).

3.3.1.1. The SupAccECN TCP Option

The Data Receiver MUST set the Kind field to 0x<KK> (TBA), which is registered in Section 6.1 as a new TCP option Kind called SupAccECN. An experimental TCP option with Kind=254 MAY be used for initial experiments, with magic number 0xACCE.

The Data Receiver MUST set the Length field to 4 [octets] on any segment with SYN=0. For initial experiments, the Length field MUST be 2 greater to accommodate the 16-bit magic number. In either case, the Data Receiver MUST pad the most significant bit with zeros up to a whole number of octets, as illustrated in Figure 3. This padding bit is currently unused (CU).

Forward compatibility: To comply with the present AccECN specification:

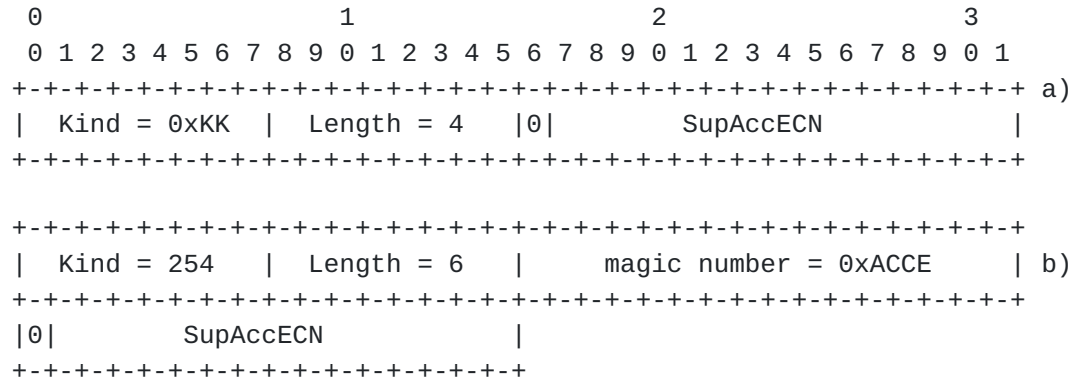
- the Incoming AccECN Protocol Handler at the Data Sender MUST ignore the padding bit, whether it is set to zero or not;

- if the Length field of the TCP option is greater than that expected from the paragraph above, a Data Sender MUST take the SupAccECN field to be aligned with the right hand end (least significant bit) of the TCP Option as calculated using the Length field;

- if the Length value is less than that expected from the paragraph above, the Incoming AccECN Protocol Handler at the Data Sender MUST discard the segment;

- a middlebox MUST forward the padding bit unaltered, whether it is set to zero or not;

- if the Length value is different to that expected from the paragraph above (whether larger or smaller), a middlebox MUST still forward the TCP option unaltered.



a) Using the permanently assigned TCP option Kind 0x<KK> (TBA); b) Using a Shared TCP Option Kind for Initial Experiments

Figure 3: Placement of the SupAccECN field within the SupAccECN TCP Option on a Segment with SYN=0

3.3.1.2. The Non-Urgent Field

If the Urgent (URG) flag in the TCP header [RFC0793] is zero, this specification experimentally renames the Urgent Pointer (bytes 19 and 20 counting from 1 of the TCP header) as the Non-Urgent field. If URG = 1, this 16 bit field keeps its original name and definition from [RFC0793] as the Urgent Pointer. Bytes 13 to 20 of the TCP header when URG=0 are illustrated in Figure 4, which shows the new experimental definition of the Non-Urgent Field.

Note that the new experimental definition of the Non-Urgent field is intended for wider use than just AccECN, which is why it solely depends on the URG flag and it is independent of whether AccECN has been negotiated or not.

Section 6.2 establishes a new registry to assign values within this Non-Urgent field. Section 6.2 also reserves space for a future standards track AccECN specification within this field.

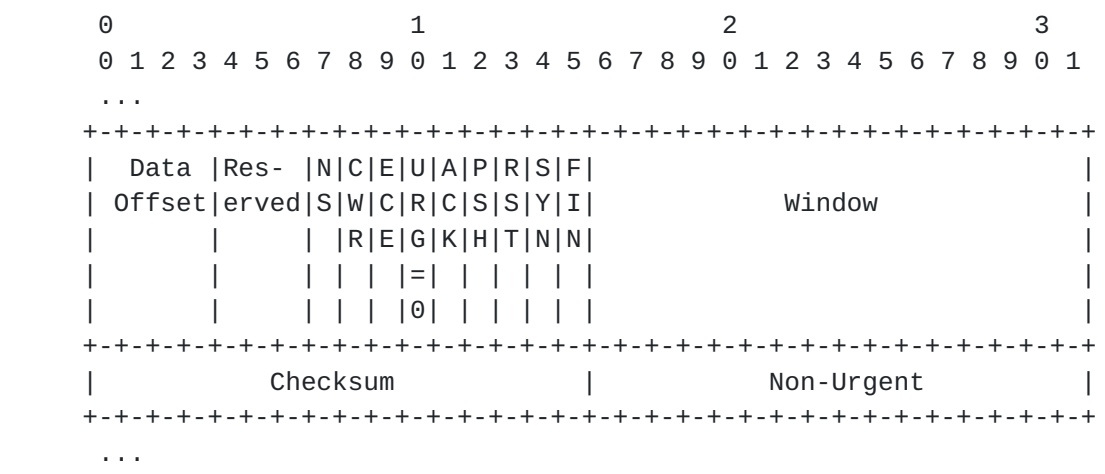


Figure 4: Experimental Renaming of the TCP Urgent Pointer (bytes 19 & 20) as the Non-Urgent field when URG=0

As required in Section 3.3.1, the Outgoing Protocol Handler of the present AccECN specification never writes into the Non-Urgent field. Nonetheless, the Incoming AccECN Protocol Handler can read the SupAccECN field from within the Non-Urgent field.

When reading the Non-Urgent field, AccECN implementations MUST take the SupAccECN field to be right-justified (i.e. the least significant bit of SupAccECN is aligned with the least significant bit of the Non-Urgent Field) as shown in Figure 5. The remaining most significant bit is currently unused (CU).

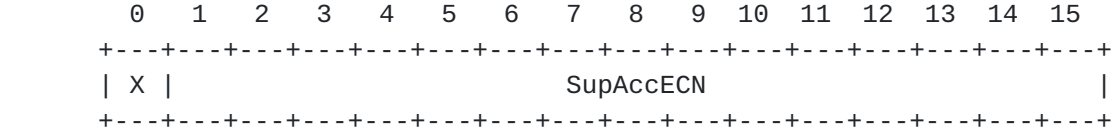


Figure 5: Placement of the SupAccECN field within the Non-Urgent field of a segment with SYN=0

Forward compatibility: To comply with the present AccECN specification:

- the Incoming Protocol Handler of an AccECN Data Sender MUST ignore the remaining most significant bit in the Non-Urgent field (shown as X in Figure 5 meaning "Don't care");

- middleboxes MUST forward the most significant bit unaltered, whether it is set to zero or not.

3.3.2. Structure of the SupAccECN Field

This section defines the structure of the Supplementary AccECN field (SupAccECN) for SYN/ACKs and for subsequent segments within each half-connection. There is no SupAccECN field in the initial SYN segment.

The size of the SupAccECN field on a segment with SYN = 0 is always 15 bits. Figure 6 shows the internal structure of the SupAccECN field on any segment with SYN = 0 including the ACK that ends the 3-way handshake.

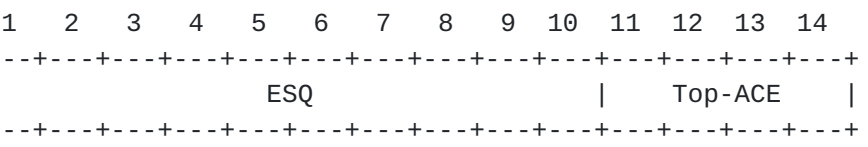


Figure 6: The Supplementary AccECN Field on a Segment with SYN = 0

The sub-fields of SupAccECN on a segment with SYN = 0 have the following meanings:

Top-ACE: Higher significant bits of the counter in ACE within the same segment (defined in Section 3.3.3).

ESQ: The ECN Sequence field (defined in Section 3.3.4).

DAC: Reserved for Delayed ACK Control (see Appendix B.4).

Forward Compatibility: In the meantime, the Outgoing AccECN Protocol Handler MUST set DAC to zero (0); the Incoming AccECN Protocol Handler MUST ignore this flag; and middleboxes MUST forward this flag unaltered whether or not it is zero.

3.3.3. Higher Resilience Congestion Counters (Top-ACE)

Four codepoints are set aside for the CI counter in the ACE field to provide reasonable resilience under expected marking and loss regimes. However, resilience against more extreme levels of CE marking, return ACK loss or ACK thinning really requires more space than the 3 bits taken from existing TCP flags for the ACE counter. At the same time, is it not necessary to deliver higher order bits with every returned segment, or even reliably at all.

Therefore on segments with SYN=0, the least significant four bits of the Supplementary AccECN field are defined as the 'Top ACE' field, as illustrated in Figure 6. Whenever an AccECN implementation encodes a counter in ACE, it MUST also encode the higher precision bits of the

same counter in the Top-ACE field of the same segment, using the following rules:

- Top-ACE MUST be initialised to 0 at the start of each half-connection.

- Whenever the CI counter (base 4) in ACE wraps, the associated Top-ACE MUST increment by 1.

- Similarly, whenever the E1 counter (base 3) in ACE wraps, Top-ACE MUST increment by 1.

- The NI counter in ACE is base 1, so it can hardly be called a counter. The presence of the NI counter in ACE MUST be interpreted as an indication that the associated Top-ACE field in the same segment has incremented, because Top-ACE on its own represents the NI counter.

Formulae for encoding and decoding the counters CI, E1 or NI into the Top-ACE and ACE fields are given in Appendix A.3, which also includes numerical examples.

The 4 bits in the Top-ACE field multiply the number of distinct codepoints for each counter by 2^4 = 16. Using Top-ACE therefore increases the numbers of distinct codepoints for each counter as follows:

Counter	codepoints in ACE	codepoints in Top-ACE with ACE
CI (counts CE)	4	16 * 4 = 64
E1 (counts ECT(1))	3	16 * 3 = 48
NI (counts Not-ECT)	1	16 * 1 = 16

Top-ACE hugely improves the resilience of AccECN against ambiguity of counters due to ACK loss, compared with that of ACE alone (quantified in Appendix A.1). With Top-ACE, the AccECN protocol can lose a whole string of ACKs covering up to 64 - 1 = 63 congestion indications without becoming ambiguous. Similarly AccECN is robust to losing a whole string of ACKs covering 47 ECT(1) markings or 15 Not-ECT markings. If, for example, about 1 in 100 data packets were marked with a CE codepoint on the forward path, all the ACKs covering about 100 * 63 = 6,300 segments would have to be missing from the reverse path before AccECN would become ambiguous. If just one of these ACKs got through, it would resolve any ambiguity.

3.3.4. Accurate ECN Sequence within Delayed ACKs

Given each delayed ACK can cover multiple segments, a Data Receiver needs to describe the order in which the ECN codepoints arrived. AccECN uses a 10-bit ECN Sequence (ESQ) field to encode this ordering. This section explains the encoding. An example encoding algorithm in pseudocode is given in Appendix A.4. Implementations MAY develop their own encoding algorithm as long as it complies with the requirements in this section.

Once the TCP 3-way handshake has completed, an AccECN Data Receiver can defer an ACK until one of these three tests does not pass:

1. The number of deferred bytes exceeds a configured limit (currently two full-sized segments [RFC5681]);

2. The longest time for which an ACK has been delayed exceeds a configured limit (currently 500ms [RFC5681]);

3. The sequence of ECN codepoints has become too complex to encode in the fixed 10b available.

AccECN can encode the order of a sequence of up to 15 ECN codepoints in one ACK. The ACE field in the ACK always encodes the ECN codepoint of the latest packet to arrive. Using the ESQ field of the same ACK, the Outgoing AccECN Protocol Handler can encode the order of arrival of up to 14 ECN codepoints that arrived before this, making a maximum coverage of 15 packets.

The encoding of the ESQ field is optimised for a selection of simple sequences that are expected to be common. Even if the first two tests pass, if a more complex sequence occurs, the third test above will fail so the Data Receiver will be forced to send an ACK earlier than it would have otherwise. The most complex sequence that AccECN can encode is a run of 'spaces' (SP) ending in one 'mark' (MK1), then another run of 'spaces', followed by a 'mark' that might be different from the first (MK2).

The internal structure of the 10-bit Accurate ECN Sequence (ESQ) field is show in Figure 7.

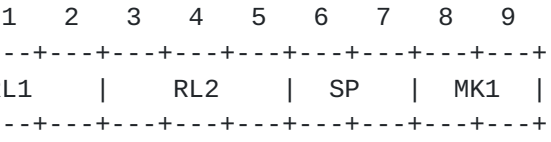


Figure 7: Internal Structure of the Accurate ECN Sequence (ESQ) Field

The sub-fields of ESQ have the following meanings:

RL1: Run-Length #1: a 3-bit field giving the length of a first run consisting of spaces (SP) ending in one mark (MK1), which is included in the length of the run;

RL2: Run-Length #2: another 3-bit field giving the length of a second run of spaces (SP). There is no mark included in this run;

SP: Space: The 2-bit ECN codepoint defined as a space, for the present ACK only;

MK1: Mark #1: The 2-bit ECN codepoint defined as the first mark, for the present ACK only.

The Incoming Protocol Handler can always determine the second mark (MK2) from the counter that the Data Receiver uses in the ACE field, which has to be the counter associated with the last ECN codepoint to have arrived (according to the rules in Section 3.2.3). Even though there is no counter associated with ECT(0), the Incoming Protocol Handler can tell if the last codepoint to arrive was ECT(0), because the counter used in ACE will not have changed relative to the previous packet.

Figure 8 gives example sequences of ECN codepoints and illustrates how the Data Receiver encodes them. The sequences use the single-character abbreviations in Table 1 for each ECN codepoint. The last codepoint to arrive is shown on the right.

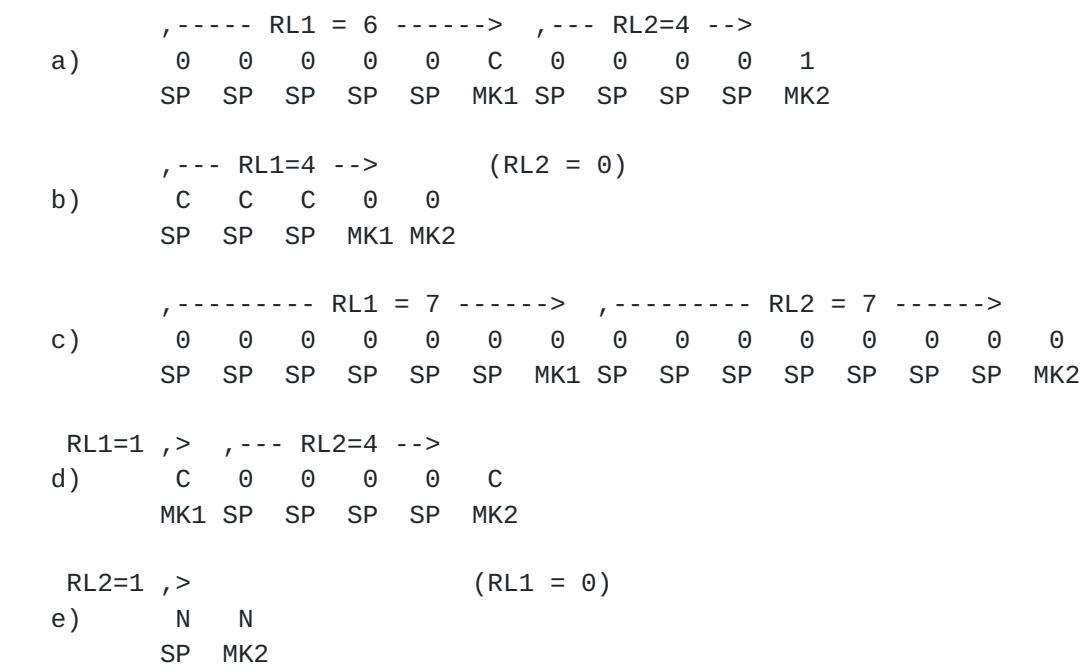


Figure 8: Examples Encodings of Sequences of ECN Codepoints in the ESQ Field

The examples should be self-explanatory, but the following points might help:

- The term 'mark' does not have to mean an 'ECN mark'. In (a) the 'spaces' are defined as ECT(0) and the first 'mark' is defined as CE. However, in (b) it is more efficient to define CE as the 'space' and ECT(0) as the first 'mark';

- A mark is defined to mean just one codepoint, so two marks in a row have to be encoded as two different marks, even if they are the same codepoint (b). The first and second marks can be defined as different (a) or the same (b or c);

- For a long run of the same codepoint, the first mark can be defined to be the same as a space, and if necessary the second mark can be the same as well (c);

- The first run (if non-zero length) always ends in one mark. So, if its run-length is 1, it contains a mark but no spaces (d);

- Either run-length might be zero (b & e), but MK2 will always be present. If the first run-length is zero, the definition of MK1 is redundant (e). If both run-lengths are zero, the definition of SP would be redundant as well.

The following normative statements govern an implementation of an AccECN Data Receiver when it defers an ACK:

- The Outgoing Protocol Handler MUST NOT encode the last packet to be acknowledged into the ESQ field;

- If the Outgoing Protocol Handler cannot encode the last ECN codepoint to arrive in the ESQ field, it MUST send an ACK immediately;

- The Outgoing Protocol Handler MUST NOT include a codepoint in the sequence of codepoints in an ACK that is from any packet already reported in another ACK;

- If RL1=0, the Outgoing Protocol Handler MUST set MK1 = ECT(0) = 0b10, even though the value of MK1 seems redundant.

- If RL2=0 and RL1=<1, the Outgoing Protocol Handler MUST set SP = ECT(0) = 0b10, even though the value of SP seems redundant.

The last two rules ensure that the value of ESQ as a whole is never all-zeros, which allows the Incoming Protocol Handler to detect interference by middleboxes (see Section 3.6).

The following normative statements govern an implementation of an AccECN Data Sender:

- The Incoming AccECN Protocol Handler MUST increment the congestion codepoint counters (other than the one associated with the ACE field) by counting the codepoints as it decodes the ESQ field;

- If the Incoming AccECN Protocol Handler finds that the value of a congestion counter calculated using ESQ would be more than that calculated using Top-ACE/ACE, it SHOULD use the higher of the two calculations.

- If the Incoming AccECN Protocol Handler finds that the value of a congestion counter calculated using ESQ would be less than that calculated using Top-ACE/ACE, it SHOULD use the higher of the two calculations. An example of an exception to this rule would be where the Incoming Protocol Handler had previously conservatively assumed counter wrap, but then missing ACKs arriving later filled the gap in the sequence feedback.

- While the Incoming AccECN Protocol Handler is calculating the value of a congestion counter using Top-ACE/ACE, if it finds that the value calculated using ESQ in a previous segment is already higher, it SHOULD use the lower value calculated using ACE/Top-

ACE. It SHOULD also consider the SupAccECN field in subsequent segments as suspect {ToDo: suggest what concrete action this implies}.

Forward Compatibility:

- if RL1=0:
  - the Incoming Protocol Handler MUST ignore the value in MK1;
  - middleboxes MUST forward the value in MK1 unaltered (whether or not it is 0b10 as it ought to be).
- if RL2=0 and RL1=<1:
  - the Incoming Protocol Handler MUST ignore the value in SP;
  - middleboxes MUST forward the value in SP unaltered (whether or not it is 0b10 as it ought to be).

3.3.5. AccECN Feedback Integrity

The ECN Nonce [RFC3540] is an experimental IETF specification intended to allow a sender to test whether ECN CE markings (or losses) are being suppressed by the receiver (or anywhere else in the feedback loop, such as another network or a middlebox). The ECN nonce has not been deployed as far as can be ascertained. The nonce would now be nearly impossible to deploy retrospectively, because to catch a misbehaving receiver it relies on the receiver volunteering feedback information to incriminate itself. A receiver that has been modified to misbehave can simply claim that it does not support nonce feedback, which will seem unremarkable given so many other hosts do not support it either.

With minor changes AccECN could be optimised for the possibility that the ECT(1) codepoint might be used as a nonce. However, given the nonce is now probably undeployable, 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.

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, 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.

- Networks generate congestion signals when they are becoming congested, so they 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 [I-D.ietf-conex-abstract-mech]. 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.

- 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. Although this section of the feedback loop is the least likely to come under malicious attack, it is increasingly likely to be tampered with accidentally by middleboxes intervening at layer 4. The AccECN fields are immutable end-to-end, so whether placed in the Non-Urgent field or a TCP option, they are amenable to default TCP-AO protection (but not if TCP-AO protection of TCP options is turned off, which is non-default but might be necessary for other reasons).

3.4. Accurate ECN Receiver Operation

A TCP receiver MUST only feedback ECN information arriving in a segment that it deems is part of the flow, by using regular TCP techniques based on sequence numbers.

{ToDo: It might be useful to describe receiver end of the feedback process, including special cases, e.g. pure ACKs, retransmissions, window probes, partial ACKs, etc. Does AccECN feed back each ECN codepoint when a data packet is duplicated?}

3.5. Accurate ECN Sender Operation

A TCP sender MUST only accept ECN feedback on ACKs that it deems is part of the flow, by using regular TCP techniques based on sequence numbers.

{ToDo: It might be useful to describe the sender end of the feedback process, including special cases, e.g. pure ACKs, retransmissions, window probes, partial ACKs, etc.}

3.6. Detection of Legacy Middlebox Interference

The definition of the SupAccECN field has been contrived so that the value all-zeros is undefined. Therefore, an Outgoing AccECN Protocol Handler MUST NOT ever set the value of SupAccECN to all-zeros.

Therefore, the Incoming AccECN Protocol Handler MUST check that the value of ESQ is non-zero (on a segment with SYN=0). If the Incoming Protocol Handler detects all-zeros in either of these fields on any segment, it MUST ignore the whole SupAccECN field on that segment, and it SHOULD ignore the SupAccECN field on all subsequent segments in the same half-connection or at least treat each with greater suspicion.

If a Data Sender ignores the incoming SupAccECN field, it MUST revert to the conservative behaviour needed when only the essential part of the AccECN protocol is available, as described in Section 3.2.2. Nonetheless, the Outgoing AccECN Protocol Handler of the same Data Sender MUST continue to set the SupAccECN field as normal (Section 3.3), because any interference might be only in one direction. The AccECN protocol does not include any requirement for a Data Sender that detects interference to notify the other end, because the complexity required to assure message integrity in the face of interference is not warranted.

3.7. Correct Middlebox Operation

A large class of middleboxes split TCP connections, acting as the receiver for one connection and the sender for another, passing data between the two, usually via a buffer. Network interface hardware to offload certain TCP processing represents another large class of middleboxes, even though it is rarely in its own 'box'.

To comply with this specification, each side of such a middlebox MUST comply with the AccECN requirements applicable to a responding host or an originating host during capability negotiation (Section 3.1) and the required AccECN behaviours as a Data Receiver or as a Data Sender throughout this specification.

Another 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 this specification, such middleboxes MUST be updated to recognise and forward values in fields that comply with the newly defined semantics of AccECN. This includes the explicitly stated requirements to forward Reserved (Rsvd) and Currently Unused (CU) values unaltered. An 'ideal' TCP normaliser would not have to change to accommodate AccECN, because AccECN does not directly contravene any existing TCP specifications, even though it uses existing TCP fields in unorthodox ways.

4. Interaction with Other TCP Variants

4.1. Compatibility with SYN Cookies

A 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 additional state while waiting for the subsequent ACK. 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].

If the server (host B) receives the final ACK of the 3-way handshake with a SupAccECN TCP option, it can infer that the originating host (A) supports AccECN. If host B supports AccECN itself, it can further infer that it would have entered AccECN mode before sending the SYN/ACK.

If, on the other hand, the originating host (A) sends the final ACK of the 3-way handshake with the SupAccECN field in the Non-Urgent field, responding host B can still infer that host A originally negotiated AccECN, by checking the fourteen least significant bits of the Non-Urgent field and the ACE field, as follows:

- Host B knows that host A would not defer the final ACK of the 3-way handshake, because TCP never delays this.

- Therefore, if host B sends the SYN/ACK with its IP-ECN field set to ECT(0) [RFC5562], then checks the fourteen least significant bits of the Non-Urgent field of the final ACK of the 3-way handshake, it can make the following inferences:

  1. lsb(Non-Urgent) == 000010100000 && ACE == 000 implies host A is AccECN and the SYN/ACK arrived unchanged as ECT(0);

2. lsb(Non-Urgent) == 000010100000 && ACE == 001 implies host A is AccECN and the SYN-ACK was CE-marked;

3. lsb(Non-Urgent) == 000010100001 && ACE == 111 implies host A is AccECN and the IP-ECN field of the SYN/ACK was zeroed;

4. lsb(Non-Urgent) == 000000000000 or any value other than those above implies host A is Not AccECN or a middlebox is interfering with the Non-Urgent field.

- If, on the other hand, host B sends the SYN/ACK with its IP-ECN field set to Not-ECT, then checks the fourteen least significant bits of the Non-Urgent field of the final ACK of the 3-way handshake, it can make the following inferences:

1. lsb(Non-Urgent) == 000010100001 && ACE == 111 implies host A is AccECN;

2. lsb(Non-Urgent) == 000000000000 or any value other than that above implies host A is Not AccECN or a middlebox is interfering with the Non-Urgent field.

4.2. Compatibility with Other 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 [I-D.ietf-tcpm-fastopen]) and Multipath TCP (MPTCP [RFC6824]). AccECN is particularly friendly to all these protocols, because space for TCP options is particularly scarce on the SYN, where AccECN consumes zero additional header space.

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 [I-D.ietf-tcpm-accecn-reqs].

Accuracy: From each ACK, the Data Sender can infer the number of new Not-ECT, ECT(0), ECT(1) and CE markings since the previous ACK.

Accuracy: The Data Receiver can feed back to the Data Sender a list of the order of the IP-ECN markings covered by each delayed ACK.

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 requires fifteen bits.

Overhead: Two alternative locations for the supplementary protocol field are proposed:

1. In the 16-bit Urgent Pointer when URG=0. This specification reserves 15 bits of this space, but while the specification is only experimental it refrains from using this space in the main TCP header. If AccECN progresses to the standards track and uses these 15b, it will require zero additional overhead, because it will overload fields that already takes up space in every TCP header

2. In a TCP option. This takes up 4B; the fifteen bits have to be rounded up to 2B, plus 2B for the TCP option Kind and Length.

Timeliness: In the absence of lost ACKs, no feedback is deferred to a future ACK, which is intended to enable latency-sensitive uses of ECN feedback.

Timeliness: {ToDo: Add improved timeliness if the Delayed ACK Control (DAC) feature is included.}

Resilience: Each ACK includes a counter of one of the ECN congestion signals. If ACKs are lost, the counter on the first ACK following the losses allows the Data Sender to immediately recover the number of one of the ECN markings that it missed.

Resilience: Subsequent ACKs will allow it to recover the number of other ECN markings that it missed.

Resilience against Bias: Undetected ACK loss is as likely to decrease as increase congestion signals detected by the Data Sender.

Resilience against Bias: However, if the supplementary part is unavailable, the required conservative decoding of feedback during ACK loss is more likely to increase perceived congestion signals, which would otherwise be more likely to be under-reported.

Timeliness vs Overhead: For efficiency, each delayed ACK only includes one of the counters at a time, therefore recovery of the count of the other signals might not be immediate if an ACK is lost that covers more than one signal. The receiver cannot predict which ACKs might get lost, if any. Therefore it repeats the count of each signal roughly in proportion to how often each signal changes.

Ordering: The order of arriving ECN codepoints is communicated in a 10-bit field in the supplementary part;

Resilience vs. Ordering: Following an ACK loss, only a count of the lost ECN signals is recovered, not their order of arrival over the sequence covered by the loss.

Ordering vs. Overhead: The encoding is tailored for sequences of ECN codepoints expected to be typical. It can encode sequences of up to 15 segments but, if the pattern of arrivals becomes too complex, the protocol forces the Data Receiver to emit an ACK. The protocol can always encode any sequence of 3 segments in one delayed ACK;

Ordering, Timeliness and Resilience: If one delayed ACK covers changes to more than one congestion counter the supplementary sequence information provides more timely congestion feedback than waiting for the other congestion counters on future ACKs, and it provides resilience against the possibility of those future ACKs going missing;

Complexity: {ToDo: Once implemented, quantify the code complexity}

Integrity: AccECN is compatible with complementary protocols that assure the integrity of ECN feedback.

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: Each endpoint can detect normalisation of the Supplementary AccECN field by middleboxes at any time during a connection. It could then fall-back to the essential part using only the fewer but safer bits in the TCP header.

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

6.1. SupAccECN TCP Option Allocation

This specification requires IANA to allocate one value from the TCP option Kind name-space, against the name "Supplementary Accurate ECN" (SupAccECN).

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

6.2. Non-Urgent Field Registry

This specification requests that IANA sets up a new TCP parameters registry in accordance with [RFC5226]. This registry enables future standards track RFCs to assign values to sub-fields of the TCP Non-Urgent field defined in Section 3.3.1.2.

Name of registry: Non-Urgent field.

Information required for assignments:

* Width and position of sub-field or sub-fields,

* Assignment of values to sub-field(s),

* Confirmation of compliance with additional conditions 1 & 2 below.

Review Process: Standards Action - Values to be assigned for Standards Track RFCs approved by the IESG. At the IESG's discretion, values MAY be assigned for Standards Track RFCs still in the process of approval, in order to resolve the catch-22 where the assignment needs deployment testing but deployment testing needs the assignment.

Size, format and syntax of registry entries: Binary values of sub-fields.

Initial assignments and reservations: This specification reserves the 15 least significant bits of the Non-Urgent field for use by a potential future standards action that might define the AccECN scheme for the standards track.

Additional conditions for assignment:

1. Assignments within the Non-Urgent field MUST be used by a protocol that is robust to the field being unavailable occasionally. This is because the Non-Urgent field is unusable and undefined on segments with URG = 1 in the TCP header [RFC0793]. The Non-Urgent field overloads the meaning of the 16-bit Urgent Pointer only when URG = 0.

2. The value zero, i.e. all 16 bits of the Non-Urgent field cleared to zero, SHOULD be undefined, because it is known that certain 'normalising' middleboxes overzealously zero the urgent pointer when URG = 0. An undefined zero value can be achieved by requiring that the value all-zeros is undefined for at least one sub-field of the Non-Urgent field. Then even if the value all-zeros is defined and used in other sub-fields, the value all-zeros for the whole field will be undefined.

7. Security Considerations

If ever the supplementary part of AccECN 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 3.2.2). These problems are unlikely to be due to malicious intervention (because if an attacker could 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 an AccECN Data Sender can detect when the supplementary part is unusable, and it is then required to switch to more conservative assumptions about wrap of congestion indication counters (see Section 3.2.2 and Appendix A.1).

AccECN does not signal the ordering of ECN codepoints covered by a delayed ACK reliably, i.e. if one delayed ACK is lost, the ECN sequence information in that ACK is not retransmitted. The design of AccECN assumes gaps in this information will not be critical, and that this information is unlikely to be security-sensitive. However, this point is mentioned for completeness.

The SYN cookie method for mitigating SYN flooding attacks is not generally compatible with enhancements to the TCP 3-way handshake. Nonetheless, Section 4.1 describes how a server can negotiate AccECN and use SYN cookies.

AccECN is compatible with all the known schemes that ensure the integrity of ECN feedback (see Section 3.3.5 for details). Given the experimental ECN nonce is now probably undeployable, AccECN has been generalised for other possible uses of the ECT(1) codepoint to avoid any risk of obsolescence.

8. Acknowledgements

We want to thank Michael Welzl for his 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.

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

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[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.

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

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

10.2. Informative References

[I-D.bensley-tcpm-dctcp] sbens@microsoft.com, s., Eggert, L., and D. Thaler, "Microsoft's Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters", draft-bensley-tcpm-dctcp-01 (work in progress), June 2014.

[I-D.ietf-conex-abstract-mech] Mathis, M. and B. Briscoe, "Congestion Exposure (ConEx) Concepts and Abstract Mechanism", draft-ietf-conex-abstract-mech-11 (work in progress), March 2014.

[I-D.ietf-tcpm-accecn-reqs]
Kuehlewind, M., Scheffenegger, R., and B. Briscoe, "Problem Statement and Requirements for a More Accurate ECN Feedback", draft-ietf-tcpm-accecn-reqs-05 (work in progress), February 2014.

[I-D.ietf-tcpm-fastopen]
Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", draft-ietf-tcpm-fastopen-09 (work in progress), July 2014.

[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-02 (work in progress), November 2007.

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

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

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, August 2007.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[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.

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

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

Appendix A. Example Algorithms

This appendix is informative, not normative. It gives examples in pseudocode for the various algorithms used by AccECN.

A.1. Example Algorithm for Safety Against Long Sequences of ACK Loss

This appendix gives an example algorithm that a Data Sender can use to heuristically detect a long enough unbroken string of ACK losses that could have concealed wrap of the congestion counter in the ACE field of the next ACK to arrive. The Data Sender is unlikely to need to run an algorithm like this unless it detects that supplementary AccECN feedback is not available (see Section 3.2.2 and Section 3.6).

It is assumed that the focus is solely safety not complete protocol precision. Therefore, this example solely detects possible wrap of the congestion indication (CI) counter, not E1 or NI. This is on the assumption that, even if ECT(1) is redefined to indicate congestion in some way, then ECN CE markings will always indicate more severe congestion. It is also assumed that numerous Not-ECT markings imply middlebox tampering, which only needs to be detected, not quantified perfectly.

If the supplementary Top-ACE field cannot be used, there is only room for 4 values of the congestion indication (CI) counter in the ACE field. The CI counter in an arriving ACK could have wrapped and become ambiguous to the Data Sender if a row of ACKs goes missing that covers a stream of data long enough to contain 4 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 (Section 3.2.3).

If the CE marking probability were p on the forward data path, ambiguity would arise if 100% of ACKs went missing from the reverse path in a row was at least 4/p long. For example, if p was 5% on the forward path, ambiguity would ensue if simultaneously on the reverse path a sequence of ACKs covering 4/0.05 = 80 packets all went missing. With a delayed ACK ratio of 2 that translates to missing 40 ACKs in a row. Obviously, missing ACKs would be far less likely if pure ACKs were allowed to be ECN-capable. However, because RFC 3168 currently precludes this, we will assume that pure ACKs are not ECN-capable.

To protect against such an unlikely event, Section 3.2.2 requires the Incoming Protocol Handler to assume that the CI field did wrap if it

could have wrapped under prevailing conditions. It could be extremely conservative and assume that ECN marking suddenly jumped to 100% on the forward path just when there were no ACKs on the reverse path to detect it.

Specifically, if the Incoming Protocol Handler receives an ACK with an acknowledgement number that acknowledges L full-sized segments since the previous ACK, it could conservatively assume that the CI field incremented by

$$D' = L - ((L-D) \% 4),$$

where D is the apparent increase in the CI field. This would still be safe if segments were 5% of full-sized as long as ECN marking was 5% or less, not 100%.

For example, imagine an ACK acknowledges 5 more full-size segments than any previous ACK, and that it apparently increases CI by 2. The above formula works out that a safe increment of CI would still be 2 (because 5 - ((5-2) % 4) = 2). However, if CI apparently increases by 2 but acknowledges 11 more full-sized segments, then CI should be assumed to have increased by 10 (because 11 - ((11-2) % 4) = 10).

Implementers could build in more heuristics to estimate prevailing segment sizes and prevailing ECN marking. For instance, L in the above formula could be replaced with L' = L*p*M/s, where M is the MSS, 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 like that given initially, rather than a perfect one.

If missing acknowledgement numbers arrive later (due to reordering), Section 3.2.2 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. Example Counter Selection Algorithms

When the Data Receiver sends an ACK, if the last IP-ECN field that arrived was ECT(0), Section 3.2.3 says, "...the Data Receiver can signal either the CI or the E1 counter. The choice of which to signal SHOULD be based on the principle that the more one counter has changed recently the more it SHOULD be signalled." A couple of

alternative algorithms are suggested below that would satisfy this requirement.

A.2.1. Counter Selection Algorithm Alt#1

Counter selection algorithm Alt#1 repeats whichever counter has been repeated proportionately less often, relative to how often it has changed, with preference for CI if they tie. Or in pseudocode:

```
if ( (e1 / r_e1) > (ci / r_ci) )
    send_ack(e1)
else
    send_ack(ci)
```

where r_e1 and r_ci are counts of how often E1 and CI were already repeated when ECT(0) was signalled. The algorithm below implements this comparison between two divisions using only integer addition. It is a little terse, so it is explained afterwards.

```
   ci   = 0    // CE counter
   w_ci = 0    // internal 'weight' variable for CI
   r_ci = 0    // internal count of how often CI has been repeated
   e1   = 0    // ECT(1) counter
   w_e1 = 0    // internal 'weight' variable for E1
   r_e1 = 0    // internal count of how often E1 has been repeated
   ni   = 0    // Not-ECT counter

   dack_to_be_sent()    // shorthand for test if a delayed ACK is needed

   switch (read(pkt.ip.ecn)) {
       case CE :
           ci++
           w_ci += r_e1
           if (dack_to_be_sent()) send_ack(ci)
       case ECT1 :
           e1++
           w_e1 += r_ci
           if (dack_to_be_sent()) send_ack(e1)
       case Not-ECT :
           ni++
           if (dack_to_be_sent()) send_ack(ni)
       case ECT0 :
           if (dack_to_be_sent()) {
               /* Choice between E1 and CI */
               if (w_e1 > w_ci) {      // Preference to CI if they tie
                   send_ack(e1)
                   r_e1++
                   w_ci += ci
               } else {
                   send_ack(ci)
                   r_ci++
                   w_e1 += e1
               }
           }
   }

   {ToDo: Handle wrap of the weights (see my notebook?).}

   Explanation: The algorithm ensures that the weights always equal the
   following products:

       w_ci = ci * r_e1,
       w_e1 = e1 * r_ci.

   It does this by incremental addition rather than multiplication:

   o  every time r_e1 increments by 1, w_ci is incremented by 1 * ci;
```

o every time ci increments by 1, w_ci is incremented by 1 * r_e1;

and the same for w_e1 and the pair of variables it consists of.

This ensures that the condition

w_e1 > w_ci

used in the algorithm is equivalent to:

e1 * r_ci > ci * r_e1,

or rearranging:

(e1 / r_e1) > (ci / r_ci),

which is the required proportionality condition.

A.2.2. Counter Selection Algorithm Alt#2

Counter selection algorithm Alt#2 implements the policy "Send each recently changed codepoint twice, unless the other one has also changed, and alternate sending CI, E1 if no counter changes."

{ToDo: Alt#2 has the disadvantage that it can repeat E1 a lot, even if E1 has never been signalled, which unnecessarily reduces the resilience of CI.

```
   ci   = 0        // CE counter
   q_ci = 0        // queue of CI's to repeat
   nxt_ci = TRUE   // Signal E1 next if FALSE
   e1   = 0        // ECT(1) counter
   q_e1 = 0        // queue of E1's to repeat
   ni   = 0        // Not-ECT counter

   dack_to_be_sent()    // shorthand for test if a delayed ACK is needed

   switch (read(pkt.ip.ecn)) {
       case CE :
           ci++
           q_ci = 2
           if (dack_to_be_sent()) send_ack(ci)
       case ECT1 :
           e1++
           q_e1 = 2
           if (dack_to_be_sent()) send_ack(e1)
       case Not-ECT :
           ni++
           if (dack_to_be_sent()) send_ack(ni)
       case ECT0 :
           if (dack_to_be_sent()) {
               /* Choice between E1 and CI */
               if (q_ci || q_e1) {     // If either queue is non-zero
                   if (q_e1 > q_ci) {  // Preference to CI if they tie
                       send_ack(e1)
                       q_e1 = max(0, q_e1 - 1)
                   } else {
                       send_ack(ci)
                       q_ci = max(0, q_ci - 1)
                   }
               } else {            // Both queues are zero
                   if (nxt_ci)
                       send_ack(ci)
                   else
                       send_ack(e1)
                   nxt_ci = !nxt_ci    // Toggle the next signal
               }
           }
   }
```

A.3. Example Encodings and Decodings of Top-ACE and ACE

This appendix gives formulae for encoding and decoding the counters CI, E1 or NI with higher resilience to ACK loss by supplementing the ACE field with the Top-ACE field, as required in Section 3.3.3.

A.3.1. Encoding Top-ACE and ACE by the Data Receiver

The values associated with codepoints in ACE for CI and E1 are respectively base 4 and base 3 numbers (see Table 3). Although there is only space for one value of NI, mathematically, NI can still be treated as a base 1 counter. Then the following general formulae allow a Data Receiver to encode any of the counters CI, E1 or NI, by calling them all cntr, and defining ACE_base as their respective number base:

```
 Top-ACE = Int(cntr / ACE_base) % 16,
ACE_cntr = cntr % ACE_base.
```

Then the Data Receiver looks up the codepoint to put in the ACE field by looking up ACE_cntr in Table 3 in the column of the relevant counter (CI, E1 or NI). Int() means round down to an integer and '%' is the modulo operator.

To implement this without a costly division operation, two counters can be maintained while processing the header information for the ACK. The first counter can be mapped into the ACE field via Table 3. A wrap every 4 increments of the counter could be implemented as a single conditional check, and when it wraps, a secondary, high-order counter could be incremented. This secondary counter could then be mapped directly into the Top ACE field. For instance, the two counters for CE markings would be implemented as follows:

```
if (read(pkt.ip.ecn) == CE) {
    if (ACE_cntr.ci == 4) {
        ACE_cntr.ci = 0
        if (Top-ACE.ci == 16) {
            Top-ACE.ci = 0
        } else
            Top-ACE.ci++
    } else
       ACE_cntr.ci++
}
```

The three examples below explain how the algorithm determines which codepoints to place in Top-ACE and ACE, for each counter in turn. For brevity, they use the first mathematical formula above, rather than the second conditional logic variant.

Example #1: if the Data Receiver has determined that it will signal its CI counter next and its local value is 73, it encodes this as:

```
    Top-ACE = INT(73 / 4) % 16
            = 2
            = 0b0010
    ACE_cntr = 73 % 4
             = 1
```

Looking up the codepoint for CI = 1 in Table 3 gives:

```
    ACE = 0b001.
```

Example #2: if the Data Receiver has determined that it will signal its E1 counter next and its local value is 75, it encodes this as:

```
    Top-ACE = INT(75 / 3) % 16
            = 9
            = 0b1001
    ACE_cntr = 75 % 3
             = 0
```

Looking up the codepoint for E1 = 0 in Table 3 gives:

```
    ACE = 0b100.
```

Example #3: if the Data Receiver has determined that it will signal its NI counter next and its local value is 43, it encodes this as:

```
    Top-ACE = INT(43 / 1) % 16
            = 11
            = 0b1011
    ACE_cntr = 43 % 1
             = 0              // Anything modulo 1 is 0
```

Looking up the codepoint for NI = 0 in Table 3 gives:

```
    ACE = 0b111.
```

A.3.2. Decoding Top-ACE and ACE by the Data Sender

An AccECN Data Sender decodes the incoming combination of Top-ACE and ACE by looking up the ACE codepoint in Table 3 to get ACE_cntr and ACE_base, then:

```
    cntr = Top-ACE * ACE_base + ACE_cntr.
```

For example, if ACE = 0b101 and Top-ACE = 0b0111 = 7, the Data Sender looks up ACE = 0b101 in Table 3 to see that this is the E1 counter and that ACE_cntr = 1 base 3. Therefore,

```
    E1 = cntr = 7 * 3 + 1
              = 22
```

The Data Sender is likely to be primarily interested in the increment in this counter relative to the previous ACK. In the case of E1, it will have to use modulo 48 arithmetic for the difference, because the encoding wraps at 48 (see Table 4). Specifically, if the Data Sender's local counter is snd_e1, then the difference,

```
    delta_e1 = (E1 + 48 - snd_e1 % 48) % 48
```

{ToDo: Provide algorithms that decode correctly with ACK reordering}

A.4. Example ECN Sequence (ESQ) Encoding Algorithms

This appendix gives an example algorithm for the Data Receiver to encode the arriving sequence of IP-ECN codepoints in the ECN Sequence (ESQ) field of a delayed ACK, as required in Section 3.3.4.

```
/* Algorithm to encode the arrival sequence of IP-ECN codepoints
 */
DEFAULT = ECT0      // Any ECN codepoint except Not-ECT
DACK_T_MAX = 500    // Max time to delay an ACK [ms]
RL_MAX = 7          // Max run-length that can fit in 3-bit field
DACK_SEG_MAX = 2    // Max full-sized delayed ACK segments:
MSS = 1500          // Example max segment size [B]
DACK_B_MAX = DACK_SEG_MAX * MSS     // Max deferred bytes

sp = mk1 = DEFAULT  // 2-bit ECN codepoints: space and mark
mk2                 // second mark (fed back in ACE, not ESQ)
rl1 = rl2 = 0       // 3-bit run-lengths
dack_b = 0          // deferred bytes

/* Strategy: in readiness for a packet arrival, hold the variables
 *  necessary to build the ECN sequence field (ESQ) of the next ACK.
 * If a packet arrives, and it can be added to the held sequence,
 *  do so and return.
 * If it can't be added to the held sequence, send the ACK
 *  with the most recent packet as the second mark.
 * If the delayed ack timer expires, unwind the last packet in the
 * held sequence to use as the second mark, and send the ACK
 */

foreach pkt {
    tmp = read(pkt.ip.ecn)      // Store incoming ECN field
    dack_b += read(pkt.ip.size) // Add to deferred bytes

    if (dack_b >= DACK_B_MAX) { // Test deferred bytes threshold
```

```
          mk2 = tmp               // Assign incoming ECN to mk2
          send_ack(rl1,rl2,sp,mk1,mk2)     // Encode ESQ and send ACK
      } elif ((rl1 + rl2) =< 0) { // Is the held sequence empty?
          sp = tmp                // Initialise with a space in run2
          rl2++
          init_timer(dack_expire, DACK_T_MAX) // Arm delayed ACK timer
      } elif (tmp == sp) {        // Is the incoming ECN another space?
          if (rl2 < RL_MAX) {     // Is there room in run2?
              rl2++               // Extend run2
          } elif (rl1 =< 0) {     // Otherwise, is run1 empty?
              mk1 = sp            // Shift run2 to run1, making mk1=sp
              rl1 = rl2
              rl2 = 1
          }
      /* If got to here, incoming ECN is assigned as a mark */
      } elif (rl1 =< 0) {     // If there's room in run1, switch to it
          mk1 = tmp
          rl1 = rl2
          rl2 = 0
      } elif ( (tmp == mk1)   // Is incoming ECN a mark already seen
            && (rl1 = 2)      //  with only one space before it?
            && (rl2 = 0) ) {
          mk1 = sp            // If so, swap marks with spaces
          sp = tmp
          rl1 = 1
          rl2 = 2
      } else {                // Cannot extend sequence
          mk2 = tmp           // Assign the incoming ECN to mk2
          send_ack(rl1,rl2,sp,mk1,mk2)    // Encode ESQ and send ACK
      }
  }

  /* dack_expire()
   * Routine called when the delayed ACK timer expires.
   * There is no incoming packet to fill mk2,
   *  so the last value from the held sequence has to be used instead
   *  (there will always be a held sequence because the timer is only
   *  armed once the sequence is non-empty).
   */
  dack_expire() {
      if (rl2 > 0) {  // run2 contains a value
          rl2--
          mk2 = sp    // copy it into mk2
      } else {        // run2 is empty, therefore run1 is not
          mk2 = mk1   // copy mk1 into mk2
          rl2 = rl1-- // shift run1 into run2 without mk1
          rl1 = 0
      }               // Last value extraction is complete
```

```
        send_ack(rl1,rl2,sp,mk1,mk2)    // Encode ESQ and send ACK
    }

    /* send_ack()
     * Algorithm to encode the arrival sequence of IP-ECN codepoints
     *  into the ECN sequence (ESQ) field of a TCP ACK, then send it.
     */
    send_ack(rl1,rl2,sp,mk1,mk2) {
        del_timer(dack)         // Remove any pending delayed ACK timer
        /* Marshall the ECN Sequence field (esq) */
        pkt.tcp.esq = lsb(2,sp) & lsb(2,mk1) & lsb(3,rl1) & lsb(3,rl2)
        /* lsb(n,x): pseudocode for the lowest n significant bits of x */
        /* x & y   : pseudocode for concatenate x and y */
        /*
         * Insert code to send ACK here, with mk2 in pkt.tcp.ace
         */
        /* Reset all variables ready for next packet arrival */
        sp = mk1 = DEFAULT
        rl1 = rl2 = 0
    }
```

Appendix B. Alternative Design Choices (To Be Removed Before Publication)

This appendix is informative, not normative. It records alternative designs that the authors chose not to include in the normative specification, but which the IETF might wish to consider for inclusion.

B.1. Supplementary AccECN Field on the SYN/ACK

{ToDo: The tcpm working group is recommended to consider including this in an AccECN RFC from the start. The AccECN protocol defined in the body of this specification currently gives no ECN feedback on the SYN/ACK on the assumption that the SYN is not ECN-capable. If it is required for the protocol to be future-proofed against the possibility that SYNs might one-day be ECN-capable, the following definition of the SupAccECN field for the SYN/ACK would need to be added to Section 3.3.1 and Section 3.3.2. The text below is written as if it is normative, but it is only informative while it is demoted to this appendix.}

B.1.1. Placement of the Supplementary AccECN Field in a SYN/ACK

To include the SupAccECN field on a SYN/ACK, the Data Receiver MUST use the SupAccECN TCP Option with TCP option Kind 0x<KK> (TBA) and set the Length field to 3 [octets], as illustrated in Figure 9. .

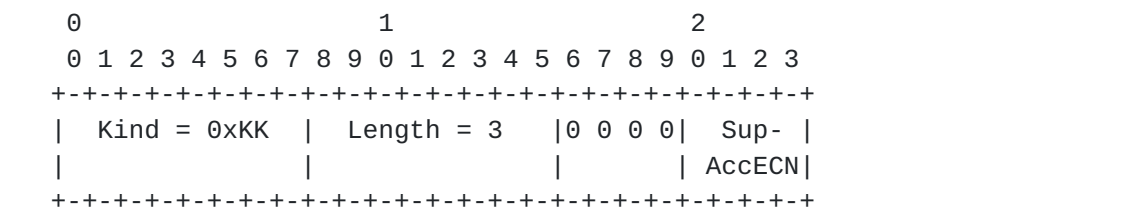


Figure 9: Placement of the SupAccECN field within the SupAccECN TCP Option on a SYN/ACK

If the Data Sender has entered AccECN mode but there is no SupAccECN TCP Option on a SYN/ACK, the Incoming AccECN Protocol Handler MUST take the SupAccECN field to be right-justified within the Non-Urgent field (i.e. the least significant bit of SupAccECN is aligned with the least significant bit of the Non-Urgent Field) as shown in Figure 10. The remaining most significant bits are currently unused (CU).

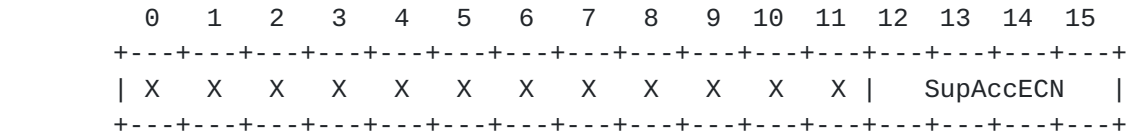


Figure 10: Placement of the SupAccECN field within the Non-Urgent field on a SYN/ACK

B.1.2. Structure of the Supplementary AccECN Field in a SYN/ACK

The size of the SupAccECN field on a SYN/ACK (i.e. a segment with SYN = 1 and ACK = 1) is always 4 bits. Figure 11 defines the sub-fields of the SupAccECN field on a SYN/ACK.

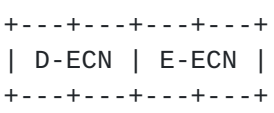


Figure 11: The Supplementary AccECN Field on a SYN/ACK Segment

The sub-fields of SupAccECN on a SYN/ACK segment have the following meanings:

E-ECN: Echo ECN, for the responding host (B) to echo the IP-ECN field that arrives in the SYN. RFC 3168 requires that the ECN field on a SYN must always be Not-ECT (0b00). Therefore initially the E-ECN field is likely to always be 0b00. However, the AccECN wire protocol allows for the possibility that ECN-capable SYNs

might be allowed in future. The responding host (B) MUST echo a copy of the IP-ECN field of the SYN in the E-ECN field of the SYN/ACK.

If the SYN were to arrive carrying a congestion indication, the responding host (B) MUST also increment the relevant counter (r.ci, r.e1 or r.e1) as specified in Section 3.2.1. Then the counters on subsequent feedback will remain consistent even though the SYN/ACK does not have an ACE field to feedback congestion counters (because it is still using the same bits as flags for capability negotiation). The E-ECN field has been defined within a SYN/ACK because the start of a flow is when it is most critical for congestion feedback to be timely. Without the E-ECN field, feedback of any congestion marking on a SYN would get deferred for at least a round trip.

D-ECN: Reserved for a Duplicate ECN field, meaning a duplicate of the ECN field in the IP header of the same packet. This field is not defined in the present specification, but it is reserved for possible use by a companion specification about ECN-fall-back (see Appendix B.3).

Forward Compatibility: In the meantime, the responding host (B) MUST set D-ECN to ECT(0) (0b10), the originating host (A) MUST ignore this field and middleboxes MUST forward this field unaltered whether or not it is 0b10.

B.2. Remove Not-ECT from ECN Sequence (ESQ) Encoding

This alternative encoding would allow the ESQ field to be 1 bit shorter (9 bits instead of 10). The trade-off is that the receiver has to send an ACK immediately whenever a Not-ECT packet arrives. This is because this alternative encoding only caters for one Not-ECT codepoint in the ACE field, and none in the ESQ field.

Once ECN has been negotiated for a connection, the sender ought to rarely send data segments with the Not-ECT codepoint. The only data segments on which RFC 3168 requires the sender to set Not-ECT are retransmissions and window probes. Pure ACKs also have to be sent as Not-ECT, but they are not data segments, so they are not included in the feedback sequence.

If the encoding of the ESQ field has to allow for Not-ECT as well as the three ECN-capable codepoints, it needs space to encode 4 possible spaces and 4 possible marks. This requires 4 bits for 4x4=16 combinations (two 2-bit fields for SP and MK1). If on the other hand Not-ECT is excluded, space for only 3x3=9 combinations is required. This many combinations can only be fitted into 3 bits if they can be

reduced to 8 codepoints by encoding two combinations as one symbol. Two combinations can be encoded as one symbol using the same encoding for sp=mk1=ECT(1) and sp=mk1=CE. This is because either an ECT(1) or CE code in the ACE field can be used to distinguish which is which. However, whenever a run of ECT(1) or of CE ended, the encoding algorithm would have to send two ACKs at once.

Arguments against this alternative design choice:

- Although retransmissions would be expected to be rare in a fully ECN-enabled network, there might be frequent losses and retransmissions during early deployment of ECN, when many bottleneck links might not be ECN-enabled. Then this alternative encoding would reduce the opportunities when a receiver could use delayed ACKs.

- Even if the sender sets Not-ECT on few data segments, incorrectly configured or buggy network equipment exists that clears the IP-ECN field to Not-ECT. With this alternative encoding, connections via such equipment would never be able to use delayed ACKs. The consequential extra ACK load might be considered an incentive for these networks to fix their bugs. However, the endpoints would also suffer the extra ACK load.

- To save 1 bit in the encoding it seems necessary for the algorithm to sometimes have to send two ACKs at once.

B.3. ECN Fall-Back

{ToDo: consider whether the present specification could be enhanced with ECN fall-back on the SYN/ACK to give earlier fall-back than in [I-D.kuehlewind-tcpm-ecn-fallback]. Space for a duplicate of the IP-ECN field on the SYN/ACK has been reserved in the SupAccECN field (Appendix B.1), but the behaviour is still TBA. A duplicate of the IP-ECN field has not been provided on the SYN, because it would be unremarkable if ECN on the SYN was zeroed by security devices, given RFC 3168 prohibited ECT on SYN because it enables DoS attacks. Therefore the IP-ECN field has to be tested on the last ACK of the 3WHS, IMO}

B.4. Remote Delayed ACK Control Proposal

{ToDo: The tcpm working group is recommended to consider including this in an AccECN RFC from the start, because it would be less useful if it was unpredictable whether it had been implemented. The text below is written as if it is normative, but it is only informative while it is demoted to this appendix.} {ToDo: Add a use-case.}

Traditionally, each decision on whether to delay an ACK is taken independently by the Data Receiver. This makes it hard to deploy behaviours where the Data Sender would like the Data Receiver not to delay feedback, perhaps so that it can measure the effect of subtle changes in the timing between packets to more rapidly get up to speed during slow-start without overshoot.

A single bit for a Delayed ACK Control (DAC) flag is defined within the SupAccECN field of segments with SYN=0. Space for this is reserved in Section 3.3.2 and illustrated in Figure 6. For either half-connection, the Data Sender can use the DAC flag to request that the remote Data Receiver turns delayed ACKing on or off:

- DAC = 0 means the sender requests that the receiver turns Delayed ACKing on, using the receiver's choice of delayed ACK factor.

- DAC = 1 means the sender requests that the receiver turns Delayed ACKing off.

For resilience, the Data Sender MUST repeat its currently chosen value of DAC continuously on every packet. The Data Receiver SHOULD start to honour the request on receipt. Therefore, as soon as a segment arrives with DAC=1, a Data Sender SHOULD immediately send any deferred ACKs and no longer withhold ACKs while it continues to receive segments with DAC=1. The DAC flag is meaningful on every packet with SYN=0. The DAC flag is not needed and therefore not present in the SupAccECN field when SYN=1 (Figure 11), because TCP never withholds the SYN/ACK or the final ACK of the 3-way handshake.

A receiver MAY ignore a request from a sender to alter its Delayed ACKing behaviour, e.g. a challenged receiver that cannot send ACKs fast enough need not turn off Delayed ACKs, or a receiver that has not implemented delayed ACKs need not turn them on.

Appendix C. **Open Protocol Design Issues (To Be Removed Before** Publication)

1. A possibility to simplify the protocol would be to remove ordering feedback entirely, but require the receiver to disable delayed ACKs during slow-start (including within a connection after a time-out or idle period) or to provide the DAC flag to allow the sender to ask the receiver to disable delayed ACKs when it needs more accuracy. However, not delaying ACKs may impact server performance. Also a new way to identify middlebox interference in the remaining SupAccECN field (Top-ACE & DAC) would have to be found.

2. The protocol currently gives no ECN feedback on the SYN/ACK on the assumption that the SYN is not ECN-capable. If it is required for the protocol to be future-proofed against the possibility that SYNs might one-day be ECN-capable, the proposal in Appendix B.1 could be adopted. This also provides earlier ECN-fall-back than would otherwise be possible.

3. Section 3.3.1 says an AccECN implementation has to be prepared to read the SupAccECN field from either a TCP option or the Non-Urgent field. If the definition of the SupAccECN field changes between this experimental spec and the standards track spec, the structure of the Non-Urgent field will have to include a version number somehow.

4. The Non-Urgent field might be used for something else in future rather than SupAccECN, despite the attempt to reserve it in this spec. Section 3.3.1 says "If a SupAccECN TCP option is present, the Non-Urgent field MUST be ignored.", which seems to correctly ensure that experimental implementations will not read the altered Non-Urgent field in this case. However, they will incorrectly read the Non-Urgent field if a future AccECN protocol uses a different TCP option.

5. There is possibly a concern that, if the supplementary field is unavailable, the counter selection (Section 3.2.3) always uses the last codepoint in a delayed ACK, which may starve visibility of other counters.

6. Counter Selection Algo #Alt2 Appendix A.2.2 needs to be altered to prevent the E1 counter being continually repeated when no ECT(1) codepoints are arriving at the Data Receiver.

7. A production version of Counter Selection Algo #Alt1 Appendix A.2.1 needs to be developed that handles wrapping of the variables, without losing proportionality.

8. Example algorithms need to be developed that decode the Top-ACE:ACE counters correctly when ACKs are reordered.

9. The definition of the D-ECN field Section 3.3.2 and ECN fall-back more generally Appendix B.3 will need to be resolved before publication.

Appendix D. Changes in This Version (To Be Removed Before Publication)

The difference between any pair of versions can be displayed at <http://datatracker.ietf.org/doc/draft-kuehlewind-tcpm-accurate-ecn/history/>

From 02 to 03:

* Extensively rewritten. No summary of changes has been prepared.

Authors' Addresses

Bob Briscoe
BT
B54/77, Adastral Park
Martlesham Heath
Ipswich IP5 3RE
UK

Phone: +44 1473 645196
EMail: bob.briscoe@bt.com
URI: http://bobbriscoe.net/

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

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

Mirja Kuehlewind
University of Stuttgart
Pfaffenwaldring 47
Stuttgart 70569
Germany

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