

TCPM Working Group  
Internet-Draft  
Obsoletes: 2861 (if approved)  
Intended status: Experimental  
Expires: December 27, 2015

G. Fairhurst  
A. Sathiaselan  
R. Secchi  
University of Aberdeen  
June 25, 2015

Updating TCP to support Rate-Limited Traffic  
draft-ietf-tcpm-newcwv-13

Abstract

This document provides a mechanism to address issues that arise when TCP is used for traffic that exhibits periods where the sending rate is limited by the application rather than the congestion window. It provides an experimental update to TCP that allows a TCP sender to restart quickly following a rate-limited interval. This method is expected to benefit applications that send rate-limited traffic using TCP, while also providing an appropriate response if congestion is experienced.

It also evaluates the Experimental specification of TCP Congestion Window Validation, CWV, defined in RFC 2861, and concludes that RFC 2861 sought to address important issues, but failed to deliver a widely used solution. This document therefore recommends that the status of RFC 2861 is moved from Experimental to Historic, and that it is replaced by the current specification.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 27, 2015.

## Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Implementation of new CWV . . . . .	5
1.2. Standards Status of this Document . . . . .	5
2. Reviewing experience with TCP-CWV . . . . .	5
3. Terminology . . . . .	7
4.1. Initialisation . . . . .	8
4.2. Estimating the validated capacity supported by a path . .	8
4.3. Preserving cwnd during a rate-limited period. . . . .	10
4.4. TCP congestion control during the non-validated phase . .	11
4.4.1. Response to congestion in the non-validated phase . .	12
4.4.2. Sender burst control during the non-validated phase .	13
4.4.3. Adjustment at the end of the Non-Validated Period (NVP) . . . . .	14
4.5. Examples of Implementation . . . . .	15
4.5.1. Implementing the pipeACK measurement . . . . .	15
4.5.2. Measurement of the NVP and pipeACK samples . . . . .	16
4.5.3. Implementing detection of the cwnd-limited condition	16
5. Determining a safe period to preserve cwnd . . . . .	17
6. Security Considerations . . . . .	18
7. IANA Considerations . . . . .	18
8. Acknowledgments . . . . .	18
9. Author Notes . . . . .	18
9.1. Other related work . . . . .	18
10. Revision notes . . . . .	20
11. References . . . . .	24
11.1. Normative References . . . . .	24
11.2. Informative References . . . . .	25
Authors' Addresses . . . . .	26

## 1. Introduction

TCP is used for traffic with a range of application behaviours. The TCP congestion window (cwnd) controls the maximum number of unacknowledged packets/bytes that a TCP flow may have in the network at any time, a value known as the FlightSize [RFC5681]. FlightSize is a measure of the volume of data that is unacknowledged at a specific time. A bulk application will always have data available to transmit. The rate at which it sends is therefore limited by the maximum permitted by the receiver advertised window and the sender congestion window (cwnd). The FlightSize of a bulk flow increases with the cwnd, and tracks the volume of data acknowledged in the last Round Trip Time (RTT).

In contrast, a rate-limited application will experience periods when the sender is either idle or is unable to send at the maximum rate permitted by the cwnd. In this case, the volume of data sent (FlightSize) can change significantly from one RTT to another, and can be much less than the cwnd. Hence, it is possible that the FlightSize could significantly exceed the recently used capacity. The update in this document targets the operation of TCP in such rate-limited cases.

Standard TCP [RFC5681] states that a TCP sender SHOULD set cwnd to no more than the Restart Window (RW) before beginning transmission, if the TCP sender has not sent data in an interval exceeding the retransmission timeout, i.e., when an application becomes idle. [RFC2861] noted that this TCP behaviour was not always observed in current implementations. Experiments [Bis08] confirm this to still be the case.

Congestion Window Validation, CWV, introduced the terminology of "application limited periods". RFC2861 describes any time that an application limits the sending rate, rather than being limited by the transport, as "rate-limited". This update improves support for applications that vary their transmission rate, either with (short) idle periods between transmission or by changing the rate at which the application sends. These applications are characterised by the TCP FlightSize often being less than cwnd. Many Internet applications exhibit this behaviour, including web browsing, http-based adaptive streaming, applications that support query/response type protocols, network file sharing, and live video transmission. Many such applications currently avoid using long-lived (persistent) TCP connections (e.g., [RFC7230] servers typically support persistent HTTP connections, but do not enable this by default). Such applications often instead either use a succession of short TCP transfers or use UDP.

Standard TCP does not impose additional restrictions on the growth of the congestion window when a TCP sender is unable to send at the maximum rate allowed by the cwnd. In this case, the rate-limited sender may grow a cwnd far beyond that corresponding to the current transmit rate, resulting in a value that does not reflect current information about the state of the network path the flow is using. Use of such an invalid cwnd may result in reduced application performance and/or could significantly contribute to network congestion.

[RFC2861] proposed a solution to these issues in an experimental method known as CWV. CWV was intended to help reduce cases where TCP accumulated an invalid (inappropriately large) cwnd. The use and drawbacks of using the CWV algorithm in RFC 2861 with an application are discussed in Section 2.

Section 3 defines relevant terminology.

Section 4 specifies an alternative to CWV that seeks to address the same issues, but does so in a way that is expected to mitigate the impact on an application that varies its sending rate. The updated method applies to the rate-limited conditions (including both application-limited and idle senders).

The goals of this update are:

- o To not change the behaviour of a TCP sender that performs bulk transfers that fully use the cwnd.
- o To provide a method that co-exists with Standard TCP and other flows that use this updated method.
- o To reduce transfer latency for applications that change their rate over short intervals of time.
- o To avoid a TCP sender growing a large "non-validated" cwnd, when it has not recently sent using this cwnd.
- o To remove the incentive for ad-hoc application or network stack methods (such as "padding") solely to maintain a large cwnd for future transmission.
- o To provide an incentive for the use of long-lived connections, rather than a succession of short-lived flows, benefiting both the flows and other flows sharing the network path when actual congestion is encountered.

Section 5 describes the rationale for selecting the safe period to preserve the cwnd.

#### 1.1. Implementation of new CWV

The method specified in Section 4 of this document is a sender-side only change to the the TCP congestion control behaviour of TCP.

The method creates a new protocol state, and requires a sender to determine when the cwnd is validated or non-validated to control the entry and exit from this state Section 4.3. It defines how a TCP sender manages the growth of the cwnd using the set of rules defined in Section 4.

Implementation of this specification requires an implementor to define a method to measure the available capacity using the pipeACK samples. The details of this measurement are implementation-specific. An example is provided in Section 4.5.1, but other methods are permitted. A sender also needs to provide a method to determine when it becomes cwnd-limited. Implementation of this may require consideration of other TCP methods (see Section 4.5.3).

A sender is also recommended to provide a method that controls the maximum burst size, Section 4.4.2. However, implementors are allowed flexibility in how this method is implemented and the choice of an appropriate method is expected to depend on the way in which the sender stack implements other TCP methods (such as TCP Segment Offload, TSO).

#### 1.2. Standards Status of this Document

The document obsoletes the methods described in [RFC2861]. It recommends a set of mechanisms, including the use of pacing during a non-validated period. The updated mechanisms are intended to have a less aggressive congestion impact than would be exhibited by a standard TCP sender.

The specification in this draft is classified as "Experimental" pending experience with deployed implementations of the methods.

#### 2. Reviewing experience with TCP-CWV

[RFC2861] described a simple modification to the TCP congestion control algorithm that decayed the cwnd after the transition to a "sufficiently-long" idle period. This used the slow-start threshold (ssthresh) to save information about the previous value of the congestion window. The approach relaxed the standard TCP behaviour [RFC5681] for an idle session, intended to improve application

performance. CWV also modified the behaviour when a sender transmitted at a rate less than allowed by cwnd.

[RFC2861] proposed two set of responses, one after an "application-limited" and one after an "idle period". Although this distinction was argued, in practice differentiating the two conditions was found problematic in actual networks (e.g., [Bis10]). While this offers predictable performance for long on-off periods ( $>1$  RTT), or slowly varying rate-based traffic, the performance could be unpredictable for variable-rate traffic and depended both upon whether an accurate RTT had been obtained and the pattern of application traffic relative to the measured RTT.

Many applications can and often do vary their transmission over a wide range of rates. Using [RFC2861] such applications often experienced varying performance, which made it hard for application developers to predict the TCP latency even when using a path with stable network characteristics. We argue that an attempt to classify application behaviour as application-limited or idle is problematic and also inappropriate. This document therefore explicitly avoids trying to differentiate these two cases, instead treating all rate-limited traffic uniformly.

[RFC2861] has been implemented in some mainstream operating systems as the default behaviour [Bis08]. Analysis (e.g., [Bis10] [Fail2]) has shown that a TCP sender using CWV is able to use available capacity on a shared path after an idle period. This can benefit variable-rate applications, especially over long delay paths, when compared to the slow-start restart specified by standard TCP. However, CWV would only benefit an application if the idle period were less than several Retransmission Time Out (RTO) intervals [RFC6298], since the behaviour would otherwise be the same as for standard TCP, which resets the cwnd to the TCP Restart Window after this period.

To enable better performance for variable-rate applications with TCP, some operating systems have chosen to support non-standard methods, or applications have resorted to "padding" streams by sending dummy data to maintain their sending rate when they have no data to transmit. Although transmitting redundant data across a network path provides good evidence that the path can sustain data at the offered rate, padding also consumes network capacity and reduces the opportunity for congestion-free statistical multiplexing. For variable-rate flows, the benefits of statistical multiplexing can be significant and it is therefore a goal to find a viable alternative to padding streams.

Experience with [RFC2861] suggests that although the CWV method benefited the network in a rate-limited scenario (reducing the probability of network congestion), the behaviour was too conservative for many common rate-limited applications. This mechanism did not therefore offer the desirable increase in application performance for rate-limited applications and it is unclear whether applications actually use this mechanism in the general Internet.

It is therefore concluded that CWV, as defined in [RFC2861], was often a poor solution for many rate-limited applications. It had the correct motivation, but had the wrong approach to solving this problem.

### 3. Terminology

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 [RFC2119].

The document assumes familiarity with the terminology of TCP congestion control [RFC5681].

The following additional terminology is introduced in this document:

**cwnd-limited:** A TCP flow that has sent the maximum number of segments permitted by the cwnd, where the application utilises the allowed sending rate (see Section 4.5.3).

**pipeACK sample:** A measure of the volume of data acknowledged by the network within an RTT.

**pipeACK variable:** A variable that measures the available capacity using the set of pipeACK samples.

**pipeACK Sampling Period:** The maximum period that a measured pipeACK sample may influence the pipeACK variable.

**Non-validated phase:** The phase where the cwnd reflects a previous measurement of the available path capacity.

**Non-validated period, NVP:** The maximum period for which cwnd is preserved in the non-validated phase.

**Rate-limited:** A TCP flow that does not consume more than one half of cwnd, and hence operates in the non-validated phase. This includes periods when an application is either idle or chooses to send at a rate less than the maximum permitted by the cwnd.

Validated phase: The phase where the cwnd reflects a current estimate of the available path capacity.

#### 4. A New Congestion Window Validation method

This section proposes an update to the TCP congestion control behaviour during a rate-limited interval. This new method intentionally does not differentiate between times when the sender has become idle or chooses to send at a rate less than the maximum allowed by the cwnd.

The period where actual usage is less than allowed by cwnd, is named the non-validated phase. The update allows an application in the non-validated phase to resume transmission at a previous rate without incurring the delay of slow-start. However, if the TCP sender experiences congestion using the preserved cwnd, it is required to immediately reset the cwnd to an appropriate value specified by the method. If a sender does not take advantage of the preserved cwnd within the Non-validated period, NVP, the value of cwnd is reduced, ensuring the value better reflects the capacity that was recently actually used.

It is expected that this update will satisfy the requirements of many rate-limited applications and at the same time provide an appropriate method for use in the Internet. New-CWV reduces this incentive for an application to send "padding" data simply to keep transport congestion state.

The method is specified in following subsections and is expected to encourage applications and TCP stacks to use standards-based congestion control methods. It may also encourage the use of long-lived connections where this offers benefit (such as persistent http).

##### 4.1. Initialisation

A sender starts a TCP connection in the validated phase and initialises the pipeACK variable to the "undefined" value. This value inhibits use of the value in cwnd calculations.

##### 4.2. Estimating the validated capacity supported by a path

[RFC6675] defines a variable, FlightSize, that indicates the instantaneous amount of data that has been sent, but not cumulatively acknowledged. In this method a new variable "pipeACK" is introduced to measure the acknowledged size of the network pipe. This is used to determine if the sender has validated the cwnd. pipeACK differs



from FlightSize in that it is evaluated over a window of acknowledged data, rather than reflecting the amount of data outstanding.

A sender determines a pipeACK sample by measuring the volume of data that was acknowledged by the network over the period of a measured Round Trip Time (RTT). Using the variables defined in [RFC6675], a value could be measured by caching the value of HighACK and after one RTT measuring the difference between the cached HighACK value and the current HighACK value. A sender MAY count TCP DupACKs that acknowledge new data when collecting the pipeACK sample. Other equivalent methods may be used.

A sender is not required to continuously update the pipeACK variable after each received ACK, but SHOULD perform a pipeACK sample at least once per RTT when it has sent unacknowledged segments.

The pipeACK variable MAY consider multiple pipeACK samples over the pipeACK Sampling Period. The value of the pipeACK variable MUST NOT exceed the maximum (highest value) within the sampling period. This specification defines the pipeACK Sampling Period as  $\text{Max}(3 \cdot \text{RTT}, 1 \text{ second})$ . This period enables a sender to compensate for large fluctuations in the sending rate, where there may be pauses in transmission, and allows the pipeACK variable to reflect the largest recently measured pipeACK sample.

When no measurements are available (e.g., a sender that has just started transmission or immediately after loss recovery), the pipeACK variable is set to the "undefined value". This value is used to inhibit entering the non-validated phase until the first new measurement of a pipeACK sample. (Section 4.5 provides examples of implementation.)

The pipeACK variable MUST NOT be updated during TCP Fast Recovery. That is, the sender stops collecting pipeACK samples during loss recovery. The method RECOMMENDS enabling the TCP SACK option [RFC2018] and RECOMMENDS the method defined in [RFC6675] to recover missing segments. This allows the sender to more accurately determine the number of missing bytes during the loss recovery phase, and using this method will result in a more appropriate cwnd following loss.

NOTE: The use of pipeACK rather than FlightSize can change the behaviour of a TCP when a sender does not always have data available to send. One example arises when there is a pause in transmission after sending a sequence of many packets, and the sender experiences loss at or near the end of its transmission sequence. In this case, the TCP flow may have used a significant amount of capacity just prior to the loss (which would be reflected in the volume of data

acknowledged, recorded in the pipeACK variable), but at the actual time of loss the number of unacknowledged packets in flight (at the end of the sequence) may be small, i.e., there is a small FlightSize. After loss recovery, the sender resets its congestion control state.

[Fail2] explored the benefits of different responses to congestion for application-limited streams. If the response is based only on the Loss FlightSize, the sender would assign a small cwnd and ssthresh, based only on the volume of data sent after the loss. When the sender next starts to transmit it can incur many RTTs of delay in slow start before it reacquires its previous rate. When the pipeACK value is also used to calculate the cwnd and ssthresh (as specified in this update in Section 4.4.1), the sender can use a value that also reflects the recently used capacity before the loss. This prevents a variable-rate application from being unduly penalised. When the sender resumes, it starts at one half its previous rate, similar to the behaviour of a bulk TCP flow [Hos15]. To ensure an appropriate reaction to on-going congestion, this method requires that the pipeACK variable is reset after it is used in this way.

#### 4.3. Preserving cwnd during a rate-limited period.

The updated method creates a new TCP sender phase that captures whether the cwnd reflects a validated or non-validated value. The phases are defined as:

- o Validated phase:  $\text{pipeACK} \geq (1/2) * \text{cwnd}$ , or pipeACK is undefined (i.e., at the start or directly after loss recovery). This is the normal phase, where cwnd is expected to be an approximate indication of the capacity currently available along the network path, and the standard methods are used to increase cwnd (currently [RFC5681]).
- o Non-validated phase:  $\text{pipeACK} < (1/2) * \text{cwnd}$ . This is the phase where the cwnd has a value based on a previous measurement of the available capacity, and the usage of this capacity has not been validated in the pipeACK Sampling Period. That is, when it is not known whether the cwnd reflects the currently available capacity along the network path. The mechanisms to be used in this phase seek to determine a safe value for cwnd and an appropriate reaction to congestion.

Note: A threshold is needed to determine whether a sender is in the validated or non-validated phase. A standard TCP sender in slow-start is permitted to double its FlightSize from one RTT to the next. This motivated the choice of a threshold value of 1/2. This threshold ensures a sender does not further increase the cwnd as long as the FlightSize is less than  $(1/2 * \text{cwnd})$ . Furthermore, a sender

with a FlightSize less than  $(1/2 * \text{cwnd})$  may in the next RTT be permitted by the cwnd to send at a rate that more than doubles the FlightSize, and hence this case needs to be regarded as non-validated and a sender therefore needs to employ additional mechanisms while in this phase.

#### 4.4. TCP congestion control during the non-validated phase

A TCP sender implementing this specification MUST enter the non-validated phase when the pipeACK is less than  $(1/2) * \text{cwnd}$ . (The note at the end of section 4.4.1 describes why  $\text{pipeACK} \leq (1/2) * \text{cwnd}$  is expected to be a safe value.)

A TCP sender that enters the non-validated phase preserves the cwnd (i.e., the cwnd only increases after a sender fully uses the cwnd in this phase, otherwise the cwnd neither grows nor reduces). The phase is concluded when the sender transmits sufficient data so that  $\text{pipeACK} > (1/2) * \text{cwnd}$  (i.e., the sender is no longer rate-limited), or when the sender receives an indication of congestion.

After a fixed period of time (the non-validated period, NVP), the sender adjusts the cwnd (Section 4.4.3). The NVP SHOULD NOT exceed 5 minutes. Section 5 discusses the rationale for choosing a safe value for this period.

The behaviour in the non-validated phase is specified as:

- o A sender determines whether to increase the cwnd based upon whether it is cwnd-limited (see Section 4.5.3):
  - \* A sender that is cwnd-limited MAY use the standard TCP method to increase cwnd (i.e., a TCP sender that fully utilises the cwnd is permitted to increase cwnd each received ACK using standard methods).
  - \* A sender that is not cwnd-limited MUST NOT increase the cwnd when ACK packets are received in this phase (i.e., needs to avoid growing the cwnd when it has not recently sent using the current size of cwnd).
- o If the sender receives an indication of congestion while in the non-validated phase (i.e., detects loss), the sender MUST exit the non-validated phase (reducing the cwnd as defined in Section 4.4.1).
- o If the Retransmission Time Out (RTO) expires while in the non-validated phase, the sender MUST exit the non-validated phase. It then resumes using the standard TCP RTO mechanism [RFC5681].

- o A sender with a pipeACK variable greater than  $(1/2)*cwnd$  SHOULD enter the validated phase. (A rate-limited sender will not normally be impacted by whether it is in a validated or non-validated phase, since it will normally not increase FlightSize to use the entire cwnd. However, a change to the validated phase will release the sender from constraints on the growth of cwnd, and result in using the standard congestion response.)

The cwnd-limited behaviour may be triggered during a transient condition that occurs when a sender is in the non-validated phase and receives an ACK that acknowledges received data, the cwnd was fully utilised, and more data is awaiting transmission than may be sent with the current cwnd. The sender MAY then use the standard method to increase the cwnd. (Note, if the sender succeeds in sending these new segments, the updated cwnd and pipeACK variables will eventually result in a transition to the validated phase.)

#### 4.4.1. Response to congestion in the non-validated phase

Reception of congestion feedback while in the non-validated phase is interpreted as an indication that it was inappropriate for the sender to use the preserved cwnd. The sender is therefore required to quickly reduce the rate to avoid further congestion. Since the cwnd does not have a validated value, a new cwnd value needs to be selected based on the utilised rate.

A sender that detects a packet-drop MUST record the current FlightSize in the variable LossFlightSize and MUST calculate a safe cwnd for loss recovery using the method below:

$$cwnd = (\text{Max}(\text{pipeACK}, \text{LossFlightSize})) / 2.$$

The pipeACK value is not updated during loss recovery (see Section 4.2). If there is a valid pipeACK value, the new cwnd is adjusted to reflect that a non-validated cwnd may be larger than the actual FlightSize, or recently used FlightSize (recorded in pipeACK). The updated cwnd therefore prevents overshoot by a sender significantly increasing its transmission rate during the recovery period.

At the end of the recovery phase, the TCP sender MUST reset the cwnd using the method below:

$$cwnd = (\text{Max}(\text{pipeACK}, \text{LossFlightSize}) - R) / 2.$$

Where R is the volume of data that was successfully retransmitted during the recovery phase. This corresponds to segments

retransmitted and considered lost by the pipe estimation algorithm at the end of recovery. It does not include the additional cost of multiple retransmission of the same data. The loss of segments indicates that the path capacity was exceeded by at least  $R$ , and hence the calculated  $cwnd$  is reduced by at least  $R$  before the window is halved.

The calculated  $cwnd$  value MUST NOT be reduced below 1 TCP Maximum Segment Size (MSS).

After completing the loss recovery phase, the sender MUST re-initialise the `pipeACK` variable to the "undefined" value. This ensures that standard TCP methods are used immediately after completing loss recovery until a new `pipeACK` value can be determined.

The `ssthresh` is adjusted using the standard TCP method (Step 6 in Section 3.2 of RFC 5681 assigns the `ssthresh` a value equal to  $cwnd$  at the end of the loss recovery).

Note: The adjustment by reducing  $cwnd$  by the volume of data not sent ( $R$ ) follows the method proposed for Jump Start [Liu07]. The inclusion of the term  $R$  makes the adjustment more conservative than standard TCP. This is required, since a sender in the non-validated state may increase the rate more than a standard TCP would have done relative to what was sent in the last RTT (i.e., more than doubled the number of segments in flight relative to what it sent in the last RTT). The additional reduction after congestion is beneficial when the `LossFlightSize` has significantly overshoot the available path capacity incurring significant loss (e.g., following a change of path characteristics or when additional traffic has taken a larger share of the network bottleneck during a period when the sender transmits less).

Note: The `pipeACK` value is only valid during a non-validated phase, and therefore this does not exceed  $cwnd/2$ . If `LossFlightSize` and  $R$  were small, then this can result in the final  $cwnd$  after loss recovery being at most one quarter of the  $cwnd$  on detection of congestion. This reduction is conservative, and `pipeACK` is then reset to undefined, hence  $cwnd$  updates after a congestion event do not depend upon the `pipeACK` history before congestion was detected.

#### 4.4.2. Sender burst control during the non-validated phase

TCP congestion control allows a sender to accumulate a  $cwnd$  that would allow it to send a burst of segments with a total size up to the difference between the `FlightsSize` and  $cwnd$ . Such bursts can impact other flows that share a network bottleneck and/or may induce congestion when buffering is limited.

Various methods have been proposed to control the sender burstiness [Hug01], [All05]. For example, TCP can limit the number of new segments it sends per received ACK. This is effective when a flow of ACKs is received, but can not be used to control a sender that has not send appreciable data in the previous RTT [All05].

This document recommends using a method to avoid line-rate bursts after an idle or rate-limited interval when there is less reliable information about the capacity of the network path: A TCP sender in the non-validated phase SHOULD control the maximum burst size, e.g., using a rate-based pacing algorithm in which a sender paces out the cwnd over its estimate of the RTT, or some other method, to prevent many segments being transmitted contiguously at line-rate. The most appropriate method(s) to implement pacing depend on the design of the TCP/IP stack, speed of interface and whether hardware support (such as TCP Segment Offload, TSO) is used. The present document does not recommend any specific method.

#### 4.4.3. Adjustment at the end of the Non-Validated Period (NVP)

An application that remains in the non-validated phase for a period greater than the NVP is required to adjust its congestion control state. If the sender exits the non-validated phase after this period, it MUST update the ssthresh:

$$\text{ssthresh} = \max(\text{ssthresh}, 3 * \text{cwnd} / 4).$$

(This adjustment of ssthresh ensures that the sender records that it has safely sustained the present rate. The change is beneficial to rate-limited flows that encounter occasional congestion, and could otherwise suffer an unwanted additional delay in recovering the sending rate.)

The sender MUST then update cwnd to be not greater than:

$$\text{cwnd} = \max((1/2) * \text{cwnd}, \text{IW}).$$

Where IW is the appropriate TCP initial window, used by the TCP sender (e.g., [RFC5681]).

Note: These cwnd and ssthresh adjustments cause the sender to enter slow-start (since ssthresh > cwnd). This adjustment ensures that the sender responds conservatively after remaining in the non-validated phase for more than the non-validated period. In this case, it reduces the cwnd by a factor of two from the preserved value. This adjustment is helpful when flows accumulate but do not use a large cwnd, and seeks to mitigate the impact when these flows later resume

transmission. This could for instance mitigate the impact if multiple high-rate application flows were to become idle over an extended period of time and then were simultaneously awakened by an external event.

#### 4.5. Examples of Implementation

This section provides informative examples of implementation methods. Implementations may choose to use other methods that comply with the normative requirements.

##### 4.5.1. Implementing the pipeACK measurement

A pipeACK sample may be measured once each RTT. This reduces the sender processing burden for calculating after each acknowledgement and also reduces storage requirements at the sender.

Since application behaviour can be bursty using CWV, it may be desirable to implement a maximum filter to accumulate the measured values so that the pipeACK variable records the largest pipeACK sample within the pipeACK Sampling Period. One simple way to implement this is to divide the pipeACK Sampling Period into several (e.g., 5) equal length measurement periods. The sender then records the start time for each measurement period and the highest measured pipeACK sample. At the end of the measurement period, any measurement(s) that are older than the pipeACK Sampling Period are discarded. The pipeACK variable is then assigned the largest of the set of the highest measured values.

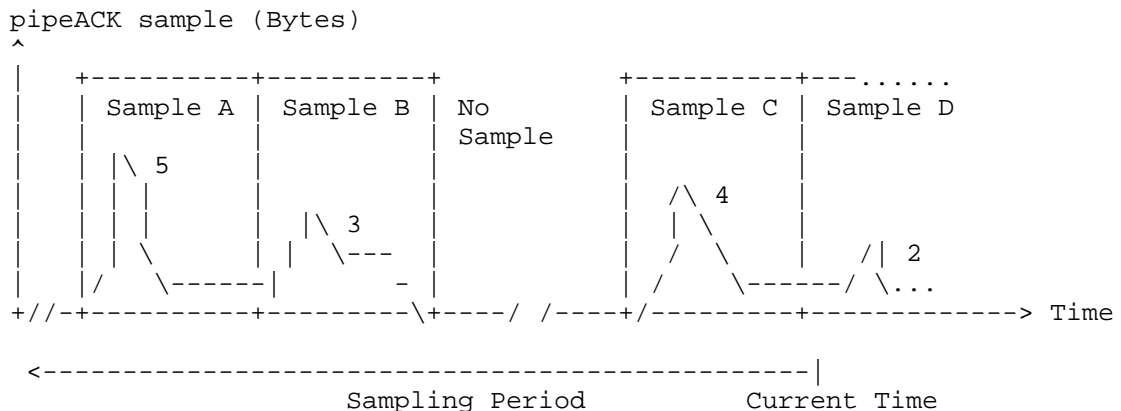


Figure 1: Example of measuring pipeACK samples

Figure 1 shows an example of how measurement samples may be collected. At the time represented by the figure new samples are being accumulated into sample D. Three previous samples also fall within the pipeACK Sampling Period: A, B, and C. There was also a period of inactivity between samples B and C during which no measurements were taken (because no new data segments were acknowledged). The current value of the pipeACK variable will be 5, the maximum across all samples. During this period, the pipeACK samples may be regarded as zero, and hence do not contribute to the calculated pipeACK value.

After one further measurement period, Sample A will be discarded, since it then is older than the pipeACK Sampling Period and the pipeACK variable will be recalculated, Its value will be the larger of Sample C or the final value accumulated in Sample D.

#### 4.5.2. Measurement of the NVP and pipeACK samples

The mechanism requires a number of measurements of time. These measurements could be implemented using protocol timers, but do not necessarily require a new timer to be implemented. Avoiding the use of dedicated timers can save operating system resources, especially when there may be large numbers of TCP flows.

The NVP could be measured by recording a timestamp when the sender enters the non-validated phase. Each time a sender transmits a new segment, this timestamp can be used to determine if the NVP has expired. If the measured period exceeds the NVP, the sender can then take into account how many units of the NVP have passed and make one reduction (defined in Section 4.4.3) for each NVP.

Similarly, the time measurements for collecting pipeACK samples and determining the Sampling Period could be derived by using a timestamp to record when each sample was measured, and to use this to calculate how much time has passed when each new ACK is received.

#### 4.5.3. Implementing detection of the cwnd-limited condition

A sender needs to implement a method that detects the cwnd-limited condition (see Section 4.4). This detects a condition where a sender in the non-validated phase receives an ACK, but the size of cwnd prevents sending more new data.

In simple terms, this condition is true only when the FlightSize of a TCP sender is equal to or larger than the current cwnd. However, an implementation also needs to consider constraints on the way in which the cwnd variable can be used, for instance implementations need to support other TCP methods such as the Nagle Algorithm and TCP Segment



Offload (TSO) that also use cwnd to control transmission. These other methods can result in a sender becoming cwnd-limited when the cwnd is nearly, rather than completely, equal to the FlightSize.

## 5. Determining a safe period to preserve cwnd

This section documents the rationale for selecting the maximum period that cwnd may be preserved, known as the NVP.

Limiting the period that cwnd may be preserved avoids undesirable side effects that would result if the cwnd were to be kept unnecessarily high for an arbitrary long period, which was a part of the problem that CWV originally attempted to address. The period a sender may safely preserve the cwnd, is a function of the period that a network path is expected to sustain the capacity reflected by cwnd. There is no ideal choice for this time.

A period of five minutes was chosen for this NVP. This is a compromise that was larger than the idle intervals of common applications, but not sufficiently larger than the period for which the capacity of an Internet path may commonly be regarded as stable. The capacity of wired networks is usually relatively stable for periods of several minutes and that load stability increases with the capacity. This suggests that cwnd may be preserved for at least a few minutes.

There are cases where the TCP throughput exhibits significant variability over a time less than five minutes. Examples could include wireless topologies, where TCP rate variations may fluctuate on the order of a few seconds as a consequence of medium access protocol instabilities. Mobility changes may also impact TCP performance over short time scales. Senders that observe such rapid changes in the path characteristic may also experience increased congestion with the new method, however such variation would likely also impact TCP's behaviour when supporting interactive and bulk applications.

Routing algorithms may change the the network path that is used by a transport. Although a change of path can in turn disrupt the RTT measurement and may result in a change of the capacity available to a TCP connection, we assume these path changes do not usually occur frequently (compared to a time frame of a few minutes).

The value of five minutes is therefore expected to be sufficient for most current applications. Simulation studies (e.g., [Bis11]) also suggest that for many practical applications, the performance using this value will not be significantly different to that observed using a non-standard method that does not reset the cwnd after idle.

Finally, other TCP sender mechanisms have used a 5 minute timer, and there could be simplifications in some implementations by reusing the same interval. TCP defines a default user timeout of 5 minutes [RFC0793] i.e., how long transmitted data may remain unacknowledged before a connection is forcefully closed.

## 6. Security Considerations

General security considerations concerning TCP congestion control are discussed in [RFC5681]. This document describes an algorithm that updates one aspect of the congestion control procedures, and so the considerations described in RFC 5681 also apply to this algorithm.

## 7. IANA Considerations

There are no IANA considerations.

## 8. Acknowledgments

This document was produced by the TCP Maintenance and Minor Extensions (tcpm) working group.

The authors acknowledge the contributions of Dr I Biswas, Dr Ziaul Hossain in supporting the evaluation of CWV and for their help in developing the mechanisms proposed in this draft. We also acknowledge comments received from the Internet Congestion Control Research Group, in particular Yuchung Cheng, Mirja Kuehlewind, Joe Touch, and Mark Allman. This work was part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700).

## 9. Author Notes

RFC-Editor note: please remove this section prior to publication.

### 9.1. Other related work

RFC-Editor note: please remove this section prior to publication.

There are several issues to be discussed more widely:

- o There are potential interactions with the Experimental update in RFC 6928 that raises the TCP initial Window to ten segments, do these cases need to be elaborated?

This relates to the Experimental specification for increasing the TCP IW defined in RFC 6928.

The two methods have different functions and different response to loss/congestion.

RFC 6928 proposes an experimental update to TCP that would increase the IW to ten segments. This would allow faster opening of the cwnd, and also a large (same size) restart window. This approach is based on the assumption that many forward paths can sustain bursts of up to ten segments without (appreciable) loss. Such a significant increase in cwnd must be matched with an equally large reduction of cwnd if loss/congestion is detected, and such a congestion indication is likely to require future use of IW=10 to be disabled for this path for some time. This guards against the unwanted behaviour of a series of short flows continuously flooding a network path without network congestion feedback.

In contrast, this document proposes an update with a rationale that relies on recent previous path history to select an appropriate cwnd after restart.

The behaviour differs in three ways:

- 1) For applications that send little initially, new-cwv may constrain more than RFC 6928, but would not require the connection to reset any path information when a restart incurred loss. In contrast, new-cwv would allow the TCP connection to preserve the cached cwnd, any loss, would impact cwnd, but not impact other flows.
- 2) For applications that utilise more capacity than provided by a cwnd of 10 segments, this method would permit a larger restart window compared to a restart using the method in RFC 6928. This is justified by the recent path history.
- 3) new-CWV is intended to also be used for rate-limited applications, where the application sends, but does not seek to fully utilise the cwnd. In this case, new-cwv constrains the cwnd to that justified by the recent path history. The performance trade-offs are hence different, and it would be possible to enable new-cwv when also using the method in RFC 6928, and yield benefits.

o There is potential overlap with the Laminar proposal (draft-mathis-tcpm-tcp-laminar)

The current draft was intended as a standards-track update to TCP, rather than a new transport variant. At least, it would be good to understand how the two interact and whether there is a possibility of a single method.

- o There is potential performance loss in loss of a short burst (off list with M Allman)

A sender can transmit several segments then become idle. If the first set of segments are all Acknowledged, the ssthresh collapses to a small value (no new data is sent by the idle sender). Loss of the later data results in congestion (e.g., maybe a RED drop or some other cause, rather than the maximum rate of this flow). When the sender performs loss recovery it may have an appreciable pipeACK and cwnd, but a very low FlightSize - the Standard algorithm therefore results in an unusually low cwnd ( $(1/2) * \text{FlightSize}$ ).

A constant rate flow would have maintained a FlightSize appropriate to pipeACK (cwnd, if it is a bulk flow).

This could be fixed by adding a new state variable? It could also be argued this is a corner case (e.g., loss of only the last segments would have resulted in RTT), the impact could be significant.

- o There is potential interaction with TCP Control Block Sharing (M Welzl)

An application that is non-validated can accumulate a cwnd that is larger than the actual capacity. Is this a fair value to use in TCB sharing?

We propose that TCB sharing should use the pipeACK in place of cwnd when a TCP sender is in the Non-validated phase. This value better reflects the capacity that the flow has utilised in the network path.

## 10. Revision notes

RFC-Editor note: please remove this section prior to publication.

Draft 03 was submitted to ICCRG to receive comments and feedback.

Draft 04 contained the first set of clarifications after feedback:

- o Changed name to application limited and used the term rate-limited in all places.
- o Added justification and many minor changes suggested on the list.
- o Added text to tie-in with more accurate ECN marking.
- o Added ref to Hug01

Draft 05 contained various updates:

- o New text to redefine how to measure the acknowledged pipe, differentiating this from the FlightSize, and hence avoiding previous issues with infrequent large bursts of data not being validated. A key point new feature is that pipeACK only triggers leaving the NVP after the size of the pipe has been acknowledged. This removed the need for hysteresis.
- o Reduction values were changed to 1/2, following analysis of suggestions from ICCRG. This also sets the "target" cwnd as twice the used rate for non-validated case.
- o Introduced a symbolic name (NVP) to denote the 5 minute period.

Draft 06 contained various updates:

- o Required reset of pipeACK after congestion.
- o Added comment on the effect of congestion after a short burst (M. Allman).
- o Correction of minor Typos.

WG draft 00 contained various updates:

- o Updated initialisation of pipeACK to maximum value.
- o Added note on intended status still to be determined.

WG draft 01 contained:

- o Added corrections from Richard Scheffenegger.
- o Raffaello Secchi added to the mechanism, based on implementation experience.

- o Removed that the requirement for the method to use TCP SACK option
- o Although it may be desirable to use SACK, this is not essential to the algorithm.
- o Added the notion of the sampling period to accommodate large rate variations and ensure that the method is stable. This algorithm to be validated through implementation.

WG draft 02 contained:

- o Clarified language around pipeACK variable and pipeACK sample - Feedback from Aris Angelogiannopoulos.

WG draft 03 contained:

- o Editorial corrections - Feedback from Anna Brunstrom.
- o An adjustment to the procedure at the start and end of Reoloss recovery to align the two equations.
- o Further clarification of the "undefined" value of the pipeACK variable.

WG draft 04 contained:

- o Editorial corrections.
- o Introduced the "cwnd-limited" term.
- o An adjustment to the procedure at the start of a cwnd-limited phase - the new text is intended to ensure that new-cwv is not unnecessarily more conservative than standard TCP when the flow is cwnd-limited. This resolves two issues: first it prevents pathologies in which pipeACK increases slowly and erratically. It also ensures that performance of bulk applications is not significantly impacted when using the method.
- o Clearly identifies that pacing (or equivalent) is requiring during the NVP to control burstiness. New section added.

WG draft 05 contained:

- o Clarification to first two bullets in Section 4.4 describing cwnd-limited, to explain these are really alternates to the same case.
- o Section giving implementation examples was restructured to clarify there are two methods described.

- o Cross References to sections updated - thanks to comments from Martin Winbjoerk and Tim Wicinski.

WG draft 06 contained:

- o The section giving implementation examples was restructured to clarify there are two methods described.
- o Justification of design decisions.
- o Re-organised text to improve clarity of argument.

WG draft 07 contained:

- o Updated publication date.
- o Text on noting that cwnd shouldn't ever be made negative.
- o Updated text on ECN to clarify the process where R is a reduction based on ECN marks.

WG draft 08 contained:

- o Removed description of how to use Accurate ECN feedback. It is not clear that this document should specify a usage of a mechanism that has not been fully defined. Accurate ECN may lead to different congestion responses and these will need to be defined in the CC specifications for using Accurate ECN.

WG draft 09 contained:

- o Removed update to RFC 5681 - the status of the present document is Experimental, and hence this document does not update RFC 5681.

WG draft 10 contained edits following WGLC:

- o Section 1.1 Implementation of new CWV: New section added to introduce the places where there are implementation flexibility.
- o Section 4.4: Clarified that the MUST is to satisfy the goal to avoid a TCP sender growing a large "non-validated" cwnd, when it has not recently sent using the current size of cwnd, and fixed format of bullet 2 in 4.4.
- o Section 4.5.2: rewritten section text.

WG draft 11 contained edits following IETF LC:

- o Updated text in section 1.1.
- o Updated text in response to AD, Gen-ART, & Sec reviews.
- o LC call comments from Mirja Kuehlewind

WG draft 12 contained edits following IETF LC (Mirja Kuehlewind):

- o Additional text (based on text in annexe notes) to clarify use of pipeACK rather than FlightSize.
- o Corrected text on undefined pipeACK - to be consistent.
- o Added text on standard TCP method (reference to RFC 5681).
- o Separated text on implementation experience of "timers" into a new implementation subsection (4.5.2), to avoid this common implementation method being overlooked.

WG draft 13 contained edits following IESG Review:

- o Jari/Gen-ART (note: MSS was defined)
- o Kathleen Moriarty (SecDir)
- o Ben Campbell
- o Barry Leiba (note: reference added to section 4, rather than new wording to requirement).

## 11. References

### 11.1. Normative References

- [RFC0793] Postel, J., "Transmission Control Protocol", September 1981.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2861] Handley, M., Padhye, J., and S. Floyd, "TCP Congestion Window Validation", RFC 2861, June 2000.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", September 2009.



- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", June 2011.
- [RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, August 2012.

## 11.2. Informative References

- [All05] Allman, M. and E. Blanton, "Notes on burst mitigation for transport protocols", March 2005.
- [Bis08] Biswas, I. and G. Fairhurst, "A Practical Evaluation of Congestion Window Validation Behaviour, 9th Annual Postgraduate Symposium in the Convergence of Telecommunications, Networking and Broadcasting (PGNet), Liverpool, UK", June 2008.
- [Bis10] Biswas, I., Sathiaselalan, A., Secchi, R., and G. Fairhurst, "Analysing TCP for Bursty Traffic, Int'l J. of Communications, Network and System Sciences, 7(3)", June 2010.
- [Bis11] Biswas, I., "PhD Thesis, Internet congestion control for variable rate TCP traffic, School of Engineering, University of Aberdeen", June 2011.
- [Fair12] Sathiaselalan, A., Secchi, R., Fairhurst, G., and I. Biswas, "Enhancing TCP Performance to support Variable-Rate Traffic, 2nd Capacity Sharing Workshop, ACM CoNEXT, Nice, France, 10th December 2012.", June 2008.
- [Hos15] Hossain, Z., "PhD Thesis, A Study of Mechanisms to Support Variable-rate Internet Applications over a Multi-service Satellite Platform, School of Engineering, University of Aberdeen", January 2015.
- [Hug01] Hughes, A., Touch, J., and J. Heidemann, "Issues in TCP Slow-Start Restart After Idle (Work-in-Progress)", December 2001.
- [Liu07] Liu, D., Allman, M., Jin, S., and L. Wang, "Congestion Control without a Startup Phase, 5th International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet), Los Angeles, California, USA", February 2007.

[RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.

Authors' Addresses

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

Email: [gorry@erg.abdn.ac.uk](mailto:gorry@erg.abdn.ac.uk)  
URI: <http://www.erg.abdn.ac.uk>

Arjuna Sathiaselalan  
University of Aberdeen  
School of Engineering  
Fraser Noble Building  
Aberdeen, Scotland AB24 3UE  
UK

Email: [arjuna@erg.abdn.ac.uk](mailto:arjuna@erg.abdn.ac.uk)  
URI: <http://www.erg.abdn.ac.uk>

Raffaello Secchi  
University of Aberdeen  
School of Engineering  
Fraser Noble Building  
Aberdeen, Scotland AB24 3UE  
UK

Email: [raffaello@erg.abdn.ac.uk](mailto:raffaello@erg.abdn.ac.uk)  
URI: <http://www.erg.abdn.ac.uk>

TCP Maintenance and Minor Extensions (tcpm)  
Internet-Draft  
Intended status: Experimental  
Expires: May 8, 2016

P. Hurtig  
A. Brunstrom  
Karlstad University  
A. Petlund  
Simula Research Laboratory AS  
M. Welzl  
University of Oslo  
November 5, 2015

TCP and Sctp RTO Restart  
draft-ietf-tcpm-rtorestart-10

Abstract

This document describes a modified sender-side algorithm for managing the TCP and Sctp retransmission timers that provides faster loss recovery when there is a small amount of outstanding data for a connection. The modification, RTO Restart (RTOR), allows the transport to restart its retransmission timer using a smaller timeout duration, so that the effective RTO becomes more aggressive in situations where fast retransmit cannot be used. This enables faster loss detection and recovery for connections that are short-lived or application-limited.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 8, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## 1. Introduction

TCP and SCTP use two almost identical mechanisms to detect and recover from data loss, specified in [RFC6298][RFC5681] (for TCP) and [RFC4960] (for SCTP). First, if transmitted data is not acknowledged within a certain amount of time, a retransmission timeout (RTO) occurs, and the data is retransmitted. While the RTO is based on measured round-trip times (RTTs) between the sender and receiver, it also has a conservative lower bound of 1 second to ensure that delayed data are not mistaken as lost. Second, when a sender receives duplicate acknowledgments, or similar information via selective acknowledgments, the fast retransmit algorithm suspects data loss and can trigger a retransmission. Duplicate (and selective) acknowledgments are generated by a receiver when data arrives out-of-order. As both data loss and data reordering cause out-of-order arrival, fast retransmit waits for three out-of-order notifications before considering the corresponding data as lost. In some situations, however, the amount of outstanding data is not enough to trigger three such acknowledgments, and the sender must rely on lengthy RTOs for loss recovery.

The amount of outstanding data can be small for several reasons:

- (1) The connection is limited by the congestion control when the path has a low total capacity (bandwidth-delay product) or the connection's share of the capacity is small. It is also limited by the congestion control in the first few RTTs of a connection or after an RTO when the available capacity is probed using slow-start.
- (2) The connection is limited by the receiver's available buffer space.
- (3) The connection is limited by the application if the available capacity of the path is not fully utilized (e.g. interactive applications), or at the end of a transfer.

While the reasons listed above are valid for any flow, the third reason is most common for applications that transmit short flows, or use a bursty transmission pattern. A typical example of applications that produce short flows are web-based applications. [RJ10] shows that 70% of all web objects, found at the top 500 sites, are too small for fast retransmit to work. [FDT13] shows that about 77% of all retransmissions sent by a major web service are sent after RTO expiry. Applications with bursty transmission patterns often send data in response to actions, or as a reaction to real life events. Typical examples of such applications are stock trading systems, remote computer operations, online games, and web-based applications using persistent connections. What is special about this class of applications is that they often are time-dependant, and extra latency can reduce the application service level [P09].

The RTO Restart (RTOR) mechanism described in this document makes the effective RTO slightly more aggressive when the amount of outstanding data is too small for fast retransmit to work, in an attempt to enable faster loss recovery while being robust to reordering. While RTOR still conforms to the requirement for when a segment can be retransmitted, specified in [RFC6298] (for TCP) and [RFC4960] (for SCTP) it could increase the risk of spurious timeouts. To determine whether this modification is safe to deploy and enable by default, further experimentation is required. Section 5 discusses experiments still needed, including evaluations in environments where the risk of spurious retransmissions are increased e.g. mobile networks with highly varying RTTs.

The remainder of this document describes RTOR and its implementation for TCP only, to make the document easier to read. However, the RTOR algorithm described in Section 4 is applicable also for SCTP. Furthermore, Section 7 details the SCTP socket API needed to control RTOR.

## 2. Terminology

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].

This document introduces the following variables:

The number of previously unsent segments (prevunsnt): The number of segments that a sender has queued for transmission, but has not yet sent.

RTO Restart threshold (rrthresh): RTOR is enabled whenever the sum of the number of outstanding and previously unsent segments (prevunsnt) is below this threshold.

### 3. RTO Overview and Rationale for RTOR

The RTO management algorithm described in [RFC6298] recommends that the retransmission timer is restarted when an acknowledgment (ACK) that acknowledges new data is received and there is still outstanding data. The restart is conducted to guarantee that unacknowledged segments will be retransmitted after approximately RTO seconds. The standardized RTO timer management is illustrated in Figure 1 where a TCP sender transmits three segments to a receiver. The arrival of the first and second segment triggers a delayed ACK (delACK) [RFC1122], which restarts the RTO timer at the sender. The RTO is restarted approximately one RTT after the transmission of the third segment. Thus, if the third segment is lost, as indicated in Figure 1, the effective loss detection time become "RTO + RTT" seconds. In some situations, the effective loss detection time becomes even longer. Consider a scenario where only two segments are outstanding. If the second segment is lost, the time to expire the delACK timer will also be included in the effective loss detection time.

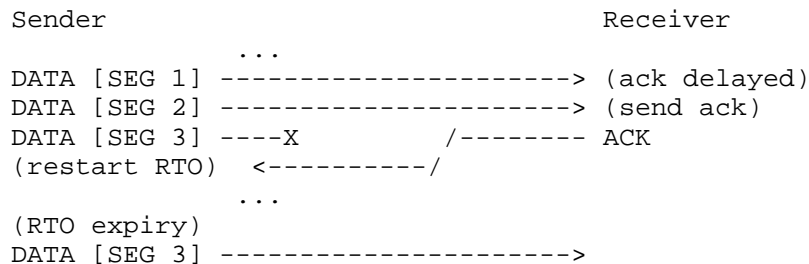


Figure 1: RTO restart example

For bulk traffic the current approach is beneficial -- it is described in [EL04] to act as a "safety margin" that compensates for some of the problems that the authors have identified with the standard RTO calculation. Notably, the authors of [EL04] also state that "this safety margin does not exist for highly interactive applications where often only a single packet is in flight". In general, however, as long as enough segments arrive at a receiver to enable fast retransmit, RTO-based loss recovery should be avoided. RTOs should only be used as a last resort, as they drastically lower the congestion window compared to fast retransmit.

Although fast retransmit is preferable there are situations where timeouts are appropriate, or the only choice. For example, if the network is severely congested and no segments arrive RTO-based recovery should be used. In this situation, the time to recover from the loss(es) will not be the performance bottleneck. However, for connections that do not utilize enough capacity to enable fast retransmit, RTO-based loss detection is the only choice and the time required for this can become a performance bottleneck.

#### 4. RTOR Algorithm

To enable faster loss recovery for connections that are unable to use fast retransmit, RTOR can be used. This section specifies the modifications required to use RTOR. By resetting the timer to "RTO - T\_earliest", where T\_earliest is the time elapsed since the earliest outstanding segment was transmitted, retransmissions will always occur after exactly RTO seconds.

This document specifies an OPTIONAL sender-only modification to TCP and SCTP which updates step 5.3 in Section 5 of [RFC6298] (and a similar update in Section 6.3.2 of [RFC4960] for SCTP). A sender that implements this method MUST follow the algorithm below:

When an ACK is received that acknowledges new data:

- (1) Set T\_earliest = 0.
- (2) If the sum of the number of outstanding and previously unsent segments (prevunsnt) is less than an RTOR threshold (rrthresh), set T\_earliest to the time elapsed since the earliest outstanding segment was sent.
- (3) Restart the retransmission timer so that it will expire after (for the current value of RTO):
  - (a)  $RTO - T_{earliest}$ , if  $RTO - T_{earliest} > 0$ .
  - (b) RTO, otherwise.

The RECOMMENDED value of rrthresh is four, as this value will ensure that RTOR is only used when fast retransmit cannot be triggered. With this update, TCP implementations MUST track the time elapsed since the transmission of the earliest outstanding segment (T\_earliest). As RTOR is only used when the amount of outstanding and previously unsent data is less than rrthresh segments, TCP implementations also need to track whether the amount of outstanding and previously unsent data is more, equal, or less than rrthresh segments. Although some packet-based TCP implementations (e.g.

Linux TCP) already track both the transmission times of all segments and also the number of outstanding segments, not all implementations do. Section 5.3 describes how to implement segment tracking for a general TCP implementation. To use RTOR, the calculated expiration time MUST be positive (step 3(a) in the list above); this is required to ensure that RTOR does not trigger retransmissions prematurely when previously retransmitted segments are acknowledged.

## 5. Discussion

Although RTOR conforms to the requirement in [RFC6298] that segments must not be retransmitted earlier than RTO seconds after their original transmission, RTOR makes the effective RTO more aggressive. In this section, we discuss the applicability and the issues related to RTOR.

### 5.1. Applicability

The currently standardized algorithm has been shown to add at least one RTT to the loss recovery process in TCP [LS00] and SCTP [HB11][PBP09]. For applications that have strict timing requirements (e.g. interactive web) rather than throughput requirements, using RTOR could be beneficial because the RTT and also the delACK timer of receivers are often large components of the effective loss recovery time. Measurements in [HB11] have shown that the total transfer time of a lost segment (including the original transmission time and the loss recovery time) can be reduced by 35% using RTOR. These results match those presented in [PGH06][PBP09], where RTOR is shown to significantly reduce retransmission latency.

There are also traffic types that do not benefit from RTOR. One example of such traffic is bulk transmission. The reason why bulk traffic does not benefit from RTOR is that such traffic flows mostly have four or more segments outstanding, allowing loss recovery by fast retransmit. However, there is no harm in using RTOR for such traffic as the algorithm only is active when the amount of outstanding and unsent segments are less than `rrthresh` (default 4).

Given that RTOR is a mostly conservative algorithm, it is suitable for experimentation as a system-wide default for TCP traffic.

### 5.2. Spurious Timeouts

RTOR can in some situations reduce the loss detection time and thereby increase the risk of spurious timeouts. In theory, the retransmission timer has a lower bound of 1 second [RFC6298], which limits the risk of having spurious timeouts. However, in practice most implementations use a significantly lower value. Initial



measurements show slight increases in the number of spurious timeouts when such lower values are used [RHB15]. However, further experiments, in different environments and with different types of traffic, are encouraged to quantify such increases more reliably.

Does a slightly increased risk matter? Generally, spurious timeouts have a negative effect on the network as segments are transmitted needlessly. However, recent experiments do not show a significant increase in network load for a number of realistic scenarios [RHB15]. Another problem with spurious retransmissions is related to the performance of TCP/SCTP, as the congestion window is reduced to one segment when timeouts occur [RFC5681]. This could be a potential problem for applications transmitting multiple bursts of data within a single flow, e.g. web-based HTTP/1.1 and HTTP/2.0 applications. However, results from recent experiments involving persistent web traffic [RHB15] revealed a net gain of using RTOR. Other types of flows, e.g. long-lived bulk flows, are not affected as the algorithm is only applied when the amount of outstanding and unsent segments is less than `rrthresh`. Furthermore, short-lived and application-limited flows are typically not affected as they are too short to experience the effect of congestion control or have a transmission rate that is quickly attainable.

While a slight increase in spurious timeouts has been observed using RTOR, it is not clear whether the effects of this increase mandate any future algorithmic changes or not -- especially since most modern operating systems already include mechanisms to detect [RFC3522][RFC3708][RFC5682] and resolve [RFC4015] possible problems with spurious retransmissions. Further experimentation is needed to determine this and thereby move this specification from experimental to the standards track. For instance, RTOR has not been evaluated in the context of mobile networks. Mobile networks often incur highly variable RTTs (delay spikes), due to e.g. handovers, and would therefore be a useful scenario for further experimentation.

### 5.3. Tracking Outstanding and Previously Unsent Segments

The method of tracking outstanding and previously unsent segments will probably differ depending on the actual TCP implementation. For packet-based TCP implementations, tracking outstanding segments is often straightforward and can be implemented using a simple counter. For byte-based TCP stacks it is a more complex task. Section 3.2 of [RFC5827] outlines a general method of tracking the number of outstanding segments. The same method can be used for RTOR. The implementation will have to track segment boundaries to form an understanding as to how many actual segments have been transmitted, but not acknowledged. This can be done by the sender tracking the boundaries of the `rrthresh` segments on the right side of the current

window (which involves tracking  $rrthresh + 1$  sequence numbers in TCP). This could be done by keeping a circular list of the segment boundaries, for instance. Cumulative ACKs that do not fall within this region indicate that at least  $rrthresh$  segments are outstanding, and therefore RTOR is not enabled. When the outstanding window becomes small enough that RTOR can be invoked, a full understanding of the number of outstanding segments will be available from the  $rrthresh + 1$  sequence numbers retained. (Note: the implicit sequence number consumed by the TCP FIN bit can also be included in the tracking of segment boundaries.)

Tracking the number of previously unsent segments depends on the segmentation strategy used by the TCP implementation, not whether it is packet-based or byte-based. In the case segments are formed directly on socket writes, the process of determining the number of previously unsent segments should be trivial. In the case that unsent data can be segmented (or re-segmented) as long as it still is unsent, a straightforward strategy could be to divide the amount of unsent data (in bytes) with the SMSS to obtain an estimate. In some cases, such an estimation could be too simplistic, depending on the segmentation strategy of the TCP implementation. However, this estimation is not critical to RTOR. The tracking of `prevunsnt` is only made to optimize a corner case in which RTOR was unnecessarily disabled. Implementations can use a simplified method by setting `prevunsnt` to `rrthresh` whenever previously unsent data is available, and set `prevunsnt` to zero when no new data is available. This will disable RTOR in the presence of unsent data and only use the number of outstanding segments to enable/disable RTOR.

## 6. Related Work

There are several proposals that address the problem of not having enough ACKs for loss recovery. In what follows, we explain why the mechanism described here is complementary to these approaches:

The limited transmit mechanism [RFC3042] allows a TCP sender to transmit a previously unsent segment for each of the first two dupACKs. By transmitting new segments, the sender attempts to generate additional dupACKs to enable fast retransmit. However, limited transmit does not help if no previously unsent data is ready for transmission. [RFC5827] specifies an early retransmit algorithm to enable fast loss recovery in such situations. By dynamically lowering the number of dupACKs needed for fast retransmit (`dupthresh`), based on the number of outstanding segments, a smaller number of dupACKs is needed to trigger a retransmission. In some situations, however, the algorithm is of no use or might not work properly. First, if a single segment is outstanding, and lost, it is impossible to use early retransmit. Second, if ACKs are lost, early

retransmit cannot help. Third, if the network path reorders segments, the algorithm might cause more spurious retransmissions than fast retransmit. The recommended value of RTOR's `rrthresh` variable is based on the `dupthresh`, but is possible to adapt to allow tighter integration with other experimental algorithms such as early retransmit.

Tail Loss Probe [TLP] is a proposal to send up to two "probe segments" when a timer fires which is set to a value smaller than the RTO. A "probe segment" is a new segment if new data is available, else a retransmission. The intention is to compensate for sluggish RTO behavior in situations where the RTO greatly exceeds the RTT, which, according to measurements reported in [TLP], is not uncommon. Furthermore, TLP also tries to circumvent the congestion window reset to one segment by instead enabling fast recovery. The Probe timeout (PTO) is normally two RTTs, and a spurious PTO is less risky than a spurious RTO because it would not have the same negative effects (clearing the scoreboard and restarting with slow-start). TLP is a more advanced mechanism than RTOR, requiring e.g. SACK to work, and is often able to reduce loss recovery times more. However, it also increases the amount of spurious retransmissions noticeably, as compared to RTOR [RHB15].

TLP is applicable in situations where RTOR does not apply, and it could overrule (yielding a similar general behavior, but with a lower timeout) RTOR in cases where the number of outstanding segments is smaller than four and no new segments are available for transmission. The PTO has the same inherent problem of restarting the timer on an incoming ACK, and could be combined with a strategy similar to RTOR's to offer more consistent timeouts.

## 7. SCTP Socket API Considerations

This section describes how the socket API for SCTP defined in [RFC6458] is extended to control the usage of RTO restart for SCTP.

Please note that this section is informational only.

### 7.1. Data Types

This section uses data types from [IEEE.1003-1G.1997]: `uintN_t` means an unsigned integer of exactly N bits (e.g., `uint16_t`). This is the same as in [RFC6458].

## 7.2. Socket Option for Controlling the RTO Restart Support (SCTP\_RTO\_RESTART)

This socket option allows the enabling or disabling of RTO Restart for SCTP associations.

Whether RTO Restart is enabled or not per default is implementation specific.

This socket option uses IPPROTO\_SCTP as its level and SCTP\_RTO\_RESTART as its name. It can be used with getsockopt() and setsockopt(). The socket option value uses the following structure defined in [RFC6458]:

```
struct sctp_assoc_value {
    sctp_assoc_t assoc_id;
    uint32_t assoc_value;
};
```

**assoc\_id:** This parameter is ignored for one-to-one style sockets. For one-to-many style sockets, this parameter indicates upon which association the user is performing an action. The special sctp\_assoc\_t SCTP\_{FUTURE|CURRENT|ALL}\_ASSOC can also be used in assoc\_id for setsockopt(). For getsockopt(), the special value SCTP\_FUTURE\_ASSOC can be used in assoc\_id, but it is an error to use SCTP\_{CURRENT|ALL}\_ASSOC in assoc\_id.

**assoc\_value:** A non-zero value encodes the enabling of RTO restart whereas a value of 0 encodes the disabling of RTO restart.

sctp\_opt\_info() needs to be extended to support SCTP\_RTO\_RESTART.

## 8. IANA Considerations

This memo includes no request to IANA.

## 9. Security Considerations

This document specifies an experimental sender-only modification to TCP and SCTP. The modification introduces a change in how to set the retransmission timer's value when restarted. Therefore, the security considerations found in [RFC6298] apply to this document. No additional security problems have been identified with RTO Restart at this time.

## 10. Acknowledgements

The authors wish to thank Michael Tuexen for contributing the SCTP Socket API considerations and Godred Fairhurst, Yuchung Cheng, Mark Allman, Anantha Ramaiah, Richard Scheffenegger, Nicolas Kuhn, Alexander Zimmermann, and Michael Scharf for commenting on the draft and the ideas behind it.

All the authors are supported by RITE (<http://riteproject.eu/>), a research project (ICT-317700) funded by the European Community under its Seventh Framework Program. The views expressed here are those of the author(s) only. The European Commission is not liable for any use that may be made of the information in this document.

## 11. Changes from Previous Versions

RFC-Editor note: please remove this section prior to publication.

### 11.1. Changed from draft-ietf-...-09 to -10

- o Changed wording in abstract, from "delay" to "timeout duration".

### 11.2. Changed from draft-ietf-...-08 to -09

- o Clarified, in the abstract, that the modified restart causes a smaller retransmission delay in total.
- o Clarified, in the introduction, that the fast retransmit algorithm may cause retransmissions upon receiving duplicate acknowledgments, not that it unconditionally does so.
- o Changed wording from "to proposed standard" to "to the standards track".
- o Changed algorithm description so that a TCP sender MUST track the time elapsed since the transmission of the earliest outstanding segment. This was not explicitly stated in previous versions of the draft.

### 11.3. Changes from draft-ietf-...-07 to -08

- o Clarified, at multiple places in the document, that the modification only causes the effective RTO to be more aggressive, not the actual RTO.
- o Removed information in the introduction that was too detailed, i.e., material that is hard to understand without knowing details of the algorithm.

- o Changed the name of Section 3 to more correctly capture the actual contents of the section.
- o Re-arranged the text in Section 3 to have a more logical structure.
- o Moved text from the algorithm description (Section 4) to the introduction of the discussion section (Section 5). The text was discussing the possible effects of the algorithm more than describing the actual algorithm.
- o Clarified why the RECOMMENDED value of rrthresh is four.
- o Reworked the introduction to be suitable for both TCP and SCTP.

#### 11.4. Changes from draft-ietf-...-06 to -07

- o Clarified, at multiple places in the document, that the modification is sender-only.
- o Added an explanation (in the introduction) to why the mechanism is experimental and what experiments are missing.
- o Added a sentence in Section 4 to clarify that the section is the one describing the actual modification.

#### 11.5. Changes from draft-ietf-...-05 to -06

- o Added socket API considerations, after discussing the draft in tsvwg.

#### 11.6. Changes from draft-ietf-...-04 to -05

- o Introduced variable to track the number of previously unsent segments.
- o Clarified many concepts, e.g. extended the description on how to track outstanding and previously unsent segments.
- o Added a reference to initial measurements on the effects of using RTOR.
- o Improved wording throughout the document.

## 11.7. Changes from draft-ietf-...-03 to -04

- o Changed the algorithm to allow RTOR when there is unsent data available, but the cwnd does not allow transmission.
- o Changed the algorithm to not trigger if  $RTOR \leq 0$ .
- o Made minor adjustments throughout the document to adjust for the algorithmic change.
- o Improved the wording throughout the document.

## 11.8. Changes from draft-ietf-...-02 to -03

- o Updated the document to use "RTOR" instead of "RTO Restart" when referring to the modified algorithm.
- o Moved document terminology to a section of its own.
- o Introduced the `rrthresh` variable in the terminology section.
- o Added a section to generalize the tracking of outstanding segments.
- o Updated the algorithm to work when the number of outstanding segments is less than four and one segment is ready for transmission, by restarting the timer when new data has been sent.
- o Clarified the relationship between fast retransmit and RTOR.
- o Improved the wording throughout the document.

## 11.9. Changes from draft-ietf-...-01 to -02

- o Changed the algorithm description in Section 3 to use formal RFC 2119 language.
- o Changed last paragraph of Section 3 to clarify why the RTO restart algorithm is active when less than four segments are outstanding.
- o Added two paragraphs in Section 4.1 to clarify why the algorithm can be turned on for all TCP traffic without having any negative effects on traffic patterns that do not benefit from a modified timer restart.
- o Improved the wording throughout the document.
- o Replaced and updated some references.

## 11.10. Changes from draft-ietf-...-00 to -01

- o Improved the wording throughout the document.
- o Removed the possibility for a connection limited by the receiver's advertised window to use RTO restart, decreasing the risk of spurious retransmission timeouts.
- o Added a section that discusses the applicability of and problems related to the RTO restart mechanism.
- o Updated the text describing the relationship to TLP to reflect updates made in this draft.
- o Added acknowledgments.

## 12. References

## 12.1. Normative References

- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<http://www.rfc-editor.org/info/rfc1122>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3042] Allman, M., Balakrishnan, H., and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, DOI 10.17487/RFC3042, January 2001, <<http://www.rfc-editor.org/info/rfc3042>>.
- [RFC3522] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC 3522, DOI 10.17487/RFC3522, April 2003, <<http://www.rfc-editor.org/info/rfc3522>>.
- [RFC3708] Blanton, E. and M. Allman, "Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions", RFC 3708, DOI 10.17487/RFC3708, February 2004, <<http://www.rfc-editor.org/info/rfc3708>>.



- [RFC4015] Ludwig, R. and A. Gurtov, "The Eifel Response Algorithm for TCP", RFC 4015, DOI 10.17487/RFC4015, February 2005, <<http://www.rfc-editor.org/info/rfc4015>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<http://www.rfc-editor.org/info/rfc4960>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<http://www.rfc-editor.org/info/rfc5681>>.
- [RFC5682] Sarolahti, P., Kojo, M., Yamamoto, K., and M. Hata, "Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP", RFC 5682, DOI 10.17487/RFC5682, September 2009, <<http://www.rfc-editor.org/info/rfc5682>>.
- [RFC5827] Allman, M., Avrachenkov, K., Ayesta, U., Blanton, J., and P. Hurtig, "Early Retransmit for TCP and Stream Control Transmission Protocol (SCTP)", RFC 5827, DOI 10.17487/RFC5827, May 2010, <<http://www.rfc-editor.org/info/rfc5827>>.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <<http://www.rfc-editor.org/info/rfc6298>>.

## 12.2. Informative References

- [EL04] Ekstroem, H. and R. Ludwig, "The Peak-Hopper: A New End-to-End Retransmission Timer for Reliable Unicast Transport", IEEE INFOCOM 2004, March 2004.
- [FDT13] Flach, T., Dukkipati, N., Terzis, A., Raghavan, B., Cardwell, N., Cheng, Y., Jain, A., Hao, S., Katz-Bassett, E., and R. Govindan, "Reducing Web Latency: the Virtue of Gentle Aggression", Proc. ACM SIGCOMM Conf., August 2013.
- [HB11] Hurtig, P. and A. Brunstrom, "SCTP: designed for timely message delivery?", Springer Telecommunication Systems 47 (3-4), August 2011.
- [IEEE.1003-1G.1997] Institute of Electrical and Electronics Engineers, "Protocol Independent Interfaces", IEEE Standard 1003.1G, March 1997.

- [LS00] Ludwig, R. and K. Sklower, "The Eifel retransmission timer", ACM SIGCOMM Comput. Commun. Rev., 30(3), July 2000.
- [P09] Petlund, A., "Improving latency for interactive, thin-stream applications over reliable transport", Unipub PhD Thesis, Oct 2009.
- [PBP09] Petlund, A., Beskow, P., Pedersen, J., Paaby, E., Griwodz, C., and P. Halvorsen, "Improving SCTP Retransmission Delays for Time-Dependent Thin Streams", Springer Multimedia Tools and Applications, 45(1-3), 2009.
- [PGH06] Pedersen, J., Griwodz, C., and P. Halvorsen, "Considerations of SCTP Retransmission Delays for Thin Streams", IEEE LCN 2006, November 2006.
- [RFC6458] Stewart, R., Tuexen, M., Poon, K., Lei, P., and V. Yasevich, "Sockets API Extensions for the Stream Control Transmission Protocol (SCTP)", RFC 6458, DOI 10.17487/RFC6458, December 2011, <<http://www.rfc-editor.org/info/rfc6458>>.
- [RHB15] Rajiullah, M., Hurtig, P., Brunstrom, A., Petlund, A., and M. Welzl, "An Evaluation of Tail Loss Recovery Mechanisms for TCP", ACM SIGCOMM CCR 45 (1), January 2015.
- [RJ10] Ramachandran, S., "Web metrics: Size and number of resources", Google <http://code.google.com/speed/articles/web-metrics.html>, May 2010.
- [TLP] Dukkupati, N., Cardwell, N., Cheng, Y., and M. Mathis, "TCP Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses", Internet-draft draft-dukkupati-tcpm-tcp-loss-probe-01.txt, February 2013.

#### Authors' Addresses

Per Hurtig  
Karlstad University  
Universitetsgatan 2  
Karlstad 651 88  
Sweden

Phone: +46 54 700 23 35  
Email: [per.hurtig@kau.se](mailto:per.hurtig@kau.se)

Anna Brunstrom  
Karlstad University  
Universitetsgatan 2  
Karlstad 651 88  
Sweden

Phone: +46 54 700 17 95  
Email: anna.brunstrom@kau.se

Andreas Petlund  
Simula Research Laboratory AS  
P.O. Box 134  
Lysaker 1325  
Norway

Phone: +47 67 82 82 00  
Email: apetlund@simula.no

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

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

TCP Maintenance & Minor Extensions (tcpm)  
Internet-Draft  
Intended status: Experimental  
Expires: April 21, 2016

B. Briscoe  
Simula Research Laboratory  
M. Kuehlewind  
ETH Zurich  
R. Scheffenegger  
NetApp, Inc.  
October 19, 2015

More Accurate ECN Feedback in TCP  
draft-kuehlewind-tcpm-accurate-ecn-05

Abstract

Explicit Congestion Notification (ECN) is a mechanism where network nodes can mark IP packets instead of dropping them to indicate incipient congestion to the end-points. Receivers with an ECN-capable transport protocol feed back this information to the sender. ECN is specified for TCP in such a way that only one feedback signal can be transmitted per Round-Trip Time (RTT). Recently, new TCP mechanisms like Congestion Exposure (ConEx) or Data Center TCP (DCTCP) need more accurate ECN feedback information whenever more than one marking is received in one RTT. This document specifies an experimental scheme to provide more than one feedback signal per RTT in the TCP header. Given TCP header space is scarce, it overloads the three existing ECN-related flags in the TCP header and provides additional information in a new TCP option.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 21, 2016.

## Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Document Roadmap . . . . .	4
1.2. Goals . . . . .	4
1.3. Experiment Goals . . . . .	5
1.4. Terminology . . . . .	5
1.5. Recap of Existing ECN feedback in IP/TCP . . . . .	6
2. AcceCN Protocol Overview and Rationale . . . . .	7
2.1. Capability Negotiation . . . . .	8
2.2. Feedback Mechanism . . . . .	8
2.3. Delayed ACKs and Resilience Against ACK Loss . . . . .	9
2.4. Feedback Metrics . . . . .	10
2.5. Generic (Dumb) Reflector . . . . .	10
3. AcceCN Protocol Specification . . . . .	11
3.1. Negotiation during the TCP handshake . . . . .	11
3.2. AcceCN Feedback . . . . .	14
3.2.1. The ACE Field . . . . .	14
3.2.2. Safety against Ambiguity of the ACE Field . . . . .	16
3.2.3. The AcceCN Option . . . . .	16
3.2.4. Path Traversal of the AcceCN Option . . . . .	17
3.2.5. Usage of the AcceCN TCP Option . . . . .	19
3.3. AcceCN Compliance by TCP Proxies, Offload Engines and other Middleboxes . . . . .	20
4. Interaction with Other TCP Variants . . . . .	21
4.1. Compatibility with SYN Cookies . . . . .	21
4.2. Compatibility with Other TCP Options and Experiments . . . . .	21
4.3. Compatibility with Feedback Integrity Mechanisms . . . . .	21
5. Protocol Properties . . . . .	23
6. IANA Considerations . . . . .	25
7. Security Considerations . . . . .	25
8. Acknowledgements . . . . .	26
9. Comments Solicited . . . . .	26

10. References	26
10.1. Normative References	26
10.2. Informative References	27
Appendix A. Example Algorithms	29
A.1. Example Algorithm to Encode/Decode the AcceCN Option	29
A.2. Example Algorithm for Safety Against Long Sequences of ACK Loss	30
A.2.1. Safety Algorithm without the AcceCN Option	30
A.2.2. Safety Algorithm with the AcceCN Option	32
A.3. Example Algorithm to Estimate Marked Bytes from Marked Packets	33
A.4. Example Algorithm to Beacon AcceCN Options	34
A.5. Example Algorithm to Count Not-ECT Bytes	35
Appendix B. Alternative Design Choices (To Be Removed Before Publication)	35
Appendix C. Open Protocol Design Issues (To Be Removed Before Publication)	36
Appendix D. Changes in This Version (To Be Removed Before Publication)	37
Authors' Addresses	37

## 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 the associated requirements document [RFC7560].

This document specifies an experimental scheme for ECN feedback in the TCP header to provide more than one feedback signal per RTT. It will be called the more accurate ECN feedback scheme, or AcceCN for short. If AcceCN progresses from experimental to the standards track, it is intended to be a complete replacement for classic ECN feedback, not a fork in the design of TCP. Thus, the applicability of AcceCN is intended to include all public and private IP networks (and even any non-IP networks over which TCP is used today). Until the AcceCN experiment succeeds, [RFC3168] will remain as the standards track specification for adding ECN to TCP. To avoid confusion, in this document we use the term 'classic ECN' for the pre-existing ECN specification [RFC3168].

AcceECN 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.

It is likely (but not required) that the AcceECN protocol will be implemented along with the following experimental additions to the TCP-ECN protocol: ECN-capable SYN/ACK [RFC5562], ECN path-probing and fall-back [I-D.kuehlewind-tcpm-ecn-fallback] and testing receiver non-compliance [I-D.moncaster-tcpm-rcv-cheat].

### 1.1. Document Roadmap

The following introductory sections outline the goals of AcceECN (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 AcceECN protocol. Then Section 3 gives the normative protocol specification. Section 4 assesses the interaction of AcceECN with commonly used variants of TCP, whether standardised or not. Section 5 summarises the features and properties of AcceECN.

Section 6 summarises the protocol fields and numbers that IANA will need to assign and Section 7 points to the aspects of the protocol that will be of interest to the security community.

Appendix A gives pseudocode examples for the various algorithms that AcceECN uses.

### 1.2. Goals

[RFC7560] enumerates requirements that a candidate feedback scheme will need to satisfy, under the headings: resilience, timeliness, integrity, accuracy (including ordering and lack of bias), complexity, overhead and compatibility (both backward and forward). It recognises that a perfect scheme that fully satisfies all the requirements is unlikely and trade-offs between requirements are likely. Section 5 presents the properties of AcceECN against these requirements and discusses the trade-offs made.

The requirements document recognises that a protocol as ubiquitous as TCP needs to be able to serve as-yet-unspecified requirements. Therefore an AccECN receiver aims to act as a generic (dumb) reflector of congestion information so that in future new sender behaviours can be deployed unilaterally.

### 1.3. Experiment Goals

TCP is critical to the robust functioning of the Internet, therefore any proposed modifications to TCP need to be thoroughly tested. The present specification describes an experimental protocol that adds more accurate ECN feedback to the TCP protocol. The intention is to specify the protocol sufficiently so that more than one implementation can be built in order to test its function, robustness and interoperability (with itself and with previous version of ECN and TCP).

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

### 1.4. Terminology

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

Classic ECN: the ECN protocol specified in [RFC3168].

Classic ECN feedback: the feedback aspect of the ECN protocol specified in [RFC3168], including generation, encoding, transmission and decoding of feedback, but not the Data Sender's subsequent response to that feedback.

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

Pure ACK: A TCP acknowledgement without a data payload.

TCP client: The TCP stack that originates a connection.

TCP server: The TCP stack that responds to a connection request.



**Data Receiver:** The endpoint of a TCP half-connection that receives data and sends AccECN feedback.

**Data Sender:** The endpoint of a TCP half-connection that sends data and receives AccECN feedback.

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

### 1.5. Recap of Existing ECN feedback in IP/TCP

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

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

Table 1: The ECN Field in the IP Header

In the TCP header the first two bits in byte 14 are defined as flags for the use of ECN (CWR and ECE in Figure 1 [RFC3168]). A TCP client indicates it supports ECN by setting ECE=CWR=1 in the SYN, and an ECN-enabled server confirms ECN support by setting ECE=1 and CWR=0 in the SYN/ACK. On reception of a CE-marked packet at the IP layer, the Data Receiver starts to set the Echo Congestion Experienced (ECE) flag continuously in the TCP header of ACKs, which ensures the signal is received reliably even if ACKs are lost. The TCP sender confirms that it has received at least one ECE signal by responding with the congestion window reduced (CWR) flag, which allows the TCP receiver to stop repeating the ECN-Echo flag. This always leads to a full RTT

of ACKs with ECE set. Thus any additional CE markings arriving within this RTT cannot be fed back.

The 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.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Header Length				Reserved		N	C	E	U	A	P	R	S	F	
						S	W	C	R	C	S	S	Y	I	
							R	E	G	K	H	T	N	N	

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

## 2. AcceCN Protocol Overview and Rationale

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

Like the original TCP approach, the Data Receiver of each TCP half-connection sends AcceCN feedback to the Data Sender on TCP acknowledgements, reusing data packets of the other half-connection whenever possible.

The AcceCN protocol has had to be designed in two parts:

- o an essential part that re-uses ECN TCP header bits to feed back the number of arriving CE marked packets. This provides more accuracy than classic ECN feedback, but limited resilience against ACK loss;
- o a supplementary part using a new AcceCN TCP Option that provides additional feedback on the number of bytes that arrive marked with each of the three ECN codepoints (not just CE marks). This provides greater resilience against ACK loss than the essential feedback, but it is more likely to suffer from middlebox interference.

The two part design was necessary, given limitations on the space available for TCP options and given the possibility that certain incorrectly designed middleboxes prevent TCP using any new options.

The essential part overloads the previous definition of the three flags in the TCP header that had been assigned for use by ECN. This design choice deliberately replaces the classic ECN feedback protocol, rather than leaving classic ECN feedback intact and adding more accurate feedback separately because:

- o this efficiently reuses scarce TCP header space, given TCP option space is approaching saturation;
- o a single upgrade path for the TCP protocol is preferable to a fork in the design;
- o otherwise classic and accurate ECN feedback could give conflicting feedback on the same segment, which could open up new security concerns and make implementations unnecessarily complex;
- o middleboxes are more likely to faithfully forward the TCP ECN flags than newly defined areas of the TCP header.

AcceECN is designed to work even if the supplementary part is removed or zeroed out, as long as the essential part gets through.

## 2.1. Capability Negotiation

AcceECN is a change to the wire protocol of the main TCP header, therefore it can only be used if both endpoints have been upgraded to understand it. The TCP client signals support for AcceECN on the initial SYN of a connection and the TCP server signals whether it supports AcceECN on the SYN/ACK. The TCP flags on the SYN that the client uses to signal AcceECN support have been carefully chosen so that a TCP server will interpret them as a request to support the most recent variant of ECN feedback that it supports. Then the client falls back to the same variant of ECN feedback.

An AcceECN TCP client does not send the new AcceECN Option on the SYN as SYN option space is limited and successful negotiation using the flags in the main header is taken as sufficient evidence that both ends also support the AcceECN Option. The TCP server sends the AcceECN Option on the SYN/ACK and the client sends it on the first ACK to test whether the network path forwards the option correctly.

## 2.2. Feedback Mechanism

A Data Receiver maintains four counters initialised at the start of the half-connection. Three count the number of arriving payload bytes marked CE, ECT(1) and ECT(0) respectively. The fourth counts the number of packets arriving marked with a CE codepoint (including control packets without payload if they are CE-marked).

The Data Sender maintains four equivalent counters for the half connection, and the AccECN protocol is designed to ensure they will match the values in the Data Receiver's counters, albeit after a little delay.

Each ACK carries the three least significant bits (LSBs) of the packet-based CE counter using the ECN bits in the TCP header, now renamed the Accurate ECN (ACE) field. The LSBs of each of the three byte counters are carried in the AccECN Option.

### 2.3. Delayed ACKs and Resilience Against ACK Loss

With both the ACE and the AccECN Option mechanisms, the Data Receiver continually repeats the current LSBs of each of its respective counters. Then, even if some ACKs are lost, the Data Sender should be able to infer how much to increment its own counters, even if the protocol field has wrapped.

The 3-bit ACE field can wrap fairly frequently. Therefore, even if it appears to have incremented by one (say), the field might have actually cycled completely then incremented by one. The Data Receiver is required not to delay sending an ACK to such an extent that the ACE field would cycle. However cycling is still a possibility at the Data Sender because a whole sequence of ACKs carrying intervening values of the field might all be lost or delayed in transit.

The fields in the AccECN Option are larger, but they will increment in larger steps because they count bytes not packets. Nonetheless, their size has been chosen such that a whole cycle of the field would never occur between ACKs unless there had been an infeasibly long sequence of ACK losses. Therefore, as long as the AccECN Option is available, it can be treated as a dependable feedback channel.

If the AccECN Option is not available, e.g. it is being stripped by a middlebox, the AccECN protocol will only feed back information on CE markings (using the ACE field). Although not ideal, this will be sufficient, because it is envisaged that neither ECT(0) nor ECT(1) will ever indicate more severe congestion than CE, even though future uses for ECT(0) or ECT(1) are still unclear. Because the 3-bit ACE field is so small, when it is the only field available the Data Sender has to interpret it conservatively assuming the worst possible wrap.

Certain specified events trigger the Data Receiver to include an AccECN Option on an ACK. The rules are designed to ensure that the order in which different markings arrive at the receiver is communicated to the sender (as long as there is no ACK loss).

Implementations are encouraged to send an AcceECN Option more frequently, but this is left up to the implementer.

#### 2.4. Feedback Metrics

The CE packet counter in the ACE field and the CE byte counter in the AcceECN Option both provide feedback on received CE-marks. The CE packet counter includes control packets that do not have payload data, while the CE byte counter solely includes marked payload bytes. If both are present, the byte counter in the option will provide the more accurate information needed for modern congestion control and policing schemes, such as DCTCP or ConEx. If the option is stripped, a simple algorithm to estimate the number of marked bytes from the ACE field is given in Appendix A.3.

Feedback in bytes is recommended in order to protect against the receiver using attacks similar to 'ACK-Division' to artificially inflate the congestion window, which is why [RFC5681] now recommends that TCP counts acknowledged bytes not packets.

#### 2.5. Generic (Dumb) Reflector

The ACE field provides information about CE markings on both data and control packets. According to [RFC3168] the Data Sender is meant to set control packets to Not-ECT. However, mechanisms in certain private networks (e.g. data centres) set control packets to be ECN capable because they are precisely the packets that performance depends on most.

For this reason, AcceECN is designed to be a generic reflector of whatever ECN markings it sees, whether or not they are compliant with a current standard. Then as standards evolve, Data Senders can upgrade unilaterally without any need for receivers to upgrade too. It is also useful to be able to rely on generic reflection behaviour when senders need to test for unexpected interference with markings (for instance [I-D.kuehlewind-tcpm-ecn-fallback] and [I-D.moncaster-tcpm-rcv-cheat]).

The initial SYN is the most critical control packet, so AcceECN provides feedback on whether it is CE marked, even though it is not allowed to be ECN-capable according to RFC 3168. However, middleboxes have been known to overwrite the ECN IP field as if it is still part of the old Type of Service (ToS) field. If a TCP client has set the SYN to Not-ECT, but receives CE feedback, it can detect such middlebox interference and send Not-ECT for the rest of the connection (see [I-D.kuehlewind-tcpm-ecn-fallback] for the detailed fall-back behaviour).

Today, if a TCP server receives CE on a SYN, it cannot know whether it is invalid (or valid) because only the TCP client knows whether it originally marked the SYN as Not-ECT (or ECT). Therefore, the server's only safe course of action is to disable ECN for the connection. Instead, the AccECN protocol allows the server to feed back the CE marking to the client, which then has all the information to decide whether the connection has to fall-back from supporting ECN (or not).

Providing feedback of CE marking on the SYN also supports future scenarios in which SYNs might be ECN-enabled (without prejudging whether they ought to be). For instance, in certain environments such as data centres, it might be appropriate to allow ECN-capable SYNs. Then, if feedback showed the SYN had been CE marked, the TCP client could reduce its initial window (IW). It could also reduce IW conservatively if feedback showed the receiver did not support ECN (because if there had been a CE marking, the receiver would not have understood it). Note that this text merely motivates dumb reflection of CE on a SYN, it does not judge whether a SYN ought to be ECN-capable.

### 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 TCP client (host A) MUST set the TCP flags NS=1, CWR=1 and ECE=1 in the initial SYN segment.

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

A TCP server in AccECN mode MUST additionally set the flag NS=1 on the SYN/ACK if the SYN was CE-marked (see Section 2.5). If the received SYN was Not-ECT, ECT(0) or ECT(1), it MUST clear NS (NS=0) on the SYN/ACK.

Once a TCP client (A) has sent the above SYN to declare that it supports AccECN, and once it has received the above SYN/ACK segment that confirms that the TCP server supports AccECN, the TCP client MUST set both its half connections into AccECN mode.

If after the normal TCP timeout the TCP client has not received a SYN/ACK to acknowledge its SYN, the SYN might just have been lost,

e.g. due to congestion, or a middlebox might be blocking segments with the AccECN flags. To expedite connection setup, the host SHOULD fall back to NS=CWR=ECE=0 on the retransmission of the SYN. It would make sense to also remove any other experimental fields or options on the SYN in case a middlebox might be blocking them, although the required behaviour will depend on the specification of the other option(s) and any attempt to co-ordinate fall-back between different modules of the stack. Implementers MAY use other fall-back strategies if they are found to be more effective (e.g. attempting to retransmit a second AccECN segment before fall-back, falling back to classic ECN feedback rather than non-ECN, and/or caching the result of a previous attempt to access the same host while negotiating AccECN).

The fall-back procedure if the TCP server receives no ACK to acknowledge a SYN/ACK that tried to negotiate AccECN is specified in Section 3.2.4.

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			Feedback Mode
AB				NS	CWR	ECE	NS	CWR	ECE	AccECN
AB				1	1	1	0	1	0	AccECN (CE on SYN)
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
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			B	1	1	1	1	1	1	Not ECN (broken)
A				1	1	1	0	1	1	Not ECN (see Appx B)
A				1	1	1	1	0	0	Not ECN (see Appx B)

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 and how the TCP server (B) indicates congestion feedback.
2. The second block shows the cases where the TCP client (A) supports AccECN but the TCP server (B) supports some earlier variant of TCP feedback, indicated in its SYN/ACK. Therefore, as soon as an AccECN-capable TCP client (A) receives the SYN/ACK shown it MUST set both its half connections into the feedback mode shown in the rightmost column.
3. The third block shows the cases where the TCP server (B) supports AccECN but the TCP client (A) supports some earlier variant of TCP feedback, indicated in its SYN. Therefore, as soon as an AccECN-enabled TCP server (B) receives the SYN shown, it MUST set both its half connections into the feedback mode shown in the rightmost column.
4. The fourth block displays combinations that are not valid or currently unused and therefore both ends MUST fall-back to Not ECN for both half connections. Especially the first case (marked 'broken') where all bits set in the SYN are reflected by the receiver in the SYN/ACK, which happens quite often if the TCP



connection is proxied. {ToDo: Consider using the last two cases for AccECN f/b of ECT(0) and ECT(1) on the SYN (Appendix B)}

The following exceptional cases need some explanation:

**ECN Nonce:** An AccECN implementation, whether client or server, sender or receiver, does not need to implement the ECN Nonce behaviour [RFC3540]. AccECN is compatible with an alternative ECN feedback integrity approach that does not use up the ECT(1) codepoint and can be implemented solely at the sender (see Section 4.3).

**Simultaneous Open:** An originating AccECN Host (A), having sent a SYN with NS=1, CWR=1 and ECE=1, might receive another SYN from host B. Host A MUST then enter the same feedback mode as it would have entered had it been a responding host and received the same SYN. Then host A MUST send the same SYN/ACK as it would have sent had it been a responding host (see the third block above).

### 3.2. AccECN Feedback

Each Data Receiver maintains four counters, `r.cep`, `r.ceb`, `r.e0b` and `r.elb`. The CE packet counter (`r.cep`), counts the number of packets the host receives with the CE code point in the IP ECN field, including CE marks on control packets without data. `r.ceb`, `r.e0b` and `r.elb` count the number of TCP payload bytes in packets marked respectively with the CE, ECT(0) and ECT(1) codepoint in their IP-ECN field. When a host first enters AccECN mode, it initialises its counters to `r.cep = 6`, `r.e0b = 1` and `r.ceb = r.elb = 0` (see Appendix A.5). Non-zero initial values are used to be distinct from cases where the fields are incorrectly zeroed (e.g. by middleboxes).

A host feeds back the CE packet counter using the Accurate ECN (ACE) field, as explained in the next section. And it feeds back all the byte counters using the AccECN TCP Option, as specified in Section 3.2.3. Whenever a host feeds back the value of any counter, it MUST report the most recent value, no matter whether it is in a pure ACK, an ACK with new payload data or a retransmission.

#### 3.2.1. The ACE Field

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

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Header Length				Reserved			ACE			U	A	P	R	S	F
										R	C	S	S	Y	I
										G	K	H	T	N	N

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

The original definition of these three flags in the TCP header, including the addition of support for the ECN Nonce, is shown for comparison in Figure 1. This specification does not rename these three TCP flags, it merely overloads them with another name and definition once an AcceECN 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 if both of its half-connections are set into AcceECN mode having successfully negotiated AcceECN (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 AcceECN negotiation is incomplete or has not succeeded.

Both parts of each of these conditions are equally important. For instance, even if AcceECN 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 AcceECN support has been successfully negotiated during a simultaneous open).

The ACE field encodes the three least significant bits of the r.cep counter, therefore its initial value will be 0b110 (decimal 6). This non-zero initialization allows a TCP server to use a stateless handshake (see Section 4.1) but still detect from the TCP client's first ACK that the client considers it has successfully negotiated AcceECN. If the SYN/ACK was CE marked, the client **MUST** increase its r.cep counter before it sends its first ACK, therefore the initial value of the ACE field will be 0b111 (decimal 7). These values have deliberately been chosen such that they are distinct from [RFC5562] behaviour, where the TCP client would set ECE on the first ACK as feedback for a CE mark on the SYN/ACK.

If the value of the ACE field on the first segment with SYN=0 in either direction is anything other than 0b110 or 0b111, the Data Receiver **MUST** disable ECN for the remainder of the half-connection by marking all subsequent packets as Not-ECT.

### 3.2.2. Safety against Ambiguity of the ACE Field

If too many CE-marked segments are acknowledged at once, or if a long run of ACKs is lost, the 3-bit counter in the ACE field might have cycled between two ACKs arriving at the Data Sender.

Therefore an AccECN Data Receiver SHOULD immediately send an ACK once 'n' CE marks have arrived since the previous ACK, where 'n' SHOULD be 2 and MUST be no greater than 6.

If the Data Sender has not received AccECN TCP Options to give it more dependable information, and it detects that the ACE field could have cycled under the prevailing conditions, it SHOULD conservatively assume that the counter did cycle. It can detect if the counter could have cycled by using the jump in the acknowledgement number since the last ACK to calculate or estimate how many segments could have been acknowledged. An example algorithm to implement this policy is given in Appendix A.2. An implementer MAY develop an alternative algorithm as long as it satisfies these requirements.

If missing acknowledgement numbers arrive later (reordering) and prove that the counter did not cycle, the Data Sender MAY attempt to neutralise the effect of any action it took based on a conservative assumption that it later found to be incorrect.

### 3.2.3. The AccECN Option

The AccECN Option is defined as shown below in Figure 3. It consists of three 24-bit fields that provide the 24 least significant bits of the r.e0b, r.ceb and r.elb counters, respectively. The initial 'E' of each field name stands for 'Echo'.

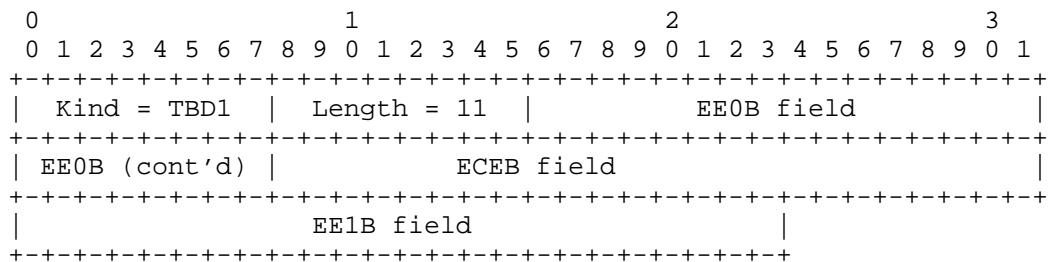


Figure 3: The AccECN Option

The Data Receiver MUST set the Kind field to TBD1, which is registered in Section 6 as a new TCP option Kind called AccECN. An experimental TCP option with Kind=254 MAY be used for initial experiments, with magic number 0xACCE.

Appendix A.1 gives an example algorithm for the Data Receiver to encode its byte counters into the AcceCN Option, and for the Data Sender to decode the AcceCN Option fields into its byte counters.

Note that there is no field to feedback Not-ECT bytes. Nonetheless an algorithm for the Data Sender to calculate the number of payload bytes received as Not-ECT is given in Appendix A.5.

Whenever a Data Receiver sends an AcceCN Option, the rules in Section 3.2.5 expect it to always send a full-length option. To cope with option space limitations, it can omit unchanged fields from the tail of the option, as long as it preserves the order of the remaining fields and includes any field that has changed. The length field MUST indicate which fields are present as follows:

Length=11: EE0B, ECEB, EE1B

Length=8: EE0B, ECEB

Length=5: EE0B

Length=2: (empty)

The empty option of Length=2 is provided to allow for a case where an AcceCN Option has to be sent (e.g. on the SYN/ACK to test the path), but there is very limited space for the option. For initial experiments, the Length field MUST be 2 greater to accommodate the 16-bit magic number.

All implementations of a Data Sender MUST be able to read in AcceCN Options of any of the above lengths. They MUST ignore an AcceCN Option of any other length.

#### 3.2.4. Path Traversal of the AcceCN Option

An AcceCN host MUST NOT include the AcceCN TCP Option on the SYN. Nonetheless, if the AcceCN negotiation using the ECN flags in the main TCP header (Section 3.1) is successful, it implicitly declares that the endpoints also support the AcceCN TCP Option.

If the TCP client indicated AcceCN support, a TCP server that confirms its support for AcceCN (as described in Section 3.1) SHOULD also include an AcceCN TCP Option in the SYN/ACK. A TCP client that has successfully negotiated AcceCN SHOULD include an AcceCN Option in the first ACK at the end of the 3WSH. However, this first ACK is not delivered reliably, so the TCP client SHOULD also include an AcceCN Option on the first data segment it sends (if it ever sends one). A host need not include an AcceCN Option in any of these three cases if

it has cached knowledge that the packet would be likely to be blocked on the path to the other host if it included an AcceCN Option.

If the TCP client has successfully negotiated AcceCN but does not receive an AcceCN Option on the SYN/ACK, it switches into a mode that assumes that the AcceCN Option is not available for this half connection. Similarly, if the TCP server has successfully negotiated AcceCN but does not receive an AcceCN Option on the first ACK or on the first data segment, it switches into a mode that assumes that the AcceCN Option is not available for this half connection.

While a host is in the mode that assumes the AcceCN Option is not available, it MUST adopt the conservative interpretation of the ACE field discussed in Section 3.2.2. However, it cannot make any assumption about support of the AcceCN Option on the other half connection, so it MUST continue to send the AcceCN Option itself.

If after the normal TCP timeout the TCP server has not received an ACK to acknowledge its SYN/ACK, the SYN/ACK might just have been lost, e.g. due to congestion, or a middlebox might be blocking the AcceCN Option. To expedite connection setup, the host SHOULD fall back to NS=CWR=ECE=0 and no AcceCN Option on the retransmission of the SYN/ACK. Implementers MAY use other fall-back strategies if they are found to be more effective (e.g. retransmitting a SYN/ACK with AcceCN TCP flags but not the AcceCN Option; attempting to retransmit a second AcceCN segment before fall-back (most appropriate during high levels of congestion); or falling back to classic ECN feedback rather than non-ECN).

Similarly, if the TCP client detects that the first data segment it sent was lost, it SHOULD fall back to no AcceCN Option on the retransmission. Again, implementers MAY use other fall-back strategies such as attempting to retransmit a second segment with the AcceCN Option before fall-back, and/or caching the result of previous attempts.

Either host MAY include the AcceCN Option in a subsequent segment to retest whether the AcceCN Option can traverse the path.

Currently the Data Sender is not required to test whether the arriving byte counters in the AcceCN Option have been correctly initialised. This allows different initial values to be used as an additional signalling channel in future. If any inappropriate zeroing of these fields is discovered during testing, this approach will need to be reviewed.

### 3.2.5. Usage of the AccECN TCP Option

The following rules determine when a Data Receiver in AccECN mode sends the AccECN TCP Option, and which fields to include:

**Change-Triggered ACKs:** If an arriving packet increments a different byte counter to that incremented by the previous packet, the Data Receiver SHOULD immediately send an ACK with an AccECN Option, without waiting for the next delayed ACK. Certain offload hardware might not be able to support change-triggered ACKs, but otherwise it is important to keep exceptions to this rule to a minimum so that Data Senders can generally rely on this behaviour;

**Continual Repetition:** Otherwise, if arriving packets continue to increment the same byte counter, the Data Receiver can include an AccECN Option on most or all (delayed) ACKs, but it does not have to. If option space is limited on a particular ACK, the Data Receiver MUST give precedence to SACK information about loss. It SHOULD include an AccECN Option if the r.ceb counter has incremented and it MAY include an AccECN Option if r.ec0b or r.ec1b has incremented;

**Full-Length Options Preferred:** It SHOULD always use full-length AccECN Options. It MAY use shorter AccECN Options if space is limited, but it MUST include the counter(s) that have incremented since the previous AccECN Option and it MUST only truncate fields from the right-hand tail of the option to preserve the order of the remaining fields (see Section 3.2.3);

**Beaconing Full-Length Options:** Nonetheless, it MUST include a full-length AccECN TCP Option on at least three ACKs per RTT, or on all ACKs if there are less than three per RTT (see Appendix A.4 for an example algorithm that satisfies this requirement).

The following example series of arriving marks illustrates when a Data Receiver will emit an ACK if it is using a delayed ACK factor of 2 segments and change-triggered ACKs: 01 -> ACK, 01, 01 -> ACK, 10 -> ACK, 10, 01 -> ACK, 01, 11 -> ACK, 01 -> ACK.

For the avoidance of doubt, the change-triggered ACK mechanism ignores the arrival of a control packet with no payload, because it does not alter any byte counters. The change-triggered ACK approach will lead to some additional ACKs but it feeds back the timing and the order in which ECN marks are received with minimal additional complexity.

**Implementation note:** sending an AccECN Option each time a different counter changes and including a full-length AccECN Option on every

delayed ACK will satisfy the requirements described above and might be the easiest implementation, as long as sufficient space is available in each ACK (in total and in the option space).

Appendix A.3 gives an example algorithm to estimate the number of marked bytes from the ACE field alone, if the AcceECN Option is not available.

If a host has determined that segments with the AcceECN Option always seem to be discarded somewhere along the path, it is no longer obliged to follow the above rules.

### 3.3. AcceECN Compliance by TCP Proxies, Offload Engines and other Middleboxes

A large class of middleboxes split TCP connections. Such a middlebox would be compliant with the AcceECN protocol if the TCP implementation on each side complied with the present AcceECN specification and each side negotiated AcceECN independently of the other side.

Another large class of middleboxes intervene to some degree at the transport layer, but attempts to be transparent (invisible) to the end-to-end connection. A subset of this class of middleboxes attempts to 'normalise' the TCP wire protocol by checking that all values in header fields comply with a rather narrow interpretation of the TCP specifications. To comply with the present AcceECN specification, such a middlebox MUST NOT change the ACE field or the AcceECN Option and it MUST attempt to preserve the timing of each ACK (for example, if it coalesced ACKs it would not be AcceECN-compliant). A middlebox claiming to be transparent at the transport layer MUST forward the AcceECN TCP Option unaltered, whether or not the length value matches one of those specified in Section 3.2.3, and whether or not the initial values of the byte-counter fields are correct. This is because blocking apparently invalid values does not improve security (because AcceECN hosts are required to ignore invalid values anyway), while it prevents the standardised set of values being extended in future (because outdated normalisers would block updated hosts from using the extended AcceECN standard).

Hardware to offload certain TCP processing represents another large class of middleboxes, even though it is often a function of a host's network interface and rarely in its own 'box'. Leeway has been allowed in the present AcceECN specification in the expectation that offload hardware could comply and still serve its function. Nonetheless, such hardware MUST attempt to preserve the timing of each ACK (for example, if it coalesced ACKs it would not be AcceECN-compliant).

#### 4. Interaction with Other TCP Variants

This section is informative, not normative.

##### 4.1. Compatibility with SYN Cookies

A TCP server can use SYN Cookies (see Appendix A of [RFC4987]) to protect itself from SYN flooding attacks. It places minimal commonly used connection state in the SYN/ACK, and deliberately does not hold any state while waiting for the subsequent ACK (e.g. it closes the thread). Therefore it cannot record the fact that it entered AccECN mode for both half-connections. Indeed, it cannot even remember whether it negotiated the use of classic ECN [RFC3168].

Nonetheless, such a server can determine that it negotiated AccECN as follows. If a TCP server using SYN Cookies supports AccECN and if the first ACK it receives contains an ACE field with the value 0b110 or 0b111, it can assume that:

- o the TCP client must have requested AccECN support on the SYN
- o it (the server) must have confirmed that it supported AccECN

Therefore the server can switch itself into AccECN mode, and continue as if it had never forgotten that it switched itself into AccECN mode earlier.

##### 4.2. Compatibility with Other TCP Options and Experiments

AccECN is compatible (at least on paper) with the most commonly used TCP options: MSS, time-stamp, window scaling, SACK and TCP-AO. It is also compatible with the recent promising experimental TCP options TCP Fast Open (TFO [RFC7413]) and Multipath TCP (MPTCP [RFC6824]). AccECN is friendly to all these protocols, because space for TCP options is particularly scarce on the SYN, where AccECN consumes zero additional header space.

When option space is under pressure from other options, Section 3.2.5 provides guidance on how important it is to send an AccECN Option and whether it needs to be a full-length option.

##### 4.3. Compatibility with Feedback Integrity Mechanisms

The ECN Nonce [RFC3540] is an experimental IETF specification intended to allow a sender to test whether ECN CE markings (or losses) introduced in one network 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 AcceECN 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 AcceECN 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. AcceECN is compatible with any of these approaches:

- o The Data Sender can test the integrity of the receiver's ECN (or loss) feedback by occasionally setting the IP-ECN field to a value normally only set by the network (and/or deliberately leaving a sequence number gap). Then it can test whether the Data Receiver's feedback faithfully reports what it expects [I-D.moncaster-tcpm-rcv-cheat]. Unlike the ECN Nonce, this approach does not waste the ECT(1) codepoint in the IP header, it does not require standardisation and it does not rely on misbehaving receivers volunteering to reveal feedback information that allows them to be detected. However, setting the CE mark by the sender might conceal actual congestion feedback from the network and should therefore only be done sparsely.
- o Networks generate congestion signals when they are becoming congested, so 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 AcceECN. Without AcceECN, the ConEx behaviour of a Data Sender would have to be more conservative than would be necessary if it had the accurate feedback of AcceECN.

- o The TCP authentication option (TCP-AO [RFC5925]) can be used to detect any tampering with AcceCN feedback between the Data Receiver and the Data Sender (whether malicious or accidental). The AcceCN fields are immutable end-to-end, so they are amenable to TCP-AO protection, which covers TCP options by default. However, TCP-AO is often too brittle to use on many end-to-end paths, where middleboxes can make verification fail in their attempts to improve performance or security, e.g. by resegmentation or shifting the sequence space.

## 5. Protocol Properties

This section is informative not normative. It describes how well the protocol satisfies the agreed requirements for a more accurate ECN feedback protocol [RFC7560].

**Accuracy:** From each ACK, the Data Sender can infer the number of new CE marked segments since the previous ACK. This provides better accuracy on CE feedback than classic ECN. In addition if the AcceCN Option is present (not blocked by the network path) the number of bytes marked with CE, ECT(1) and ECT(0) are provided.

**Overhead:** The AcceCN scheme is divided into two parts. The essential part reuses the 3 flags already assigned to ECN in the IP header. The supplementary part adds an additional TCP option consuming up to 11 bytes. However, no TCP option is consumed in the SYN.

**Ordering:** The order in which marks arrive at the Data Receiver is preserved in AcceCN feedback, because the Data Receiver is expected to send an ACK immediately whenever a different mark arrives.

**Timeliness:** While the same ECN markings are arriving continually at the Data Receiver, it can defer ACKs as TCP does normally, but it will immediately send an ACK as soon as a different ECN marking arrives.

**Timeliness vs Overhead:** Change-Triggered ACKs are intended to enable latency-sensitive uses of ECN feedback by capturing the timing of transitions but not wasting resources while the state of the signalling system is stable. The receiver can control how frequently it sends the AcceCN TCP Option and therefore it can control the overhead induced by AcceCN.

**Resilience:** All information is provided based on counters. Therefore if ACKs are lost, the counters on the first ACK

following the losses allows the Data Sender to immediately recover the number of the ECN markings that it missed.

**Resilience against Bias:** Because feedback is based on repetition of counters, random losses do not remove any information, they only delay it. Therefore, even though some ACKs are change-triggered, random losses will not alter the proportions of the different ECN markings in the feedback.

**Resilience vs Overhead:** If space is limited in some segments (e.g. because more option are need on some segments, such as the SACK option after loss), the Data Receiver can send AccECN Options less frequently or truncate fields that have not changed, usually down to as little as 5 bytes. However, it has to send a full-sized AccECN Option at least three times per RTT, which the Data Sender can rely on as a regular beacon or checkpoint.

**Resilience vs Timeliness and Ordering:** Ordering information and the timing of transitions cannot be communicated in three cases: i) during ACK loss; ii) if something on the path strips the AccECN Option; or iii) if the Data Receiver is unable to support Change-Triggered ACKs.

**Complexity:** An AccECN implementation solely involves simple counter increments, some modulo arithmetic to communicate the least significant bits and allow for wrap, and some heuristics for safety against fields cycling due to prolonged periods of ACK loss. Each host needs to maintain eight additional counters. The hosts have to apply some additional tests to detect tampering by middleboxes, but in general the protocol is simple to understand, simple to implement and requires few cycles per packet to execute.

**Integrity:** AccECN is compatible with at least three approaches that can assure the integrity of ECN feedback. If the AccECN Option is stripped the resolution of the feedback is degraded, but the integrity of this degraded feedback can still be assured.

**Backward Compatibility:** If only one endpoint supports the AccECN scheme, it will fall-back to the most advanced ECN feedback scheme supported by the other end.

**Backward Compatibility:** If the AccECN Option is stripped by a middlebox, AccECN still provides basic congestion feedback in the ACE field. Further, AccECN can be used to detect mangling of the IP ECN field; mangling of the TCP ECN flags; blocking of ECT-marked segments; and blocking of segments carrying the AccECN Option. It can detect these conditions during TCP's 3WSH so that

it can fall back to operation without ECN and/or operation without the AccECN Option.

Forward Compatibility: The behaviour of endpoints and middleboxes is carefully defined for all reserved or currently unused codepoints in the scheme, to ensure that any blocking of anomalous values is always at least under reversible policy control.

## 6. IANA Considerations

This document defines a new TCP option for AccECN, assigned a value of TBD1 (decimal) from the TCP option space. This value is defined as:

Kind	Length	Meaning	Reference
TBD1	N	Accurate ECN (AccECN)	RFC XXXX

[TO BE REMOVED: This registration should take place at the following location: <http://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml#tcp-parameters-1>]

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

## 7. Security Considerations

If ever the supplementary part of AccECN based on the new AccECN TCP Option is unusable (due for example to middlebox interference) the essential part of AccECN's congestion feedback offers only limited resilience to long runs of ACK loss (see Section 3.2.2). These problems are unlikely to be due to malicious intervention (because if an attacker could strip a TCP option or discard a long run of ACKs it could wreak other arbitrary havoc). However, it would be of concern if AccECN's resilience could be indirectly compromised during a flooding attack. AccECN is still considered safe though, because if the option is not presented, the AccECN Data Sender is then required to switch to more conservative assumptions about wrap of congestion indication counters (see Section 3.2.2 and Appendix A.2).

Section 4.1 describes how a TCP server can negotiate AccECN and use the SYN cookie method for mitigating SYN flooding attacks.

There is concern that ECN markings could be altered or suppressed, particularly because a misbehaving Data Receiver could increase its own throughput at the expense of others. Given the experimental ECN nonce is now probably undeployable, AcceCN has been generalised for other possible uses of the ECT(1) codepoint to avoid obsolescence of the codepoint even if the nonce mechanism is obsoleted. AcceCN is compatible with the three other schemes known to assure the integrity of ECN feedback (see Section 4.3 for details). If the AcceCN Option is stripped by an incorrectly implemented middlebox, the resolution of the feedback will be degraded, but the integrity of this degraded information can still be assured.

The AcceCN protocol is not believed to introduce any new privacy concerns, because it merely counts and feeds back signals at the transport layer that had already been visible at the IP layer.

## 8. Acknowledgements

We want to thank Koen De Schepper, Praveen Balasubramanian and Michael Welzl for their input and discussion. The idea of using the three ECN-related TCP flags as one field for more accurate TCP-ECN feedback was first introduced in the re-ECN protocol that was the ancestor of ConEx.

Bob Briscoe was part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700) and through the Trilogy 2 project (ICT-317756). The views expressed here are solely those of the authors.

## 9. Comments Solicited

Comments and questions are encouraged and very welcome. They can be addressed to the IETF TCP maintenance and minor modifications working group mailing list <tcpm@ietf.org>, and/or to the authors.

## 10. References

### 10.1. Normative References

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

- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<http://www.rfc-editor.org/info/rfc3168>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<http://www.rfc-editor.org/info/rfc5681>>.
- [RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, DOI 10.17487/RFC6994, August 2013, <<http://www.rfc-editor.org/info/rfc6994>>.

## 10.2. Informative References

- [I-D.bensley-tcpm-dctcp] Bensley, S., Eggert, L., Thaler, D., Balasubramanian, P., and G. Judd, "Microsoft's Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters", draft-bensley-tcpm-dctcp-05 (work in progress), July 2015.
- [I-D.ietf-conex-abstract-mech] Mathis, M. and B. Briscoe, "Congestion Exposure (ConEx) Concepts, Abstract Mechanism and Requirements", draft-ietf-conex-abstract-mech-13 (work in progress), October 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-03 (work in progress), July 2014.
- [RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, DOI 10.17487/RFC3540, June 2003, <<http://www.rfc-editor.org/info/rfc3540>>.
- [RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <<http://www.rfc-editor.org/info/rfc4987>>.

- [RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", RFC 5562, DOI 10.17487/RFC5562, June 2009, <<http://www.rfc-editor.org/info/rfc5562>>.
- [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <<http://www.rfc-editor.org/info/rfc5925>>.
- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <<http://www.rfc-editor.org/info/rfc6824>>.
- [RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <<http://www.rfc-editor.org/info/rfc7413>>.
- [RFC7560] Kuehlewind, M., Ed., Scheffenegger, R., and B. Briscoe, "Problem Statement and Requirements for Increased Accuracy in Explicit Congestion Notification (ECN) Feedback", RFC 7560, DOI 10.17487/RFC7560, August 2015, <<http://www.rfc-editor.org/info/rfc7560>>.

## Appendix A. Example Algorithms

This appendix is informative, not normative. It gives example algorithms that would satisfy the normative requirements of the AcceCN protocol. However, implementers are free to choose other ways to implement the requirements.

### A.1. Example Algorithm to Encode/Decode the AcceCN Option

The example algorithms below show how a Data Receiver in AcceCN mode could encode its CE byte counter `r.ceb` into the ECEB field within the AcceCN TCP Option, and how a Data Sender in AcceCN mode could decode the ECEB field into its byte counter `s.ceb`. The other counters for bytes marked ECT(0) and ECT(1) in the AcceCN Option would be similarly encoded and decoded.

It is assumed that each local byte counter is an unsigned integer greater than 24b (probably 32b), and that the following constant has been assigned:

$$\text{DIVOPT} = 2^{24}$$

Every time a CE marked data segment arrives, the Data Receiver increments its local value of `r.ceb` by the size of the TCP Data. Whenever it sends an ACK with the AcceCN Option, the value it writes into the ECEB field is

$$\text{ECEB} = \text{r.ceb} \% \text{DIVOPT}$$

where `'%'` is the modulo operator.

On the arrival of an AcceCN Option, the Data Sender uses the TCP acknowledgement number and any SACK options to calculate `newlyAckedB`, the amount of new data that the ACK acknowledges in bytes. If `newlyAckedB` is negative it means that a more up to date ACK has already been processed, so this ACK has been superseded and the Data Sender has to ignore the AcceCN Option. Then the Data Sender calculates the minimum difference `d.ceb` between the ECEB field and its local `s.ceb` counter, using modulo arithmetic as follows:

```
if (newlyAckedB >= 0) {
    d.ceb = (ECEB + DIVOPT - (s.ceb % DIVOPT)) % DIVOPT
    s.ceb += d.ceb
}
```

For example, if `s.ceb` is 33,554,433 and ECEB is 1461 (both decimal), then



```
s.ceb % DIVOPT = 1
d.ceb = (1461 + 2^24 - 1) % 2^24
      = 1460
s.ceb = 33,554,433 + 1460
      = 33,555,893
```

#### A.2. Example Algorithm for Safety Against Long Sequences of ACK Loss

The example algorithms below show how a Data Receiver in AcceCN mode could encode its CE packet counter `r.cep` into the ACE field, and how the Data Sender in AcceCN mode could decode the ACE field into its `s.cep` counter. The Data Sender's algorithm includes code to heuristically detect a long enough unbroken string of ACK losses that could have concealed a cycle of the congestion counter in the ACE field of the next ACK to arrive.

Two variants of the algorithm are given: i) a more conservative variant for a Data Sender to use if it detects that the AcceCN Option is not available (see Section 3.2.2 and Section 3.2.4); and ii) a less conservative variant that is feasible when complementary information is available from the AcceCN Option.

##### A.2.1. Safety Algorithm without the AcceCN Option

It is assumed that each local packet counter is a sufficiently sized unsigned integer (probably 32b) and that the following constant has been assigned:

```
DIVACE = 2^3
```

Every time a CE marked packet arrives, the Data Receiver increments its local value of `r.cep` by 1. It repeats the same value of ACE in every subsequent ACK until the next CE marking arrives, where

```
ACE = r.cep % DIVACE.
```

If the Data Sender received an earlier value of the counter that had been delayed due to ACK reordering, it might incorrectly calculate that the ACE field had wrapped. Therefore, on the arrival of every ACK, the Data Sender uses the TCP acknowledgement number and any SACK options to calculate `newlyAckedB`, the amount of new data that the ACK acknowledges. If `newlyAckedB` is negative it means that a more up to date ACK has already been processed, so this ACK has been superseded and the Data Sender has to ignore the AcceCN Option. If `newlyAckedB` is zero, to break the tie the Data Sender could use timestamps (if present) to work out `newlyAckedT`, the amount of new time that the ACK acknowledges. Then the Data Sender calculates the minimum difference

d.cep between the ACE field and its local s.cep counter, using modulo arithmetic as follows:

```
if ((newlyAcedB > 0) || (newlyAcedB == 0 && newlyAcedT > 0))
    d.cep = (ACE + DIVACE - (s.cep % DIVACE)) % DIVACE
```

Section 3.2.2 requires the Data Sender to assume that the ACE field did cycle if it could have cycled under prevailing conditions. The 3-bit ACE field in an arriving ACK could have cycled and become ambiguous to the Data Sender if a row of ACKs goes missing that covers a stream of data long enough to contain 8 or more CE marks. We use the word 'missing' rather than 'lost', because some or all the missing ACKs might arrive eventually, but out of order. Even if some of the lost ACKs are piggy-backed on data (i.e. not pure ACKs) retransmissions will not repair the lost AcceECN information, because AcceECN requires retransmissions to carry the latest AcceECN counters, not the original ones.

The phrase 'under prevailing conditions' allows the Data Sender to take account of the prevailing size of data segments and the prevailing CE marking rate just before the sequence of ACK losses. However, we shall start with the simplest algorithm, which assumes segments are all full-sized and ultra-conservatively it assumes that ECN marking was 100% on the forward path when ACKs on the reverse path started to all be dropped. Specifically, if newlyAcedB is the amount of data that an ACK acknowledges since the previous ACK, then the Data Sender could assume that this acknowledges newlyAcedPkt full-sized segments, where newlyAcedPkt = newlyAcedB/MSS. Then it could assume that the ACE field incremented by

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

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

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

The simple algorithm for dSafer.cep above requires no monitoring of prevailing conditions and it would still be safe if, for example, segments were on average at least 5% of full-sized as long as ECN marking was 5% or less. Assuming it was used, the Data Sender would increment its packet counter as follows:

```
s.cep += dSafer.cep
```

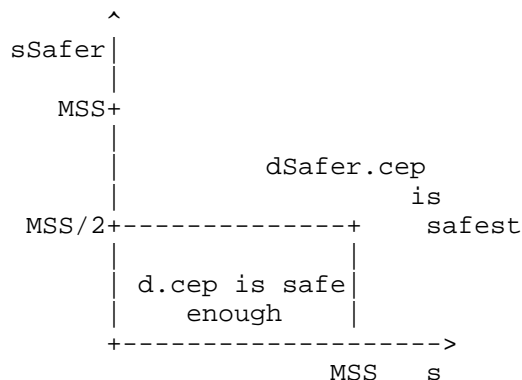
If missing acknowledgement numbers arrive later (due to reordering), Section 3.2.2 says "the Data Sender MAY attempt to neutralise the effect of any action it took based on a conservative assumption that it later found to be incorrect". To do this, the Data Sender would have to store the values of all 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.2.2. Safety Algorithm with the AcceCN Option

When the AcceCN Option is available on the ACKs before and after the possible sequence of ACK losses, if the Data Sender only needs CE-marked bytes, it will have sufficient information in the AcceCN Option without needing to process the ACE field. However, if for some reason it needs CE-marked packets, if dSafer.cep is different from d.cep, it can calculate the average marked segment size that each implies to determine whether d.cep is likely to be a safe enough estimate. Specifically, it could use the following algorithm, where d.ceb is the amount of newly CE-marked bytes (see Appendix A.1):

```
SAFETY_FACTOR = 2
if (dSafer.cep > d.cep) {
    s = d.ceb/d.cep
    if (s <= MSS) {
        sSafer = d.ceb/dSafer.cep
        if (sSafer < MSS/SAFETY_FACTOR)
            dSafer.cep = d.cep      % d.cep is a safe enough estimate
    } % else
        % No need for else; dSafer.cep is already correct,
        % because d.cep must have been too small
}
```

The chart below shows when the above algorithm will consider d.cep can replace dSafer.cep as a safe enough estimate of the number of CE-marked packets:



The following examples give the reasoning behind the algorithm, assuming  $MSS=1,460$  [B]:

- o if  $d.cep=0$ ,  $dSafer.cep=8$  and  $d.ceb=1,460$ , then  $s=infinity$  and  $sSafer=182.5$ .  
Therefore even though the average size of 8 data segments is unlikely to have been as small as  $MSS/8$ ,  $d.cep$  cannot have been correct, because it would imply an average segment size greater than the  $MSS$ .
- o if  $d.cep=2$ ,  $dSafer.cep=10$  and  $d.ceb=1,460$ , then  $s=730$  and  $sSafer=146$ .  
Therefore  $d.cep$  is safe enough, because the average size of 10 data segments is unlikely to have been as small as  $MSS/10$ .
- o if  $d.cep=7$ ,  $dSafer.cep=15$  and  $d.ceb=10,200$ , then  $s=1,457$  and  $sSafer=680$ .  
Therefore  $d.cep$  is safe enough, because the average data segment size is more likely to have been just less than one  $MSS$ , rather than below  $MSS/2$ .

If pure ACKs were allowed to be ECN-capable, missing ACKs would be far less likely. However, because [RFC3168] currently precludes this, the above algorithm assumes that pure ACKs are not ECN-capable.

#### A.3. Example Algorithm to Estimate Marked Bytes from Marked Packets

If the AccECN Option is not available, the Data Sender can only decode CE-marking from the ACE field in packets. Every time an ACK arrives, to convert this into an estimate of CE-marked bytes, it needs an average of the segment size,  $s_{ave}$ . Then it can add or subtract  $s_{ave}$  from the value of  $d.ceb$  as the value of  $d.cep$  increments or decrements.

To calculate `s_ave`, it could keep a record of the byte numbers of all the boundaries between packets in flight (including control packets), and recalculate `s_ave` on every ACK. However it would be simpler to merely maintain a counter `packets_in_flight` for the number of packets in flight (including control packets), which it could update once per RTT. Either way, it would estimate `s_ave` as:

$$s\_ave \sim \text{flightsize} / \text{packets\_in\_flight},$$

where `flightsize` is the variable that TCP already maintains for the number of bytes in flight. To avoid floating point arithmetic, it could right-bit-shift by `lg(packets_in_flight)`, where `lg()` means log base 2.

An alternative would be to maintain an exponentially weighted moving average (EWMA) of the segment size:

$$s\_ave = a * s + (1-a) * s\_ave,$$

where `a` is the decay constant for the EWMA. However, then it is necessary to choose a good value for this constant, which ought to depend on the number of packets in flight. Also the decay constant needs to be power of two to avoid floating point arithmetic.

#### A.4. Example Algorithm to Beacon AccECN Options

Section 3.2.5 requires a Data Receiver to beacon a full-length AccECN Option at least 3 times per RTT. This could be implemented by maintaining a variable to store the number of ACKs (pure and data ACKs) since a full AccECN Option was last sent and another for the approximate number of ACKs sent in the last round trip time:

```
if (acks_since_full_last_sent > acks_in_round / BEACON_FREQ)
    send_full_AccECN_Option()
```

For optimised integer arithmetic, `BEACON_FREQ = 4` could be used, rather than 3, so that the division could be implemented as an integer right bit-shift by `lg(BEACON_FREQ)`.

In certain operating systems, it might be too complex to maintain `acks_in_round`. In others it might be possible by tagging each data segment in the retransmit buffer with the number of ACKs sent at the point that segment was sent. This would not work well if the Data Receiver was not sending data itself, in which case it might be necessary to beacon based on time instead, as follows:

```
if (time_now > time_last_option_sent + RTT / BEACON_FREQ)
    send_full_AccECN_Option()
```

However, this time-based approach does not work well when all the ACKs are sent early in each round trip, as is the case during slow-start.

{ToDo: A simple and robust beaconing algorithm for all circumstances is still work-in-progress.}

#### A.5. Example Algorithm to Count Not-ECT Bytes

A Data Sender in AccECN mode can infer the amount of TCP payload data arriving at the receiver marked Not-ECT from the difference between the amount of newly ACKed data and the sum of the bytes with the other three markings, d.ceb, d.e0b and d.elb. Note that, because r.e0b is initialised to 1 and the other two counters are initialised to 0, the initial sum will be 1, which matches the initial offset of the TCP sequence number on completion of the 3WHS.

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

However, such precision is unlikely to be necessary. The only known use of a count of Not-ECT marked bytes is to test whether equipment on the path is clearing the ECN field (perhaps due to an out-dated attempt to clear, or bleach, what used to be the ToS field). To detect bleaching it will be sufficient to detect whether nearly all bytes arrive marked as Not-ECT. Therefore there should be no need to keep track of the details of retransmissions.

#### 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:

Feedback all four ECN codepoints on the SYN/ACK: The last two negotiation combinations in Table 2 could also be used to indicate AccECN support and to feedback that the arriving SYN was ECT(0) or ECT(1). This could be used to probe the client to server path for incorrect forwarding of the ECN field [I-D.kuehlewind-tcpm-ecn-fallback]. Note, however, that 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.

Feedback all four ECN codepoints on the First ACK: To probe the server to client path for incorrect ECN forwarding, it could be useful to have four feedback states on the first ACK from the TCP client. This could be achieved by assigning four combinations of the ECN flags in the main TCP header, and only initialising the ACE field on subsequent segments.

Empty AceECN Option: It might be useful to allow an empty (Length=2) AceECN Option on the SYN/ACK and first ACK. Then if a host had to omit the option because there was insufficient space for a larger option, it would not give the impression to the other end that a middlebox had stripped the option.

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

1. Currently it is specified that the receiver 'SHOULD' use Change-Triggered ACKs. It is controversial whether this ought to be a 'MUST' instead. A 'SHOULD' would leave the Data Sender uncertain whether it can rely on the timing and ordering information in ACKs. If the sender guesses wrongly, it will probably introduce at least 1RTT of delay before it can use this timing information. Ironically it will most likely be wanting this information to reduce ramp-up delay. A 'MUST' could make it hard to implement AceECN in offload hardware. However, it is not known whether AceECN would be hard to implement in such hardware even with a 'SHOULD' here. For instance, was it hard to offload DCTCP to hardware because of change-triggered ACKs, or was this just one of many reasons? The choice between MUST and SHOULD here is critical. Before that choice is made, a clear use-case for certainty of timing and ordering information is needed, plus well-informed discussion about hardware offload constraints.
2. There is possibly a concern that a receiver could deliberately omit the AceECN Option pretending that it had been stripped by a middlebox. No known way can yet be contrived to take advantage of this downgrade attack, but it is mentioned here in case someone else can contrive one.
3. The s.cep counter might increase even if the s.ceb counter does not (e.g. due to a CE-marked control packet). The sender's response to such a situation is considered out of scope, because this ought to be dealt with in whatever future specification allows ECN-capable control packets. However, it is possible that the situation might arise even if the sender has not sent ECN-

capable control packets, in which case, this draft might need to give some advice on how the sender should respond.

#### 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 04 to 05::

- \* Corrected ambiguity between Classic ECN and Classic ECN feedback throughout
- \* Changed MUST to SHOULD send AcceECN option on SYN/ACK last ACK of 3WHS and first data segment from client, to allow for cached knowledge of option traversal problems.
- \* Removed duplication of normative language about sending a full-length option in the sections on "The AcceECN Option" and "Usage of the AcceECN Option", and mutually cross referenced.
- \* Acknowledged Koen De Schepper and Praveen Balasubramanian
- \* Noted in Appendix that algo to beacon a full-length option is work-in-progress
- \* Editorial corrections and clarifications throughout

#### Authors' Addresses

Bob Briscoe  
Simula Research Laboratory  
  
EMail: [ietf@bobbriscoe.net](mailto:ietf@bobbriscoe.net)  
URI: <http://bobbriscoe.net/>

Mirja Kuehlewind  
ETH Zurich  
Gloriastrasse 35  
Zurich 8092  
Switzerland  
  
EMail: [mirja.kuehlewind@tik.ee.ethz.ch](mailto:mirja.kuehlewind@tik.ee.ethz.ch)



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

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

TCP Maintenance and Minor Extensions  
Internet-Draft  
Intended status: Experimental  
Expires: January 4, 2015

T. Moncaster, Ed.  
University of Cambridge  
B. Briscoe  
A. Jacquet  
BT  
July 03, 2014

A TCP Test to Allow Senders to Identify Receiver Non-Compliance  
draft-moncaster-tcpm-rcv-cheat-03

Abstract

The TCP protocol relies on receivers sending accurate and timely feedback to the sender. Currently the sender has no means to verify that a receiver is correctly sending this feedback according to the protocol. A receiver that is non-compliant has the potential to disrupt a sender's resource allocation, increasing its transmission rate on that connection which in turn could adversely affect the network itself. This document presents a two stage test process that can be used to identify whether a receiver is non-compliant. The tests enshrine the principle that one shouldn't attribute to malice that which may be accidental. The first stage test causes minimum impact to the receiver but raises a suspicion of non-compliance. The second stage test can then be used to verify that the receiver is non-compliant. This specification does not modify the core TCP protocol - the tests can either be implemented as a test suite or as a stand-alone test through a simple modification to the sender implementation.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 4, 2015.

## Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Requirements notation . . . . .	5
3. The Problems . . . . .	5
3.1. Concealing Lost Segments . . . . .	6
3.2. Optimistic Acknowledgements . . . . .	7
4. Requirements for a robust solution . . . . .	9
5. Existing Proposals . . . . .	10
5.1. Randomly Skipped Segments . . . . .	10
5.2. The ECN nonce . . . . .	10
5.3. A transport layer nonce . . . . .	11
6. The Test for Receiver Non-compliance . . . . .	12
6.1. Solution Overview . . . . .	12
6.2. Probabilistic Testing . . . . .	12
6.2.1. Performing the Probabilistic Test . . . . .	13
6.2.2. Assessing the Probabilistic Test . . . . .	15
6.2.3. RTT Measurement Considerations . . . . .	15
6.2.4. Negative Impacts of the Test . . . . .	17
6.2.5. Protocol Details for the Probabilistic Test . . . . .	18
6.3. Deterministic Testing . . . . .	19
6.3.1. Performing the Deterministic Test . . . . .	20
6.3.2. Assessing the Deterministic Test . . . . .	20
6.3.3. Protocol Details for the Deterministic Test . . . . .	20
6.4. Responding to Non-Compliance . . . . .	21
6.5. Possible Interactions With Other TCP Features . . . . .	21
6.5.1. TCP Secure . . . . .	22
6.5.2. Nagle Algorithm . . . . .	22
6.5.3. Delayed Acknowledgements . . . . .	22
6.5.4. Best Effort Transport Service . . . . .	22
6.6. Possible Issues with the Tests . . . . .	22
7. Comparison of the Different Solutions . . . . .	23
8. Alternative Uses of the Test . . . . .	25

9. Evaluating the Experiment . . . . .	25
9.1. Criteria for Success . . . . .	25
9.2. Duration of the Experiment . . . . .	25
9.3. Arguments for Obsoleting the ECN Nonce . . . . .	25
10. IANA Considerations . . . . .	26
11. Security Considerations . . . . .	26
12. Conclusions . . . . .	27
13. Acknowledgements . . . . .	28
14. Comments Solicited . . . . .	28
15. References . . . . .	29
15.1. Normative References . . . . .	29
15.2. Informative References . . . . .	29
Appendix A. Changes from previous drafts (to be removed by the RFC Editor) . . . . .	30
Authors' Addresses . . . . .	31

## 1. Introduction

This document details an experimental test designed to allow a TCP sender to identify when a receiver is misbehaving or is non-compliant. It uses the standard wire protocol and protocol semantics of basic TCP [RFC0793] without modification. The hope is that if the experiment proves successful then we will be able to obsolete the experimental TCP nonce [RFC3540], hence freeing up valuable codepoints in both the IPv4 header and the TCP header.

When any network resource (e.g. a link) becomes congested, the congestion control protocol [RFC5681] within TCP/IP expects all receivers to correctly feed back congestion information and it expects each sender to respond by backing off its rate in response to this information. This relies on the voluntary compliance of all senders and all receivers.

Over recent years the Internet has become increasingly adversarial. Self-interested or malicious parties may produce non-compliant protocol implementations if it is to their advantage, or to the disadvantage of their chosen victims. Enforcing congestion control when trust can not be taken for granted is extremely hard within the current Internet architecture. This specification deals with one specific case: where a TCP sender is TCP compliant and wants to ensure its receivers are compliant as well.

Simple attacks have been published showing that TCP receivers can manipulate feedback to fool TCP senders into massively exceeding the compliant rate [Savage]. Such receivers might want to make senders unwittingly launch a denial of service attack on other flows sharing part of the path between them [Sherwood]. But a more likely motivation is simple self-interest---a receiver can improve its own

download speed with the sender acting as an unwitting accomplice. [Savage] quotes results that show this attack can reduce the time taken to download an HTTP file over a real network by half, even with a relatively cautious optimistic acknowledgment strategy.

There is currently no evidence that any TCP implementations are exploiting any of the attacks mentioned above. However this may be simply because there is no widely available test to identify such attacks. This document describes a test process that can identify such non-compliance by receivers should it start to become an issue. The aim of the authors is to provide a test that is safe to implement and that can be recommended by the IETF. The test can be deployed as a separate test suite, or in existing senders, but this document does not mandate that it should be implemented by senders.

The measures in this specification are intended for senders that can be trusted to behave. This scheme can not prevent misbehaving senders from causing congestion collapse of the Internet. However the very existence of a test scheme such as this should act as a disincentive against non-compliant receivers.

Senders do not have to be motivated solely by "the common good" to deploy these changes. It is directly in their own interest for senders serving multiple receivers (e.g. large file servers and certain file-sharing peers) to detect non-compliant receivers. A large server relies in part on network congestion feedback to efficiently apportion its own resources between receivers. If such a large server devotes an excessive fraction of its own resources to non-compliant receivers, it may well hit its own resource limits and have to starve other half-connections even if their network path has spare capacity.

The proposed tests do not require the receiver to have deployed any new or optional protocol features, as any misbehaving receiver could simply circumvent the test by claiming it did not support the optional feature. Instead, the sender emulates network re-ordering and then network loss to test that the receiver reacts as it should according to the basic TCP protocol. It is important that the level of emulated re-ordering that such a test introduces should not adversely impact compliant receivers.

This document specifies a two-stage test in which the sender deliberately re-orders some data segments so as to check if the destination correctly acknowledges out-of-order segments. The first stage test introduces a small reordering which will have a related very minor performance hit. It is not a conclusive test of compliance. However, failing it strongly suggests the receiver is non-compliant. This raises sufficient suspicion to warrant the more

intrusive but conclusive second stage if this non-compliance is going to be sanctioned. The second stage proves beyond doubt whether the receiver is non-compliant but it also requires significant re-ordering, which harms performance. Therefore it should not be used unless a receiver is already strongly suspected of non-compliance (through failing the first stage).

The technique is designed to work with all known variants of TCP, with or without ECN [RFC3168], with or without SACK [RFC2018], and so on. The technique is probably transferable to derivatives of TCP, such as SCTP [RFC2960], but separate specifications will be required for such related transports. The requirements for a robust solution in Section 4 serve as guidelines for these separate specifications.

## 2. Requirements notation

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 [RFC2119].

## 3. The Problems

TCP is widely used as the end-to-end transport in the Internet. TCP utilises a number of mechanisms to avoid congestion [RFC5681] in order to avoid the congestion collapses that plagued the Internet in the mid 1980s. These mechanisms all rely on knowing that data has been received (through acknowledgments of that data) and knowing when congestion has happened (either through knowing that a segment was lost in flight or through being notified of an Explicit Congestion Notification (ECN) [RFC3168]). TCP also uses a flow control mechanism to control the rate at which data is sent [RFC0813]. Both the flow control and congestion avoidance mechanisms utilise a transmission window that limits the number of unacknowledged segments that are allowed to be sent at any given time. In order to work out the size of the transmission window, TCP monitors the average round trip time (RTT) for each flow and the number of unacknowledged segments still in flight.

A strategising receiver can take advantage of the congestion and flow control mechanisms to increase its data throughput. The three known ways in which it can do this are: optimistic acknowledgements, concealing segment losses and dividing acknowledgements into smaller parts. The first two are examined in more detail below and details of the third can be found in [Savage].

### 3.1. Concealing Lost Segments

TCP is designed to view a lost segment as an indication of congestion on the channel. This is because TCP makes the reasonable assumption that packets are most likely to be lost through deliberately being dropped by a congested node rather than through transmission losses or errors.

In order to avoid congestion collapse [RFC3714], whichever TCP connection detects the congestion (through detecting that a packet has been dropped or marked) is expected to respond to it either by reducing its congestion window to 1 segment after a timeout or by halving it on receipt of three duplicate acks (the precise rules are set out in [RFC5681]).

For applications where missing data is not an issue, it is in the interest of a receiver to maximise the data rate it gets from the sender. If it conceals lost segments by falsely generating acknowledgements for them it will not suffer a reduction in data rate. There are a number of ways to make an application loss-insensitive. Some applications such as streaming media are inherently insensitive anyway, as a loss will just be seen as a transient error. TCP is widely used to transmit media files, either audio or video, which are relatively insensitive to data loss (depending on the encoding used). Also senders may be serving data containing redundant parity to allow the application to recreate lost data. A misbehaving receiver can also exploit application layer protocols such as the partial GET in HTTP 1.1 [RFC2616] to recover missing data over a secondary connection.

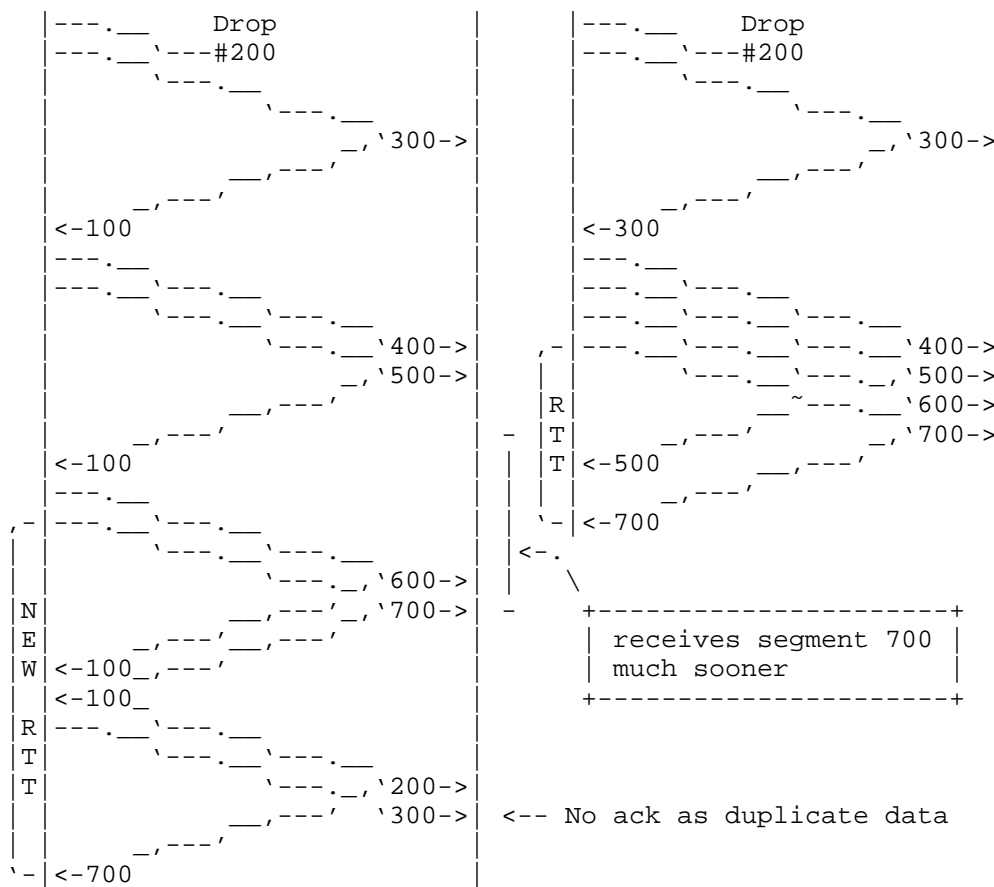


Figure 1: Concealing lost segments

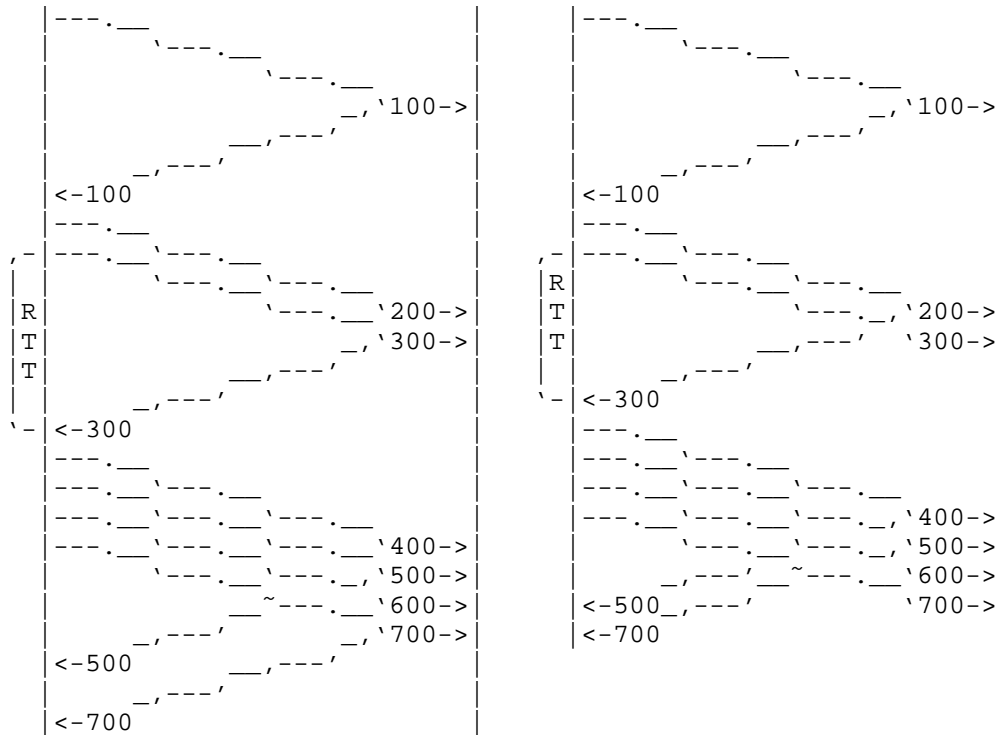
### 3.2. Optimistic Acknowledgements

Optimistic acknowledgements were identified as a possible attack in [Savage]. If a receiver is downloading a file from a server, it is probably in its interest to acquire as high a bandwidth as possible for this. One way of increasing the bandwidth is to encourage the sender to believe the round trip time is shorter than it actually is. This means the sender will open up its transmission window faster and thus will send data faster. Of course any lost segments will also be concealed during this attack.

The receiver can achieve this by sending acknowledgements for data it hasn't actually received yet. As long as the acknowledgement is for a packet that has already been transmitted, the sender will assume the RTT has become shorter. This will cause it to increase its



transmission window more rapidly and thus send more data. Optimistic acknowledgements are particularly damaging since they can also be used to significantly amplify the effect of a denial of service (DoS) attack on a network. This form of attack is explained in more detail in [Sherwood].



The flow on the left acknowledges data only once it is received. The flow on the right acknowledges data before it is received and consequently the apparent RTT is reduced.

Figure 2: Optimistic acknowledgements

In 2005 US-CERT (the United States Computer Emergency Readiness Team) issued a vulnerability notice [VU102014] specifically addressed to 80 major network equipment manufacturers and vendors who could be affected if someone maliciously exploited optimistic acknowledgements to cause a denial of service. This highlights the potential severity of such an attack were one to be launched. It should be noted however that the primary motivation for using optimistic acknowledgement is likely to be the performance gain it gives rather than the possible negative impact on the network. Application writers may well produce "Download Accelerators" that use optimistic

acknowledgements to achieve the performance increase rather than the current parallel connection approach most use. Users of such software would be effectively innocent parties to the potential harm that such a non-compliant TCP could cause.

#### 4. Requirements for a robust solution

Since the above problems come about through the inherent behaviour of the TCP protocol, there is no gain in introducing a new protocol as misbehaving receivers can claim to only support the old protocol. The best approach is to provide a mechanism within the existing protocol to test whether a receiver is compliant. The following requirements should be met by any such test in TCP and are likely to be applicable for similar tests in other transport protocols:

1. The compliance test must not adversely affect the existing congestion control and avoidance algorithms since one of the primary aims of any compliance test is to reinforce the integrity of congestion control.
2. Any test should utilise existing features of the TCP protocol. If it can be implemented without altering the existing protocol then implementation and deployment are easier.
3. The receiver should not play an active role in the process. It is much more secure to have a check for compliance that only requires the receiver to behave as it should anyway.
4. It should not require the use of any negotiable TCP options. Since the use of such options is by definition optional, any misbehaving receiver could just choose not to use the appropriate option.
5. If this is a periodic test, the receiver must not be aware that it is being tested for compliance. If a misbehaving receiver can tell that it is being tested (by identifying the pattern of testing) it can choose to respond compliantly only whilst it is being tested. If the test is always performed this clearly doesn't apply.
6. If the sender actively sanctions any non-compliance it identifies, it should be certain of the receiver's non-compliance before taking action against it. Any false positives might lead to inefficient use of network resources and could damage end-user confidence in the network.
7. The testing should not significantly reduce the performance of an innocent receiver.

## 5. Existing Proposals

### 5.1. Randomly Skipped Segments

[Sherwood] suggests a simple approach to test a receiver's compliance. The test involves randomly dropping segments at the sender before they are transmitted. All TCP "flavours" require that a receiver should generate duplicate acknowledgements for all subsequent segments until a missing segment is received. This system requires that SACK be enabled so the sender can reliably tell that the duplicate acknowledgements are generated by the segment that is meant to be missing and are not concealing other congestion. Once the first duplicate acknowledgement arrives, the missing segment can then be "re-transmitted". Because this loss has been deliberately introduced, the sender doesn't treat it as a sign of congestion. If a receiver sends an acknowledgement for a segment that was sent after the gap, it proves it is misbehaving or that its TCP is completely non-compliant. It can then be sanctioned. As soon as the first duplicate acknowledgement is received the missing segment is "re-transmitted". This will introduce a 1 RTT delay for some segments which could adversely affect some low-latency applications.

This scheme does work perfectly well in principle and does allow the sender to clearly identify misbehaviour. However it fails to meet requirement 4 in Section 4 above since it requires SACK to be used. If SACK were not used then it would fail to meet requirement 1 as it would be impossible to differentiate between the loss introduced on purpose and any additional loss introduced by the network.

It might be possible to incentivise the use of SACK by receivers by stating that senders are entitled to discriminate against receivers that don't support it. Given that SACK is now widely implemented across the Internet this might be a feasible, but controversial, deployment strategy. However the solution in Section 6 builds on Sherwood's scheme but avoids the need for SACK.

### 5.2. The ECN nonce

The authors of the ECN scheme [RFC3168] identified the failure to echo ECN marks as a potential attack on ECN. The ECN nonce was proposed as a possible solution to this in the experimental [RFC3540]. It uses a 1 bit nonce in every IP header. The nonce works by randomly setting the ECN field to ECT(0) or ECT(1). The sender then maintains the least significant bit of the sum of this value and stores the expected sum for each segment boundary. At the receiver end, the same cumulative 1-bit sum is calculated and is echoed back in the NS (nonce sum) flag added to the TCP header. If a packet has been congestion marked then it loses the information of

which ECT codepoint it was carrying. A receiver wishing to conceal the ECN mark will have to guess whether to increment NS or not. Once congestion has been echoed back and the source has started a congestion response the nonce sum in the TCP header is not checked. Once congestion recovery is over the source resets its NS to that of the destination and starts checking again.

On the face of it this solution also fully covers the two problems identified in Section 3. If a receiver conceals a lost segment it has to guess what mark was there and, over several guesses, is very likely to be found out. If a receiver tries to use optimistic acknowledgements it has to guess what nonce was set on all the packets it acknowledges but hasn't received yet. However there are some key weaknesses to this system. Firstly, it assumes that ECN will be widely deployed (not currently true). Secondly, it relies on the receiver honestly declaring support for both ECN and the ECN nonce - a strategising receiver can simply declare it is neither ECN nor ECN nonce capable and thus avoid the nonce. Thirdly, the mechanism is suspended during any congestion response. Comparing it against the requirements in Section 4 above, it is clear that the ECN nonce fails to meet requirements 3 and 4 and arguably fails to meet requirement 2 as [RFC3540] is experimental. The authors do state that any sender that implements the ECN nonce is entitled to discriminate against any receiver that doesn't support it. Given there are currently no implementations of the ECN nonce, discriminating against the overwhelming majority of receivers that don't support it is not a feasible deployment strategy.

### 5.3. A transport layer nonce

One possible solution to the above issues is a multi-bit transport layer nonce. Two versions of this are proposed in [Savage]. The first is the so called "Singular Nonce" where each segment is assigned a unique random number. This value is then echoed back to the receiver with the ack for that segment. The second version is the "Cumulative Nonce" where the nonce is set as before, but the cumulative sum of all nonces is echoed back. Whilst such a system is robust and allows a sender to correctly identify a misbehaving receiver, it has the key drawback that it requires either the creation of a new TCP option to carry the nonce and nonce reply or it requires the TCP header to be extended to include both these fields.

This proposal clearly breaches several of the requirements listed in Section 4. It breaches requirement 2 in that it needs a completely new TCP option or a change to the TCP header. It breaches requirement 3 because it needs the receiver to actively echo the nonce (as does the ECN nonce scheme) and if it uses a TCP option it

breaches requirement 4. On the face of it there is no obvious route by which this sort of system can be widely implemented.

## 6. The Test for Receiver Non-compliance

### 6.1. Solution Overview

The ideal solution to the above problems should fully meet the requirements set out in Section 4. The most important of these is that the solution should leverage existing TCP behaviours rather than mandating new behaviours and options. The proposed solution utilises TCP's receiver behaviour on detecting missing data. To test a receiver the sender delays a segment during transmission by  $D$  segments. There is a trade off because increasing  $D$  increases the probability of detecting non-compliance but also increases the probability of masking a congestion event during the test. The completely safe strategy for the sender would be to reduce its rate pessimistically as if there were congestion during the test however this will impact the performance of its receivers, thus breaching requirement 7. To overcome this dilemma, the test consists of two stages. In the first stage, the sender uses small displacements without the pessimistic congestion response to determine which receivers appear to be non-compliant. The sender can then prove the non-compliance of these receivers by subjecting them to a deterministic test. This test uses a longer displacement but given the receiver is already under suspicion, it can risk harming performance by pessimistically reducing its rate as if the segment it held back was really lost by the network. The tests can either be implemented as part of a test suite or as a stand-alone modification to the TCP sender implementation. References to the TCP sender in the rest of this document should be taken to include either type of implementation.

### 6.2. Probabilistic Testing

The first requirement for a sender is to decide when to test a receiver. This document doesn't specify when the test should be performed but the following guidance may be helpful. The simplest option is for a sender to perform the test at frequent random intervals for all its half-connections. There are also some heuristic triggers that might indicate the need for a test. Firstly, if a sender is itself too busy, it would be sensible for it to test all its receivers. Secondly, if the sender has many half-connections that are within a RTT of a congestion response, it would be sensible to test all the half-connections that aren't in a congestion response. Thirdly, the sender could aim to test all its half-connections at least once. Finally it is to be expected that there is a certain degree of existing segment reordering and thus a sender

should be suspicious of any receiver that isn't generating as many duplicate acknowledgements as other receivers. [Piratla] explores how prevalent reordering might be in the Internet though it is unclear whether the figures given are more widely applicable.

Like the skipped segment solution in Section 5.1, the proposed solution depends on the strict requirement that all TCP receivers have to send a duplicate acknowledgement as soon as they receive an out-of-order segment. This acknowledges that some data has been received, however the acknowledgement is for the last in order segment that was received (hence duplicating an acknowledgment already made). SACK extends this behaviour to allow the sender to infer exactly which segments are missing. This leads to a simple statement: if a receiver is behaving compliantly it must respond to an out-of-order packet by generating a duplicate acknowledgement.

Following from the above statement, a sender can test the compliance of a given receiver by simply delaying transmission of a segment by several places. A compliant receiver will respond to this by generating a number of duplicate acknowledgements. The sender would strongly suspect a receiver of non-compliance if it received no duplicate acknowledgements as a result of the test. A misbehaving receiver can only conceal its actions by waiting until the delayed segment arrives and then generating an appropriate stream of duplicate acknowledgements to appear to be honest. This removes any benefits it may be gaining from cheating because it will significantly increase the RTT observed by the sender.

#### 6.2.1. Performing the Probabilistic Test

The actual mechanism for conducting the test is extremely simple. Having decided to conduct a test the sender selects a segment,  $N$ . It then chooses a displacement,  $D$  (in segments) for this segment where strictly  $2 < D < K - 2$  where  $K$  is the current window size. In practice only low values of  $D$  should be chosen to conceal the test among the background reordering and limit the chance of masking congestion.  $D$  SHOULD be 6 or less for an initial test.  $D$  MUST be greater than 2 to allow for the standard fast retransmit threshold of 3 duplicate acknowledgements. If  $K$  is less than 5, the sender should arguably not perform any compliance testing. This is because when the window is so small then non-compliance is not such a significant issue. The exception to this might be when this test is being used for testing new implementations. To conduct the probabilistic test, instead of transmitting segment  $N$ , it transmits  $N+1$ ,  $N+2$ , etc. as shown in the figure below. Once it has transmitted  $N+D$  it can transmit segment  $N$ . The sender needs to record the sequence number,  $N$  as well as the displacement,  $D$ .

[illegible]

During testing, loss of segment L in the range from  $N+1$  to  $N+D$  inclusive will be temporarily masked by the duplicate acknowledgements from the intentional gap that was introduced. In this case the sender's congestion response will be delayed by at most the offset  $D$ . If there is an actual loss during the test then, once the receiver receives segment  $N$ , it will generate an acknowledgement for  $L-1$ . This will lie between  $N$  and  $N+D$ . Thus it is reasonable to treat receipt of any acknowledgement between  $N$  and  $N+D$  inclusive as an indication of congestion and react accordingly. This will also discourage the receiver from sending optimistic acknowledgements in case these prove to lie in the middle of a testing sequence, in which case it will trigger a congestion response by the sender. It also means a dishonest receiver has to wait for a full  $K$  segments after any genuine lost segment to be sure it isn't a test as it will otherwise trigger a congestion response. Delaying by that long will quickly increase the RTT estimate and will soon reduce the transmission rate by as much as if the receiver had reacted honestly to the congestion.

As an additional safety measure, if the sender is performing slow start when it decides to test the receiver, it should change to congestion avoidance. The reason for this is in case there is any congestion that is concealed during the test. If there is congestion, and the sender's window is still increasing exponentially, this might significantly exacerbate the situation. This does mean that any receiver being tested during this period will suffer reduced throughput, but such testing should only be triggered by the sender being overloaded.

#### 6.2.2. Assessing the Probabilistic Test

This approach to testing receiver compliance appears to meet all the requirements set out in Section 4. The most attractive feature is that it enforces equivalence with compliant behaviour. That is to say, a receiver can either honestly report the missing packets or it can suffer a reduced throughput by delaying segments and increasing the RTT. The only significant drawback is that during a test it introduces some delay to the reporting of actual congestion. Given that TCP only reacts once to congestion in each RTT the delay doesn't significantly adversely affect the overall response to severe congestion.

Some receivers may choose to misbehave despite this. These can be quickly identified by looking at their acknowledgements. A receiver that never sends duplicate acknowledgements in response to being tested is likely to be misbehaving. Equally, a receiver that delays transmission of the duplicate acknowledgements until it is sure it is being tested will leave an obvious pattern of acknowledgements that the sender can identify. Because a receiver is unlikely to be able to differentiate this test from actual re-ordering events, the receiver will be forced to behave in the same fashion for any re-ordered packet even in the absence of a test, making it continually appear to have longer RTT.

#### 6.2.3. RTT Measurement Considerations

Clearly, if the sender has re-ordered segment N, it cannot use it to take an accurate RTT measurement. However it is desirable to ensure that, during a test, the sender still measures the RTT of the flow. One of the key aspects of this test is that the only way for an actually dishonest receiver to cheat the test is to delay sending acknowledgements until it is certain a test is happening. If accurate RTTs can be measured during a test, this delay will cause a dishonest receiver to suffer an increase in RTT and thus a reduction in data throughput.



Measurement of the RTT usually depends on receiving an acknowledgement for a segment and measuring the delay between when the segment was sent and when the acknowledgement arrives. The TCP timestamp option is often used to provide accurate RTT measurement but again, this is not going to function correctly during the test phase. During a test therefore, the RTT has to be estimated using the arrival of duplicate acknowledgements. Figure 4 shows how one can measure the RTT in this way, and also demonstrates how this will increase if a dishonest sender chooses to cheat. However it is not sufficient simply to measure a single RTT during the test.

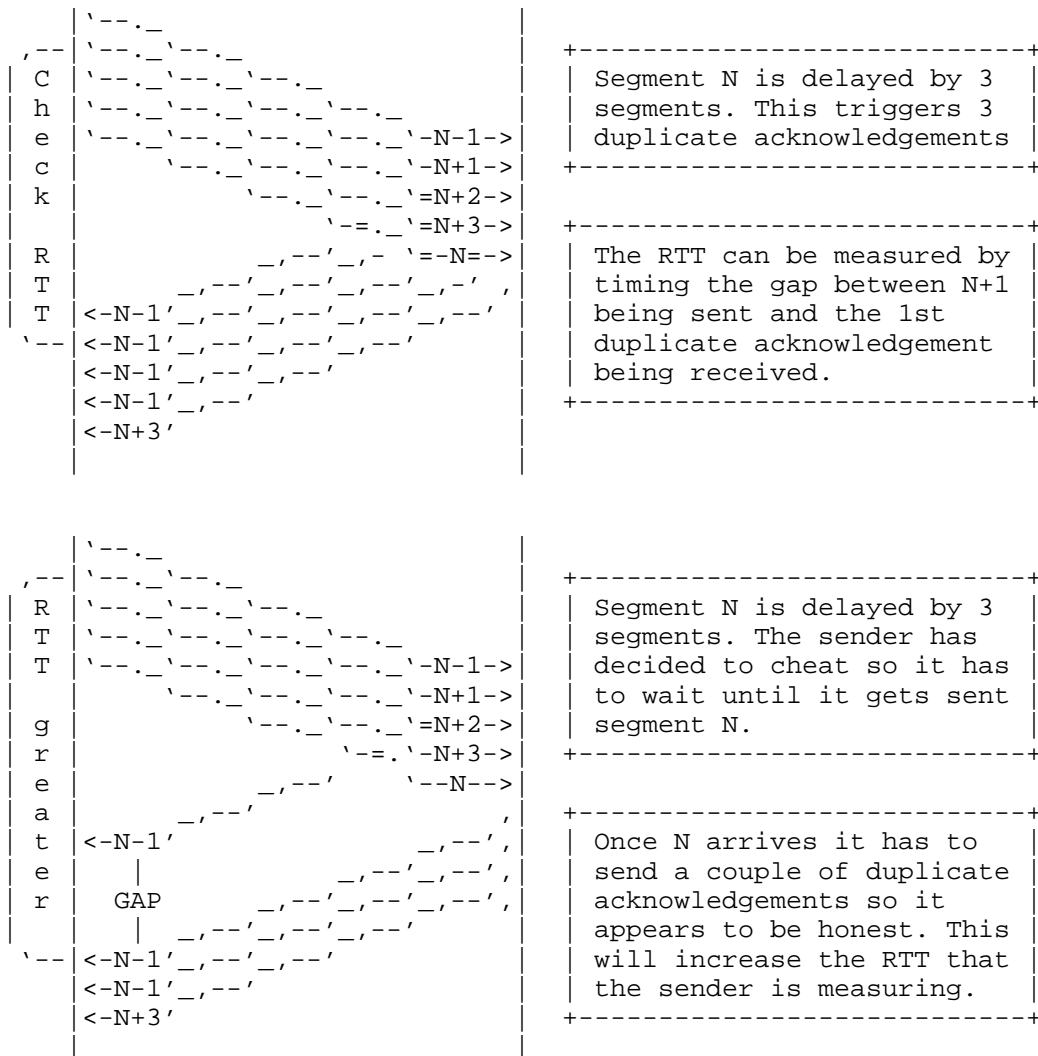


Figure 4: Measuring the RTT during a test

## 6.2.4. Negative Impacts of the Test

It is important to be aware that keeping track of out-of-order data segments uses some memory resources at the receiver. Clearly this test introduces additional re-ordering to the network and consequently will lead to receivers using additional resources. In order to mitigate against this, any sender that implements the test

should only conduct the test at relatively long intervals (of the order of several RTTs).

#### 6.2.5. Protocol Details for the Probabilistic Test

- o Any TCP sender MAY use the probabilistic test periodically and randomly to check the compliance of its receivers. In particular, it would be advantageous for any sender that is heavily loaded to identify if it is being taken advantage of by non-compliant receivers.
- o The decision to test MUST be randomised and MAY be based on: the current load on the sender; whether the receiver is undergoing a congestion response; whether the receiver appears to have different flow characteristics to the others; when the receiver was last tested. The interval between tests SHOULD be relatively long (order of several RTTs).
- o To perform the test, the sender selects a segment N. The transmission of this segment will be delayed by D places. D MUST lie between 2 and K-2 exclusively where K is the current size of the transmit window. D SHOULD lie between 3 and 6 inclusively except in those circumstances when a receiver has failed to respond as expected to an earlier test but the sender chooses not to proceed to the deterministic test. D MUST be generated pseudo-randomly and unpredictably. The actual delay SHOULD be such that the receiver can't distinguish the test segment from the background traffic. If there are less than D segments worth of data in the send buffer then the test SHOULD be omitted.
- o If  $K < 5$ , the sender SHOULD NOT conduct a compliance test.
- o The sequence number N of the delayed segment MUST be recorded by the sender as must the amount of delay D.
- o The senders enters the test phase when it transmits segment N+1 instead of N.
- o The sender MUST NOT use segment N to measure the RTT of the flow. This is because it won't get a true acknowledgement for this segment.
- o The sender SHOULD use segment N+1 to measure the RTT using the first duplicate acknowledgement it receives to calculate the RTT. This is to ensure that a dishonest receiver will suffer from an increased RTT estimate. The sender SHOULD continue checking the RTT throughout the test period.

- o If the sender receives any duplicate acknowledgements during the test phase it MUST check to see if they were generated by the delayed segment (i.e. the acknowledged sequence number must be that of the preceding segment). If they are generated to report the missing segment N the sender SHOULD NOT react as if they are an indication of congestion.
- o If the sender receives an acknowledgement for a segment with a sequence number between N and N+D inclusively it MUST treat this as an indication of congestion and react appropriately.
- o A sender stops being in the test phase when either it receives the acknowledgement for segment N+D or when it has received at least D duplicate acknowledgments, whichever happens sooner.
- o If a sender in the test phase receives D or more duplicate acknowledgements, then it MUST retransmit segment N and react as if there is congestion as specified in [RFC5681]. This is to allow for the possibility that segment N may be lost.
- o If the sender is in the slow start phase it MUST move to congestion avoidance as soon as it begins a test. It MAY choose to return to slow start once the test is completed.
- o If a sender is in the test phase and receives no duplicate acknowledgements from the receiver it MUST treat this as suspicious and SHOULD perform the more rigorous deterministic test set out in Section 6.3.3.
- o If a sender is in the test phase and the next segment to be transmitted has either the FIN or RST bits set, then it must immediately stop the test, and transmit segment N before transmitting the FIN or RST segment.
- o A sender MAY choose to monitor the pattern of acknowledgements generated by a receiver. A dishonest receiver is likely to send a distinctive pattern of duplicate acknowledgments during the test phase. As they are unable to detect whether it is a test or not they are also forced to behave the same in the presence of any segment reordering caused by the network.

### 6.3. Deterministic Testing

If after one or more probabilistic tests the sender deems that a receiver is acting suspiciously, the sender can perform a deterministic test similar to the skipped segment scheme in Section 5.1 above.

### 6.3.1. Performing the Deterministic Test

In order to perform the deterministic test the sender again needs to choose a segment, M to use for testing. This time the sender holds back the segment until the receiver indicates that it is missing. Once the receiver sends a duplicate acknowledgement for segment M-1 then the sender transmits segment M. In the meantime data transmission should proceed as usual. If SACK is not in use, this test clearly increases the delay in reporting of genuine segment losses by up to a RTT. This is because it is only once segment M reaches the receiver that it will be able to acknowledge the later loss. Therefore, unless SACK is in use, the sender MUST pessimistically perform a congestion response following the arrival of 3 duplicate acknowledgements for segment M-1 as mandated in [RFC5681].

### 6.3.2. Assessing the Deterministic Test

A dishonest receiver that is concealing segment losses will establish that this isn't a probabilistic test once the missing segment fails to arrive within the space of 1 congestion window. In order to conceal the loss the receiver will simply carry on acknowledging all subsequent data. The sender can therefore state that if it receives an acknowledgement for a segment with a sequence number greater than M before it has actually sent segment M then the receiver must either be cheating or is very non-compliant.

It is important to be aware that a third party who is able to correctly guess the initial sequence number of a connection might be able to masquerade as a receiver and send acknowledgements on their behalf to make them appear non-compliant or even dishonest. Such an attack can be identified because an honest receiver will also be generating a stream of duplicate acknowledgements until such time as it receives the missing segment.

### 6.3.3. Protocol Details for the Deterministic Test

- o If a sender has reason to suspect that a receiver is reacting in a non-compliant manner to the probabilistic test it SHOULD perform the more thorough deterministic test.
- o To perform the deterministic test the sender MUST select a segment M at random. The sender MUST store this segment in the buffer of unacknowledged data without sending it and MUST record the sequence number.
- o If SACK is not being used, the receiver MUST pessimistically perform a congestion response following the arrival of the first 3

duplicate acknowledgments for segment M-1 as mandated in [RFC5681].

- o If the receiver sends an acknowledgement for a segment that was sent after segment M should have been sent, but before segment M is actually sent, then the receiver has proved its non-compliance. The only possible exception to this is if the receiver is also sending a correct stream of duplicate acknowledgements as this implies that a third party is interfering with the connection.
- o As soon as the first duplicate acknowledgement for segment M-1 arrives, segment M MUST be transmitted. The effective delay, D, of segment M MUST be calculated and stored.
- o If a sender is in the test phase and the next segment to be transmitted has either the FIN or RST bits set, then it must immediately stop the test, and transmit segment N before transmitting the FIN or RST segment.
- o Any subsequent acknowledgement for a segment between M and M+D MUST be treated as an indication of congestion and responded to appropriately as specified in [RFC5681].

#### 6.4. Responding to Non-Compliance

Having identified that a receiver is actually being dishonest, the appropriate response is to terminate the connection with that receiver. If a sender is under severe attack it might also choose to ignore all subsequent requests to connect by that receiver. However this is a risky strategy as it might give an increased incentive to launch an attack against someone by making them appear to be behaving dishonestly. It is also risky in the current network where many users might share quite a small bank of IP addresses assigned dynamically to them by their ISP's DHCP server. A safer alternative to blacklisting a given IP address might be to simply test future connections more rigorously.

#### 6.5. Possible Interactions With Other TCP Features

In order to be safe to deploy, this test must not cause any unforeseen interactions with other existing TCP features. This section looks at some of the possible interactions that might happen and seeks to show that they are not harmful.

#### 6.5.1. TCP Secure

[RFC5961] is a WG Internet Draft that provides a solution to some security issues around the injection of spoofed TCP packets into a TCP connection. The mitigations to these attacks revolve round limiting the acceptable sequence numbers for RST and SYN segments. In order to ensure there is no unforeseen interaction between TCP Secure and this test the test protocol has been specified such that the test will be aborted if a RST segment is sent.

#### 6.5.2. Nagle Algorithm

The Nagle algorithm [RFC0896] allows a TCP sender to buffer data waiting to be sent until such time as it receives an acknowledgement for the previous segment. This means that there is only ever one segment in flight and as such this test should not be performed when the Nagle algorithm is being used.

#### 6.5.3. Delayed Acknowledgements

[RFC5681] allows for the generation of delayed acknowledgements for data segments. However the tests in this document rely on triggering the generation of duplicate acknowledgements. These must be generated for every out of order packet that is received and should be generated immediately the packet is received. Consequently these mechanisms have no effect on the tests set out in this document.

#### 6.5.4. Best Effort Transport Service

The Best Effort Transport Service (BETS) is one operating mode of the Space Communications Protocol Standards (SCPS) [SCPS]. SCPS is a set of communications protocols optimised for extremely high bandwidth-delay product links such as those that exist in space. SCPS-TP (SCPS - Transport Protocol) is based on TCP and is an official TCP option (number 20). The BETS option within SCPS-TP is designed to provide a semi-reliable transport between endpoints. As such it doesn't necessarily ACK data in the same manner as TCP and thus, if this option has been negotiated on a link the tests described above should not be used.

#### 6.6. Possible Issues with the Tests

Earlier in this document we asserted that these tests don't change the TCP protocol. We make this assertion for two reasons. Firstly the protocol can be implemented as a shim that sits between the TCP and IP layers. Secondly the network and receiver are unable to differentiate between a sender that implements these tests and a sender where the IP layer re-orders packets before transmission.

However the tests might have some impact on the debugging of a TCP implementation. It will also have an impact on debugging traces as it creates additional reordering. The authors feel that these effects are sufficiently minor to be safely ignored. If an author of a new TCP implementation wishes to be certain that they won't be affected by the tests during debugging they simply need to ensure that the sender they are connecting to is not undertaking the tests.

A potentially more problematic consequence is the slight increase in packet reordering that this test might introduce. However the degree of reordering introduced in the probabilistic test is strictly limited. This should have minimal impact on the network as a whole although this assertion would benefit from testing by the wider Internet Community.

The final potential problem is that this test relies on the flows being long-running. However this may not be a real issue since for a short running flow none of the attacks described in Section 3 would give the receiver any advantage in a short flow.

## 7. Comparison of the Different Solutions

The following table shows how all the approaches described in this document compare against the requirements set out in Section 4.



Requirement	Rand skip segs	ECN nonce	Transp. nonce	Stage 1 test	Stage 2 test
Congestion Control unaffected	Yes	Yes	Yes	Yes	Yes
Utilise existing features	Yes	No**	No	Yes	Yes
Receiver passive role	Yes	No	No	Yes	Yes
No negotiable TCP options	Yes *	No	No	Yes	Yes
Receiver unaware	Yes	N/A	N/A	Yes	Yes
Certain of non-compliance	Yes	Yes	Yes	strong suspicion	Yes
Innocent rcvr. not adversely affected	No	Yes	Yes	Yes	No

\* Safer when SACK is used

\*\* Currently Experimental RFC with no known available implementation

#### Comparing different solutions against the requirements

The table highlights that the three existing schemes looked at in detail in Section 5 all fail on at least two of these requirements. Whilst this doesn't necessarily make them bad solutions it does mean that they are harder to deploy than the new tests presented in this document. These new tests do have potential issues (see Section 6.6). However, as the table shows, they are minor compared to the problems the nonce-based schemes face, particularly the need for cooperation from the receiver and the use of additional codepoints in the IPv4 and TCP headers.

## 8. Alternative Uses of the Test

Thus far, the two stage test process described in this document has been examined in terms of being a test for compliance by a receiver to the TCP protocol, specifically in terms of the protocol's reaction to segment reordering. The probabilistic test however could also be used for other test purposes. For instance the test can be used to confirm that a receiver has correctly implemented TCP SACK. Because the sender knows exactly which segments have been reordered, it can confirm that the gaps in the data as reported by SACK are indeed correct. The test could also be incorporated as part of a test suite to test the overall compliance of new TCP implementations.

## 9. Evaluating the Experiment

As stated in the introduction, this is an experimental protocol. The main aim of the experiment is to prove that the two tests described in Section 6 provide a robust and safe test for receiver non-compliance. The second aim is to show that the experimental ECN Nonce is no longer needed as these tests provide a more robust defence against receiver non-compliance.

### 9.1. Criteria for Success

The criteria for a successful experiment are very simple.

- o Do the tests accurately identify misbehaving receivers?
- o Are the tests as described in Section 6.2 and Section 6.3 safe?  
By this we mean is the impact of the test such that it causes no harm to other flows and only minimal harm to honest receivers?

### 9.2. Duration of the Experiment

We believe that the experiment should be proved one way or another within a one year period (subject to volunteers agreeing to help with the evaluation). At the end of the experiment if it is shown to be successful we will go back to the IESG to ask for this test to be moved to standards track. At that point, it would be possible to obsolete the experimental ECN Nonce [RFC3540] and recover the codepoints assigned to it.

### 9.3. Arguments for Obsoleting the ECN Nonce

We believe the tests presented in this document provide significantly greater protection against misbehaving TCP receivers than that provided by the ECN Nonce[RFC3540].

1. The ECN Nonce is acting to block the wider use of the two ECT codepoints defined in ECN [RFC3168]. Currently these have to be treated as having identical meanings except in specific controlled circumstances as mandated in [RFC4774] (PCN [RFC6660] is an example of such a use). The authors are aware of a number of research projects to reduce queuing latency or to speed up slow-start that depend on the availability of the ECT(1) codepoint. If the codepoint were freed up, these projects would gain traction and those with promise could be brought to the IETF. Furthermore the nonce is also holding back a flag in the TCP header (the Nonce Sum or NS flag).
2. The ECN Nonce is an experimental standard 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). In the 11 years since it was presented there has been no evidence of any deployment. To the best of our knowledge only two implementations have ever existed. One was that of the original authors and the other was written to test an alternative use of the nonce [Spurious]. Furthermore 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.
3. As explained in Section 7, the ECN Nonce is only a limited solution to the security implications of failing to provide accurate congestion feedback. However some authors may not realise its limitations and may choose to argue that its existence offers them sufficient protection from misbehaving receivers.

#### 10. IANA Considerations

This memo includes no request to IANA.

#### 11. Security Considerations

The two tests described in this document provide a solution to two of the significant security problems that were outlined in [Savage]. Both these attacks could potentially cause major congestion of senders own resources (by making them transmit at too high a rate) and could lead to network congestion collapse through subverting the correct reporting of congestion or by amplifying any DoS attack [Sherwood]. The proposed solution cannot alone prevent misbehaving

senders from causing congestion collapse of the Internet. However, the more widely it is deployed by trustworthy senders, the more these particular attacks would be mitigated through ensuring accurate reporting of segment losses. The more senders that deploy these measures, the less likely it is that a misbehaving receiver will be able to find a sender to fool into causing congestion collapse.

It should be noted that if a third party is able to correctly guess the initial sequence number of a connection, they might be able to masquerade as a receiver and send acknowledgements on their behalf to make them appear dishonest during a deterministic test.

Due to the wording of [RFC5681] a receiver wishing to establish whether a probabilistic test is happening can keep their acknowledgement clock running (thus maintaining transmission rate) by generating pairs of duplicate acknowledgements for segments it received prior to the gap in the data stream caused by the test. This would allow a receiver to subsequently send any additional duplicate acknowledgements that would be necessary to make it appear honest. Such behaviour by a receiver would be readily apparent by examining the pattern of the acknowledgements. Should receivers prove able to exploit this to their advantage, there might be a need to change some of the musts and shoulds laid out in Section 6.2.5.

[Savage] also identified a further attack involving splitting acknowledgements into smaller parts. TCP is designed such that increases in the congestion window are driven by the arrival of a valid acknowledgement. It doesn't matter if this acknowledgement covers all of a transmitted segment or not. This means a receiver that divides all its acknowledgements into two will cause the congestion window to open at twice the rate it would do otherwise. The tests described above can't protect against that attack. However there is a straightforward solution to this - every time the sender transmits a new segment it increments a counter; every acknowledgment it receives decrements that counter; if the counter reaches zero, the sender won't increase its congestion window in response to a new acknowledgement arriving. To comply with this document, senders MUST implement a solution to this problem.

## 12. Conclusions

The issue of mutual trust between TCP senders and receivers is a significant one in the current Internet. This document has introduced a mechanism by which senders can verify that their receivers are compliant with the current TCP protocol. The whole process is robust, lightweight, elegant and efficient. The probabilistic test might delay a congestion notification by a fraction of a RTT, however this is compensated for by the protocol

reacting more rapidly to any such indication. The deterministic test carries a greater risk of delaying congestion notification and consequently the protocol mandates that a congestion response should happen whilst performing the test. The two tests combine to provide a mechanism to allow the sender to judge the compliance of a receiver in a manner that both encourages compliant behaviour and proves non-compliance in a robust manner. The most attractive feature of this scheme is that it requires no active participation by the receiver as it utilises the standard behaviour of TCP in the presence of missing data. The only changes required are at the sender.

As mentioned in the introduction, the tests described in this document aren't intended to become a necessary feature for compliant TCP stacks. Rather, the intention is to provide a safe testing mechanism that a sender could choose to implement were it to decide there is a need. If optimistic acknowledgements do start to become widely exploited the authors of this draft feel it would be valuable to have an IETF-approved test that can be used to identify non-compliant receivers. In the mean-time these tests can be used for a number of alternative purposes such as testing that a new receiver stack is indeed compliant with the protocol and testing if a receiver has correctly implemented SACK.

In the longer term it would be hoped that the TCP protocol could be modified to make it robust against such non-compliant behaviour, possibly through the incorporation of a cumulative transport layer nonce as described in Section 5.3.

### 13. Acknowledgements

The authors would like to acknowledge the assistance and comments they received from contributors to the TCPM mailing list. In particular we would like to thank Mark Allman, Caitlin Bestler, Lars Eggert, Gorrry Fairhurst, John Heffner, Alfred Hoenes, David Mallone, Gavin McCullagh, Anantha Ramaiah, Rob Sherwood, Joe Touch and Michael Welzl.

Bob Briscoe was part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The views expressed here are solely those of the authors.

### 14. Comments Solicited

Comments and questions are encouraged and very welcome. They can be addressed to the IETF TCP Maintenance and Minor Extensions working group mailing list <tcpm@ietf.org>, and/or to the authors.

## 15. References

## 15.1. Normative References

- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
- [RFC0813] Clark, D., "Window and Acknowledgement Strategy in TCP", RFC 813, July 1982.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.
- [RFC5961] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", RFC 5961, August 2010.

## 15.2. Informative References

- [Piratla] Piratla, N., Jayasumana, A., and T. Banka, "On reorder density and its application to characterization of packet reordering", IEEE Conference on Local Computer Networks 2005, 2005.
- [RFC0896] Nagle, J., "Congestion control in IP/TCP internetworks", RFC 896, January 1984.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

- [RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, June 2003.
- [RFC3714] Floyd, S. and J. Kempf, "IAB Concerns Regarding Congestion Control for Voice Traffic in the Internet", RFC 3714, March 2004.
- [RFC4774] Floyd, S., "Specifying Alternate Semantics for the Explicit Congestion Notification (ECN) Field", BCP 124, RFC 4774, November 2006.
- [RFC6660] Briscoe, B., Moncaster, T., and M. Menth, "Encoding Three Pre-Congestion Notification (PCN) States in the IP Header Using a Single Diffserv Codepoint (DSCP)", RFC 6660, July 2012.
- [SCPS] Consultative Committee for Space Data Systems, "Space Control Protocol Specification - Transport Protocol", CCSDS Recommended Standard CCSDS 714.0-B-2, 2006.
- [Savage] Savage, S., Cardwell, N., Wetherall, D., and T. Anderson, "TCP congestion control with a misbehaving receiver", ACM SIGCOMM Computer Communications Review Vol.29/5, 1999.
- [Sherwood] Sherwood, R., Bhattacharjee, B., and R. Braud, "Misbehaving TCP receivers can cause Internet-wide congestion collapse", Proceedings of the 12th ACM conference on Computer and communications security 2005, 2005.
- [Spurious] Welzl, M., "Using the ecn nonce to detect spurious loss events in TCP", IEEE Global Telecommunications Conference 2008, 2008.
- [VU102014] US Cert, "Optimistic TCP acknowledgements can cause denial of service", Vulnerability Note 102014, 2005.

Appendix A. Changes from previous drafts (to be removed by the RFC Editor)

From -02 to -03:

Draft revived after 6 year hiatus. Status changed to experimental. The primary aim of the experiment is to show that

these tests correctly and safely identify misconfigured or misbehaving TCP receivers. The secondary aim is to demonstrate that the ECN Nonce is not needed and hence show that that experiment has failed. Minor changes made to tighten the text.

From -01 to -02:

A number of changes made following an extensive review from Alfred Hoenes. These were largely to better comply with the stated aims of the previous version but also included some tidying up of the protocol details and a new section on a possible unwanted interaction.

From -00 to -01:

Draft rewritten to emphasise testing for non-compliance. Some changes to protocol to remove possible unwanted interactions with other TCP variants. Sections added on comparison of solutions and alternative uses of test.

#### Authors' Addresses

Toby Moncaster (editor)  
University of Cambridge  
Computer Laboratory  
J.J. Thomson Avenue  
Cambridge CB3 0FD  
UK

Phone: +44 1223 763654  
Email: toby.moncaster@cl.cam.ac.uk

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

Phone: +44 1473 645196  
Email: bob.briscoe@bt.com



Arnaud Jacquet  
BT  
B54/70, Adastral Park  
Martlesham Heath  
Ipswich IP5 3RE  
UK

Phone: +44 1473 647284  
Email: arnaud.jacquet@bt.com

Network Working Group  
Internet-Draft  
Intended status: Experimental  
Expires: April 19, 2016

Y. Nishida  
GE Global Research  
October 17, 2015

A-PAWS: Alternative Approach for PAWS  
draft-nishida-tcpm-apaws-02

Abstract

This documents describe a technique called A-PAWS which can provide protection against old duplicates segments like PAWS. While PAWS requires TCP to set timestamp options in all segments in a TCP connection, A-PAWS supports the same feature without using timestamps. A-PAWS is designed to be used complementary with PAWS. TCP needs to use PAWS when it is necessary and activates A-PAWS only when it is safe to use. Without impairing the reliability and the robustness of TCP, A-PAWS can provide more option space to other TCP extensions.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 19, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	2
2. Conventions and Terminology . . . . .	3
3. The A-PAWS Design . . . . .	3
3.1. Signaling Methods . . . . .	4
3.2. A-PAWS Negotiation Logic for non-SYN Segment Signaling . . . . .	5
3.3. Sending Behavior . . . . .	6
3.4. Receiving Behavior . . . . .	6
4. When To Activate A-PAWS . . . . .	6
5. Discussion . . . . .	7
5.1. Protection Against Early Incarnations . . . . .	7
5.2. Protection Against Security Threats . . . . .	7
5.3. Middlebox Considerations . . . . .	8
5.4. Aggressive Mode in A-PAWS . . . . .	8
6. Security Considerations . . . . .	9
7. IANA Considerations . . . . .	9
8. References . . . . .	9
8.1. Normative References . . . . .	9
8.2. Informative References . . . . .	9
Author's Address . . . . .	10

## 1. Introduction

PAWS (Protect Against Wrapped Sequences) defined in [RFC1323] is a technique that can identify old duplicate segments in a TCP connection. An old duplicate segment can be generated when it has been delayed by queueing, etc. If such a segment has the sequence number which falls within the receiver's current window, the receiver will accept it without any warning or error. However, this segment can be a segment created by an old connection that has the same port and address pair, or a segments sent 2\*\*32 bytes earlier on the same connection. Although this situation rarely happens, it impairs the reliability of TCP.

PAWS utilizes timestamp option in [RFC1323] to provide protection against this. It is assumed that every received TCP segment contains a timestamp. PAWS can identify old duplicate segments by comparing the timestamp in the received segments and the timestamps from other segments received recently. If both TCP endpoints agree to use PAWS, all segments belong to this connection should have timestamp. Since PAWS is the only standardized protection against old duplicate segments, it has been implemented and used in most TCP

implementations. However, as some TCP extensions such as [RFC2018], [RFC5925] and [RFC6824] also requires a certain amount of option space in non-SYN segments, using 10-12 bytes length in option space for timestamp in all segments tends to be considered expensive in recent discussions.

In addition, although PAWS is necessary for connections which transmit more than  $2^{32}$  bytes, it is not very important for other connections since [RFC0793] already has protection against segments from old connections by using timers. Moreover, some research results indicates that most of TCP flows tend to transmit small amount of data, which means only small fraction of TCP connections really need PAWS [QIAN11]. Timestamp option is also used for RTTM (Round Trip Time Measurement) in [RFC1323]. Gathering many RTT samples from the timestamp in every TCP segment looks useful approach to improve RTO estimation. However, some research results shows the number of samples per RTT does not affect the effectiveness of the RTO [MALLMAN99]. Hence, we can think if PAWS is not used, sending a few timestamps per RTT will be sufficient.

Based on these observations, we propose a new technique called A-PAWS which can archive similar protection against old duplicates segments. The basic idea of A-PAWS is to attain the same protection against old all duplicate segments as PAWS while reducing the use of TS options in segments. A-PAWS is designed to be used complementary with PAWS. This means an implementation that supports A-PAWS is still required to supports PAWS. A-PAWS is activated only when it is safe to use. This sounds the applicability of A-PAWS is limited, however, we believe TCP will have a lot of chances to save the option space if it uses A-PAWS.

There are some discussions that PAWS can also be used to enhance security, however, we still believe that A-PAWS can maintain the same level of security as PAWS. Detailed discussions on this point are provided in Section 5. A-PAWS is an experimental idea yet, but we hope it will contribute to facilitating the use of TCP option space.

## 2. Conventions and Terminology

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 [RFC2119].

## 3. The A-PAWS Design

A-PAWS assumes PAWS as it is designed to be used complementary with PAWS. Hence, a node which supports A-PAWS MUST support PAWS. The following mechanisms are required in TCP in order to perform A-PAWS.

### 3.1. Signaling Methods

An endpoint that supports A-PAWS can use the following signaling methods to activate A-PAWS logic.

#### 1) Option Exchange in SYN

This method uses a new experimental TCP option defined in [RFC6994] and exchanges it during SYN negotiation. The format of the option is depicted in Figure 1. The option does not have any content as it simply indicates the endpoint supports A-PAWS. In this signaling method, when an endpoint wants to use A-PAWS, it MUST put A-PAWS option in SYN or SYN-ACK segment. If an endpoint does not find A-PAWS option in received SYN or SYN-ACK segment, it MUST not send segments with A-PAWS logic in Section 3.3. However, it MUST activate A-PAWS receiver logic in Section 3.4 if it has sent A-PAWS option in SYN or SYN-ACK segment. This is because some middleboxes may remove A-PAWS option in SYN or SYN-ACK segment. A-PAWS receiver logic in Section 3.4 can interact with both A-PAWS and PAWS sender. This signaling requires additional option space in SYN segments, hence non-SYN segment signaling should be used when there is not enough space in SYN option space.

#### 2) Option Exchange in non-SYN Segments

This method uses the option in Figure 1 as well as the SYN segment signaling. However, the options are not exchanged during SYN negotiation. When a endpoint sets A-PAWS option in the segments, it indicates that it can receive the segments from A-PAWS senders. Hence, it MUST activate A-PAWS receiver logic in Section 3.4 if it sends the options. However, it MUST not send segments with A-PAWS logic in Section 3.3 until it receives A-PAWS options. This approach does not require extra option space or special timestamp value in SYN segments. However, negotiating features in non-SYN segments will require to address further arguments such as when to send the options or how to retransmits the options. We discuss these points in the next section and provide some recommended rules for implementations.

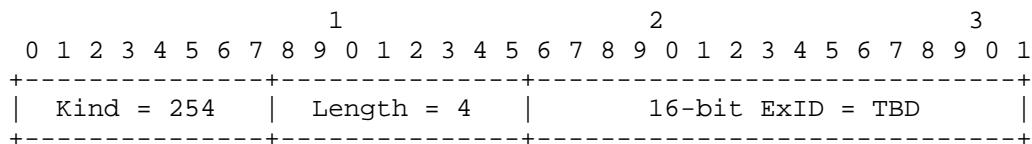


Figure 1: A-PAWS option format

### 3.2. A-PAWS Negotiation Logic for non-SYN Segment Signaling

One important characteristic for A-PAWS is its signaling mechanism does not require tight synchronization between endpoints since A-PAWS receivers can interact with both A-PAWS senders and PAWS senders. This allows us not to invent another three-way handshake like mechanisms for non-SYN segments. This approach will require drastic changes in the current TCP semantics. Instead, we propose a relatively simple and easy mechanism for feature negotiation by using the following rules on A-PAWS endpoints.

Rule 1: An endpoint **MUST** activate A-PAWS receiver logic in Section 3.4 before it sends A-PAWS option.

Rule 2: An endpoint **MUST** not send segments with A-PAWS logic in Section 3.3 until it receives A-PAWS option from the other endpoint.

These rules can avoid situations where an endpoint sends segments by A-PAWS logic to an endpoint that doesn't use A-PAWS logic.

Another discussion point for this signaling method is when to set A-PAWS option in segments. As A-PAWS employs asynchronous signaling, both endpoints basically can set A-PAWS option in segments anytime they want. However, it is recommended to use the following rules for setting A-PAWS options.

Rule 3: An endpoint **SHOULD** use a data segment when it sets A-PAWS option in a segment.

Rule 4: When an endpoint receives a data segment with A-PAWS option, it **SHOULD** set A-PAWS option for its ACK segment.

Rule 5: An endpoint **MAY** use A-PAWS options in retransmitted segments.

These rules allow endpoints to have loose synchronized signaling so that they can at least solicit responses from their peers. Of course, even an endpoint solicits a response by setting A-PAWS option in a data segment, it might not receive A-PAWS option in the ACK segment. This can be caused by the loss of the ACK segment or middleboxes that remove unknown options. In order to address these cases, the following rules can be used.

Rule 6: As long as an endpoint does not violate the other rules, it **MAY** set A-PAWS option in multiple data segments with a certain interval in case no A-PAWS options have been sent from the peer.

This rule can address the cases where A-PAWS options has been removed by middleboxes or segments with A-PAWS options has been lost.

### 3.3. Sending Behavior

A-PAWS enabled TCP transmits segments, it needs to follow the rules below.

1. TCP needs to check how many bytes has been transmitted in a connection. If the transmitted bytes exceeds  $2^{32}$  - 'Sender.Offset', TCP migrates PAWS mode and MUST set timestamp option in all segments to be transmitted. The value for 'Sender.Offset' is discussed in Section 5.
2. If the number of bytes transmitted in a TCP connection does not exceeds  $2^{32}$  - 'Sender.Offset', TCP MAY omit timestamp option in segments as long as it does not affect RTTM. This draft does not define how much TCP can omit timestamps because it should be determined by RTTM.

### 3.4. Receiving Behavior

A-PAWS enabled TCP receives segments, it needs to follow the rules below.

1. TCP needs to check how many bytes has been received in a TCP connection. If it exceeds  $2^{32}$  bytes, A-PAWS nodes SHOULD discard the received segments which does not have timestamp option. TCP MUST perform PAWS check when received bytes exceeds  $2^{32}$  bytes.
2. If the number of bytes received in a TCP connection does not exceeds  $2^{32}$  bytes, A-PAWS nodes SHOULD accept the segments even if it does not have timestamp option. A-PAWS nodes MAY skip PAWS check until the received bytes exceeds  $2^{32}$  bytes.

## 4. When To Activate A-PAWS

In basic principal, A-PAWS capable nodes can always use A-PAWS logic as long as the peers agree with them. However, the following cases require special considerations to enable A-PAWS.

1. As "When To Keep Quiet" section in [RFC0793] suggests, it is recommended that TCP keeps quiet for a MSL upon starting up or recovering from a crash where memory of sequence numbers has been lost. However, if timestamps are being used and if the timestamp clock can be guaranteed to be increased monotonically, this quiet time may be unnecessary. Because TCP can identify the segments

from old connections by checking the timestamp. We think some TCP implementations may disable the quiet time because of using timestamps from this reason. However, since A-PAWS nodes does not set timestamp options in all segments, TCP cannot rely on this approach. To avoid decreasing the robustness of TCP connection, TCP MUST NOT use A-PAWS for a MSL upon starting up or recovering from a crash.

2. Various TCP implementations provide APIs such as `setsockopt()` that can set `SO_REUSEADDR` flag on TCP connections. If this flag is set, the TCP connection allows to reuse the same local port without waiting for 2 MSL period. While this option is useful when users want to relaunch applications immediately, it makes the TCP connection a little vulnerable as TCP stack might receive duplicate segments from earlier incarnations. It has been said that PAWS can contribute to mitigate this risk by checking the timestamps in segments. In order to keep the same level of protection, TCP SHOULD NOT send A-PAWS option when `SO_REUSEADDR` flag is set. This rule prevents the peer from sending segments to this node with A-PAWS logic. However, the node can send segments with A-PAWS logic as long as it received A-PAWS option from the peer.

## 5. Discussion

As A-PAWS is an experimental logic, the following points need to be considered and discussed.

### 5.1. Protection Against Early Incarnations

There are some discussions that timestamp can enhance the robustness against early incarnations. Since A-PAWS does not set timestamps in all segments, some may say that it degrades the robustness of TCP. We believe that the degradation caused by A-PAWS on this point is negligible. As long as TCP limits the usage of A-PAWS as described in Section 4, duplicate segments from early incarnations should not be received by TCP.

### 5.2. Protection Against Security Threats

A TCP connection can be identified by a 5-tuple: source address, destination address, source port number, destination port number and protocol. Crackers need to guess all these parameters when they try malicious attacks on the connection. PAWS can enhance the protection for this as it additionally requires timestamp checking. However, we think the effect of PAWS against malicious attacks is limited due to the simplicity of PAWS check. In PAWS, a segment can be considered as an old duplicate if the timestamp in the segment less than some



timestamps recently received on the connection. The "less than" in this context is determined by processing timestamp values as 32 bit unsigned integers in a modular 32-bit space. For example, if  $t_1$  and  $t_2$  are timestamp values,  $t_1 < t_2$  is verified when  $0 < (t_2 - t_1) < 2^{31}$  computed in unsigned 32-bit arithmetic. Hence, if crackers set a random value in the timestamp option, there will be 50% chance for them to trick PAWS check. Moreover, there will be more chances if they send multiple segments with different timestamps, which will not be difficult to perform.

In addition, we think there might be a case where using PAWS increases security risks. PAWS recommends to increase timestamp over a system when TCP waives the "quiet time" described in [RFC0793]. However, if timestamps are generated from a global counter, it may leak some information such as system uptime as discussed in [SILBERSACK05]. A-PAWS might be able to allow TCP to use random timestamp values per connections.

### 5.3. Middlebox Considerations

A-PAWS is designed to be robust against middleboxes. This means that endpoints will not be messed up even if middleboxes discard A-PAWS option. This is because A-PAWS sender logic is activated only when TCP receives a segment with A-PAWS options. A-PAWS receiver logic does not need to know whether the sender is using PAWS or A-PAWS. Activating A-PAWS receiving logic for PAWS sender might be redundant as it requires additional overheads. However, we believe the overhead will be acceptable in most cases because of the simplicity of A-PAWS logic.

Another concern on middleboxes is that they can insert or delete some bytes in TCP connections. If a middlebox inserts extra bytes into a TCP connections, there might be a situation where an A-PAWS sender can transmit segments without timestamp, while an A-PAWS receiver perform PAWS check on them as it already has received  $2^{32}$  bytes. In order to avoid discarding segments unnecessarily, we recommend that A-PAWS sender should have a certain amount of offset bytes in order to migrate PAWS mode before the receiver receives  $2^{32}$  bytes. We call this protocol parameter 'Sender.Offset'. The proper value for 'Sender.Offset' needs to be discussed.

### 5.4. Aggressive Mode in A-PAWS

The current A-PAWS requires TCP to migrate PAWS mode after sending/receiving  $2^{32}$  bytes. However, if both nodes check if 2 MSL has already passed during sending/receiving  $2^{32}$  bytes, it is safe to continue using A-PAWS. We call this Aggressive mode. The use of Aggressive mode will be explored in future versions.

## 6. Security Considerations

We believe A-PAWS can maintain the same level of security as PAWS does, but further discussions will be needed. Some security aspects of A-PAWS are discussed in Section 5.

## 7. IANA Considerations

This document uses the Experimental Option Experiment Identifier. An application for this codepoint in the IANA TCP Experimental Option ExID registry will be submitted.

## 8. References

### 8.1. Normative References

- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
- [RFC1323] Jacobson, V., Braden, R., and D. Borman, "TCP Extensions for High Performance", RFC 1323, DOI 10.17487/RFC1323, May 1992, <<http://www.rfc-editor.org/info/rfc1323>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

### 8.2. Informative References

- [MALLMAN99] Allman, M. and V. Paxson, "On Estimating End-to-End Network Path Properties", Proceedings of the ACM SIGCOMM , September 1999.
- [QIAN11] Qian, L. and B. Carpenter, "A Flow-Based Performance Analysis of TCP and TCP Applications", 3rd International Conference on Computer and Network Technology (ICCNT 2011) , February 2011.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, DOI 10.17487/RFC2018, October 1996, <<http://www.rfc-editor.org/info/rfc2018>>.
- [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <<http://www.rfc-editor.org/info/rfc5925>>.

- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <<http://www.rfc-editor.org/info/rfc6824>>.
- [RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, August 2013.
- [SILBERSACK05] Silbersack, M., "Improving TCP/IP security through randomization without sacrificing interoperability.", EuroBSDCon 2005 , November 2005.

Author's Address

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

Email: [nishida@wide.ad.jp](mailto:nishida@wide.ad.jp)

TCPM WG  
Internet Draft  
Updates: 793  
Intended status: Standards Track  
Expires: January 2015

J. Touch  
USC/ISI  
Wes Eddy  
MTI Systems  
July 2, 2014

TCP Extended Data Offset Option  
draft-touch-tcpm-tcp-edo-03.txt

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at  
<http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at  
<http://www.ietf.org/shadow.html>

This Internet-Draft will expire on January 2, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

## Abstract

TCP segments include a Data Offset field to indicate space for TCP options, but the size of the field can limit the space available for complex options that have evolved. This document updates RFC 793 with an optional TCP extension to that space to support the use of multiple large options such as SACK with either TCP Multipath or TCP AO. It also explains why the initial SYN of a connection cannot be extending as a single segment.

## Table of Contents

1. Introduction.....	2
2. Conventions used in this document.....	3
3. Requirements for Extending TCP's Data Offset.....	3
4. The TCP EDO Option.....	3
5. TCP EDO Interaction with TCP.....	6
5.1. TCP User Interface.....	6
5.2. TCP States and Transitions.....	6
5.3. TCP Segment Processing.....	6
5.4. Impact on TCP Header Size.....	6
5.5. Connectionless Resets.....	7
5.6. ICMP Handling.....	8
6. Interactions with Middleboxes.....	8
7. Comparison to Previous Proposals.....	9
7.1. EDO Criteria.....	9
7.2. Summary of Approaches.....	10
7.3. Extended Segments.....	11
7.4. TCPx2.....	11
7.5. LOO/SLO.....	12
7.6. LOIC.....	12
7.7. Problems with Extending the Initial SYN.....	13
8. Security Considerations.....	14
9. IANA Considerations.....	14
10. References.....	14
10.1. Normative References.....	14
10.2. Informative References.....	15
11. Acknowledgments.....	16

## 1. Introduction

TCP's Data Offset is a 4-bit field, which indicates the number of 32-bit words of the entire TCP header [RFC793]. This limits the current total header size to 60 bytes, of which the basic header occupies 20, leaving 40 bytes for options. These 40 bytes are increasingly becoming a limitation to the development of advanced

capabilities, such as when SACK [RFC2018][RFC6675] is combined with either Multipath TCP [RFC6824] or TCP-AO [RFC5925].

This document specifies the TCP Extended Data Offset (EDO) option, and is independent of (and thus compatible with) IPv4 and IPv6. EDO extends the space available for TCP options, except for the initial SYN segment (i.e., SYN and not ACK, the first segment of a new connection). This document also explains why the option space of a single initial SYN segment cannot be extended without severe impact on TCP's initial handshake.

## 2. Conventions used in this document

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

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

In this document, the characters ">>" preceding an indented line(s) indicates a compliance requirement statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the explicit compliance requirements of this RFC.

## 3. Requirements for Extending TCP's Data Offset

The primary goal of extending the TCP Data Offset field is to increase the space available for TCP options in all segments except the initial SYN.

An important requirement of any such extension is that it not impact legacy endpoints. Endpoints seeking to use this new option should not incur additional delay or segment exchanges to connect to either new endpoints supporting this option or legacy endpoints without this option. We call this a "backward downgrade" capability.

## 4. The TCP EDO Option

TCP EDO extends the option space for all segments except the initial SYN (i.e., SYN set and ACK not set). The EDO option is organized as indicated in Figure 1 and Figure 2. For initial SYN segments (i.e., those whose ACK bit is not set), the EDO request option consists of the required Kind and Length fields only. All other segments optionally use the EDO length option, which adds a Header\_Length field (in network-standard byte order), indicating the length of the

entire TCP header in bytes. The codepoint value of the EDO Kind is EDO-OPT.

```

+-----+-----+
| Kind  | Length |
+-----+-----+

```

Figure 1 TCP EDO request option

```

+-----+-----+-----+-----+
| Kind  | Length | Header_length |
+-----+-----+-----+-----+

```

Figure 2 TCP EDO length option

EDO support is determined in both directions using the same exchange. An endpoint seeking to enable EDO support includes the EDO request option in the initial SYN.

>> Connections using EDO MUST negotiate its availability during the initial three-way handshake.

>> An endpoint confirming EDO support MUST respond with EDO length option in its SYN-ACK.

The EDO length option is required in SYN-ACKs when confirming support for EDO. The SYN-ACK thus can take advantage of EDO, using it to extend its option space. If such extension is not required, then EDO would be equal to 4 \* Data Offset (i.e., EDO would indicate in bytes the same length indicated by Data Offset in words).

>> Once negotiated on a connection, EDO MAY be present as needed on other segments in either direction. The EDO option SHOULD NOT be used if the total option space needed can be accommodated by the existing Data Offset field.

>> The EDO request option (i.e., whose option length is 2) MAY occur in an initial SYN as desired (e.g., by the user/application), but MUST NOT be inserted in other segments. If the EDO request option is received, it MUST be silently ignored.

>> The EDO length option MAY occur in segments other than the initial SYN if EDO has been negotiated on a connection. If EDO has not been negotiated and agreed, the EDO length option MUST be silently ignored. The EDO length option MUST NOT be sent in an initial SYN segment, and MUST be silently ignored and not acknowledged if so received.

EDO extends the space available for options, but does not consume space unless needed. Segments that don't need the additional space can use the existing Data Offset field as currently specified [RFC793]. When processing a segment, EDO needs to be visible within the area indicated by the Data Offset field, so that processing can use the EDO Header\_length to override the Data Offset for that segment.

>> The EDO length option MUST occur within the length of the TCP Data Offset.

>> The EDO length option indicates the total length of the header. The EDO Header\_length field MUST NOT exceed that of the total segment size (i.e., TCP Length). The EDO Header\_length SHOULD be a multiple of 4 to simplify processing.

>> The EDO request option SHOULD be aligned on a 16-bit boundary and the EDO length option SHOULD be aligned on a 32-bit boundary, in both cases for simpler processing.

For example, a segment with only EDO would have a Data Offset of 6, where EDO would be the first option processed, at which point the EDO length option would override the Data Offset and processing would continue until the end of the TCP header as indicated by the EDO Header\_length field.

There are cases where it might be useful to process other options before EDO, notably those that determine whether the TCP header is valid, such as authentication, encryption, or alternate checksums. In those cases, the EDO length option is preferably the first option after a validation option, and the payload after the Data Offset is treated as user data for the purposes of validation.

>> The EDO length option SHOULD occur as early as possible, either first or just after any authentication or encryption, and SHOULD be the last option covered by the Data Offset value.

Other options are generally handled in the same manner as when the EDO option is not active, unless they interact with other options. One such example is TCP-AO [RFC5925], which optionally ignores the contents of TCP options, so it would need to be aware of EDO to operate correctly when options are excluded from the HMAC calculation.

>> Options that depend on other options, such as TCP-AO [RFC5925] (which may include or exclude options in MAC calculations) MUST also



be augmented to interpret the EDO length option to operate correctly.

## 5. TCP EDO Interaction with TCP

The following subsections describe how EDO interacts with the TCP specification [RFC793].

### 5.1. TCP User Interface

The TCP EDO option is enabled on a connection using a mechanism similar to any other per-connection option. In Unix systems, this is typically performed using the 'setsockopt' function.

>> Implementations can also employ system-wide defaults, however systems SHOULD NOT use this extension by default to avoid interfering with legacy applications.

### 5.2. TCP States and Transitions

TCP EDO does not alter the existing TCP state or state transition mechanisms.

### 5.3. TCP Segment Processing

TCP EDO alters segment processing during the TCP option processing step. Once detected, the TCP EDO length option overrides the TCP Data Offset field for all subsequent option processing. Option processing continues at the next option after the EDO length option.

Implementers need to be careful about the potential for offload support interfering with this option. The EDO data needs to be passed to the protocol stack as part of the option space, not integrated with the user segment, to allow the offload to independently determine user data segment boundaries and combine them correctly with the extended option data.

### 5.4. Impact on TCP Header Size

The TCP EDO request option increases SYN header length by a minimum of 2 bytes. Currently popular SYN options total 19 bytes, which leaves more than enough room for the EDO request:

- o SACK permitted (2 bytes in SYN, optionally 2 + 8N bytes after) [RFC2018][RFC6675]
- o Timestamp (10 bytes) [RFC1323]

- o Window scale (3 bytes) [RFC1323]
- o MSS option (4 bytes) [RFC793]

Adding the EDO option would result in a total of 21 bytes of SYN option space. Subsequent segments would use 19 bytes of option space without any SACK blocks or allow up to 4 SACK blocks before needing to use EDO; with EDO, the number of SACK blocks or additional options would be substantially increased.

TCP EDO can also be negotiated in SYNs with either of the following options:

- o TCP-AO (authentication) (16 bytes) [RFC5925]
- o Multipath TCP (12 bytes in SYN and SYN-ACK, 20 after) [RFC6824]

Including TCP-AO increases the SYN option space use to 37 bytes; with Multipath TCP the use is 33 bytes. When Multipath TCP is enabled with the typical options, later segments might require 39 bytes without SACK, thus effectively disabling the SACK option unless EDO is also supported on at least non-SYN segments.

The full combination of the above options (49 bytes including EDO) does not fit in the existing SYN option space and (as noted) that space cannot be extended within a single SYN segment. There has been a proposal to change TS to a 2 byte "TS permitted" signal in the initial SYN, provided it can be safely enabled during the connection later or might be avoided completely [Nil4]. Even using "TS-permitted", the total space is still too large to support in the initial SYN without SYN option space extension [To14].

The EDO option has negligible impact on other headers, because it can either come first or just after security information, and in either case the additional 4 bytes are easily accommodated within the TCP Data Offset length. Once the EDO option is processed, the entirety of the remainder of the TCP segment is available for any remaining options.

## 5.5. Connectionless Resets

A RST may arrive during a currently active connection or may be needed to cleanup old state from an abandoned connection. The latter occurs when a new SYN is sent to an endpoint with matching existing connection state, at which point that endpoint responds with a RST and both ends remove stale information.

The EDO option is not mandatory in any TCP segment, except the SYN and SYN-ACK of the three-way handshake to establish its support.

>> The EDO length option MAY occur in a RST when the endpoint has connection state that has negotiated EDO. However, unless the RST is generated by an incoming segment that includes an EDO option, the RST MUST NOT include the EDO length option.

## 5.6. ICMP Handling

ICMP responses are intended to include the IP and the port fields of TCP and UDP headers of typical TCP/IP and UDP/IP packets [RFC792]. This includes the first 8 data bytes of the original datagram, intended to include the transport port numbers used for connection demultiplexing. Later specifications encourage returning as much of the original payload as possible [RFC1812]. In either case, legacy options or new options in the EDO extension area might or might not be included, and so options are generally not assumed to be part of ICMP processing anyway.

## 6. Interactions with Middleboxes

Any new TCP option may be impacted by the presence of any on-path device that examines or modifies transport headers [RFC3234]. Boxes that parse or modify TCP options need to follow the same requirements of TCP endpoints in supporting EDO, or they could interfere with connections. The primary concern is so-called "transparent" rewriting proxies, which modify TCP segment boundaries and thus would mix option information with user data if they do not support EDO. Such devices interfere with many other TCP options, and although their use is not common they would interfere with connections that use EDO.

More common are NATs, which rewrite IP address and/or transport port fields. NATs are not affected by the EDO option.

Deep-packet inspection systems that inspect TCP segment payloads or attempt to reconstitute the data stream would incorrectly include options, which might interfere with their operation.

It may be important to detect misbehavior that could cause EDO space to be misinterpreted as user data. In such cases, EDO SHOULD be used in conjunction with integrity protection mechanisms, such as IPsec, TCP-AO, etc. It is useful to note that such protection helps find only non-compliant components.

---

?? should we include a header checksum, as suggested by Oliver Bonaventure, too? Should it be integrated with EDO or independent? If so, we either need one of the following:

Kind	Length	Header_length
EDO-checksum		

Figure 3 TCP EDO length + sum option

Kind	Length	Header_checksum
------	--------	-----------------

Figure 4 TCP header checksum option

The former adds a header checksum to EDO - this could be optional or required, and its presence can be determined from the option length. The checksum would be over only the additional EDO space.

The latter requires a new option and we'd need to decide what the option covered. It could cover just the option space.

Finally, another option would be to try to make rewriting middleboxes fail by some other, more opaque means - such as using DO=0 when EDO is used as suggested by John Leslie (but this would also require considering whether EDO would be required on all packets, or just on DO=0, as well as where EDO would be and whether it could precede security/integrity options such as TCP-AO).

## 7. Comparison to Previous Proposals

EDO is the latest in a long line of attempts to increase TCP option space [Al06][Ed08][Ko04][Ra12][Yo11]. The following is a comparison of these approaches to EDO, based partly on a previous summary [Ra12]. This comparison differs from that summary by using a different set of success criteria.

### 7.1. EDO Criteria

Our criteria for a successful solution are as follows:

- o Zero-cost fallback to legacy endpoints.
- o Minimal impact on middlebox compatibility.

- o No additional side-effects.

Zero-cost fallback requires that upgraded hosts incur no penalty for attempting to use EDO. This disqualifies dual-stack approaches, because the client might have to delay connection establishment to wait for the preferred connection mode to complete. Note that the impact of legacy endpoints that silently reflect unknown options are not considered, as they are already non-compliant with existing TCP requirements [RFC793].

Minimal impact on middlebox compatibility requires that EDO works through simple NAT and NATP boxes, which modify IP addresses and ports and recompute IPv4 header and TCP segment checksums. Middleboxes that reject unknown options or that process segments in detail without regard for unknown options are not considered; they process segments as if they were an endpoint but do so in ways that are not compliant with existing TCP requirements (e.g., they should have rejected the initial SYN because of its unknown options rather than silently relaying it).

EDO also attempts to avoid creating side-effects, such as might happen if options were split across multiple TCP segments (which could arrive out of order or be lost) or across different TCP connections (which could fail to share fate through firewalls or NAT/NATPs).

These requirements are similar to those noted in [Ra12], but EDO groups cases of segment modification beyond address and port - such as rewriting, segment drop, sequence number modification, and option stripping - as already in violation of existing TCP requirements regarding unknown options, and so we do not consider their impact on this new option.

## 7.2. Summary of Approaches

There are three basic ways in which TCP option space extension has been attempted:

1. Use of a TCP option.
2. Redefinition of the existing TCP header fields.
3. Use of option space in multiple TCP segments (split across multiple segments).

A TCP option is the most direct way to extend the option space and is the basis of EDO. This approach cannot extend the option space of the initial SYN.

Redefining existing TCP header fields can be used to either contain additional options or as a pointer indicating alternate ways to interpret the segment payload. All such redefinitions make it difficult to achieve zero-impact backward compatibility, both with legacy endpoints and middleboxes.

Splitting option space across separate segments can create unintended side-effects, such as increased delay to deal with path latency or loss differences.

The following discusses three of the most notable past attempts to extend the TCP option space: Extended Segments, TCPx2, LOO/SLO, and LOIC. [Ra12] suggests a few other approaches, including use of TCP option cookies, reuse/overload of other TCP fields (e.g., the URG pointer), or compressing TCP options. None of these is compatible with legacy endpoints or middleboxes.

### 7.3. Extended Segments

TCP Extended Segments redefined the meaning of currently unused values of the Data Offset (DO) field [Ko04]. TCP defines DO as indicating the length of the TCP header, including options, in 32-bit words. The default TCP header with no options is 5 such words, so the minimum currently valid DO value is 5 (meaning 40 bytes of option space). This document defines interpretations of values 0-4: DO=0 means 48 bytes of option space, DO=1 means 64, DO=2 means 128, DO=3 means 256, and DO=4 means unlimited (e.g., the entire payload is option space). This variant negotiates the use of this capability by using one of these invalid DO values in the initial SYN.

Use of this variant is not backward-compatible with legacy TCP implementations, whether at the desired endpoint or on middleboxes. The variant also defines a way to initiate the feature on the passive side, e.g., using an invalid DO during the SYN-ACK when the initial SYN had a valid DO. This capability allows either side to initiate use of the feature but is also not backward compatible.

### 7.4. TCPx2

TCPx2 redefines legacy TCP headers by basically doubling all TCP header fields [Al06]. It relies on a new transport protocol number to indicate its use, defeating backward compatibility with all

existing TCP capabilities, including firewalls, NATs/NAPTs, and legacy endpoints and applications.

#### 7.5. LOO/SLO

The TCP Long Option (LO, [Ed08]) is very similar to EDO, except that presence of LO results in ignoring the existing DO field and that LO is required to be the first option. EDO considers the need for other fields to be first and declares that the EDO is the last option as indicated by the DO field value. Unlike LO, EDO is not required in every segment once negotiated, saving 6 bytes if not actively needed.

The TCP Long Option draft also specified the SYN Long Option (SLO) [Ed08]. If SLO is used in the initial SYN and successfully negotiated, it is used in each subsequent segment until all of the initial SYN options are transmitted.

LO is backward compatible, as is SLO; in both cases, endpoints not supporting the option would not respond with the option, and in both cases the initial SYN is not itself extended.

SLO does modify the three-way handshake because the connection isn't considered completely established until the first data byte is ACKed. Legacy TCP can establish a connection even in the absence of data. SLO also changes the semantics of the SYN-ACK; for legacy TCP, this completes the active side connection establishment, where in SLO an additional data ACK is required. A connection whose initial SYN options have been confirmed in the SYN-ACK might still fail upon receipt of additional options sent in later SLO segments. This case - of late negotiation fail - is not addressed in the specification.

#### 7.6. LOIC

TCP Long Options by Invalid Checksum is a dual-stack approach that uses two initial SYNS to initiate all updated connections [Yo11]. One SYN negotiates the new option and the other SYN payload contains only the entire options. The negotiation SYN is compliant with existing procedures, but the option SYN has a deliberately incorrect TCP checksum (decremented by 2). A legacy endpoint would discard the segment with the incorrect checksum and respond to the negotiation SYN without the LO option.

Use of the option SYN and its incorrect checksum both interfere with other legacy components. Segments with incorrect checksums will be silently dropped by most middleboxes, including NATs/NAPTs. Use of two SYNs creates side-effects that can delay connections to upgraded

endpoints, notably when the option SYN is lost or the SYNs arrive out of order. Finally, by not allowing other options in the negotiation SYN, all connections to legacy endpoints either use no options or require a separate connection attempt (either concurrent or subsequent).

#### 7.7. Problems with Extending the Initial SYN

The key difficulty with most previous proposals is the desire to extend the option space in all TCP segments, including the initial SYN, i.e., SYN with no ACK, typically the first segment of a connection. It has proven difficult to extend space in this initial SYN in the absence of prior negotiation while maintaining current TCP three-way handshake properties.

A new TCP option cannot extend the Data Offset of a single TCP initial SYN segment. All TCP segments, including the initial SYN, may include user data in the payload data [RFC793], and this can be useful for some proposed features such as TCP Fast Open [Ch14]. Legacy endpoints that ignore the new option would process the payload contents as user data and send an ACK. Once ACK'd, this data cannot be removed from the user stream.

The six Reserved TCP header bits cannot be redefined easily, because the original specification did not require their contents to be ignored, even though three of those bits have already been redefined (ECE/CWR [RFC3168] and NS [RFC3540]). Legacy endpoints are required to drop TCP segments where those bits are not zero, making it difficult to assume backward downgrade.

TCP initial SYN (SYN and not ACK) segments can use every other TCP header field except the Acknowledgement number, which is not used because the ACK field is not set. In all other segments, all fields except the three remaining Reserved header bits are actively used. The total amount of available header fields, in either case, is insufficient to be useful in extending the option space.

The representation of TCP options can be optimized to minimize the space needed. In such cases, multiple Kind and Length fields are combined, so that a new Kind would indicate a specific combination of options, whose order is fixed and whose length is indicated by one Length field. Most TCP options use fields whose size is much larger than the required Kind and Length components, so the resulting efficiency is typically insufficient for additional options.



The option space of an initial SYN segment might be extended by using multiple initial segments (e.g., multiple SYNs or a SYN and non-SYN) or based on the context of previous or parallel connections. Because of their potential complexity, these approaches are addressed in a separate document [To14].

Option space cannot be extended in outer layer headers, e.g., IPv4 or IPv6. These layers typically try to avoid extensions altogether, to simplify forwarding processing at routers. Introducing new shim layers to accommodate additional option space would interfere with deep-packet inspection mechanisms that are in widespread use.

As a result, EDO does not attempt to extend the space available for options in TCP initial SYNs. It does extend that space in all other segments (including SYN-ACK), which has always been trivially possible once an option is defined.

## 8. Security Considerations

It is meaningless to have the Data Offset further exceed the position of the EDO data offset option.

>> When the EDO length option is present, the EDO length option SHOULD be the last non-null option covered by the TCP Data Offset, because it would be the last option affected by Data Offset.

This also makes it more difficult to use the Data Offset field as a covert channel.

## 9. IANA Considerations

We request that, upon publication, this option be assigned a TCP Option codepoint by IANA, which the RFC Editor will replace EDO-OPT in this document with codepoint value.

This section is to be removed prior to publication as an RFC.

## 10. References

### 10.1. Normative References

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

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

## 10.2. Informative References

- [Al06] Allman, M., "TCPx2: Don't Fence Me In", draft-allman-tcp2-hack-00 (work in progress), May 2006.
- [Ch14] Cheng, Y., Chu, J., and A. Jain, "TCP Fast Open", draft-ietf-tcpm-fastopen-09, June 2014.
- [Ed08] Eddy, W. and A. Langley, "Extending the Space Available for TCP Options", draft-eddy-tcp-loo-04 (work in progress), July 2008.
- [Ko04] Kohler, E., "Extended Option Space for TCP", draft-kohler-tcpm-extopt-00 (work in progress), September 2004.
- [Ni14] Nishida, Y., "A-PAWS: Alternative Approach for PAWS", draft-nishida-tcpm-apaws-01 (work in progress), June 2014.
- [Ra12] Ramaiah, A., "TCP option space extension", draft-ananth-tcpm-tcproptext-00 (work in progress), March 2012.
- [RFC792] Postel, J., "Internet Control Message Protocol", RFC 792.
- [RFC1323] Jacobson, V., Braden, R., and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.
- [RFC1812] Baker, F. (Ed.), "Requirements for IP Version 4 Routers", RFC 1812, June 1995.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.
- [RFC3234] Carpenter, B. and S. Brim, "Middleboxes: Taxonomy and Issues", RFC 3234, February 2002.
- [RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, June 2003.
- [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, June 2010.

- [RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y.. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, August 2012.
- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, January 2013.
- [Tol4] Touch, J., (et al., TBD), "Extending TCP Option Space During Connection Establishment", draft-touch-tcp-syn-edo-00 (work in progress), July 2014.
- [Yol1] Yourtchenko, A., "Introducing TCP Long Options by Invalid Checksum", draft-yourtchenko-tcp-loic-00 (work in progress), April 2011.

## 11. Acknowledgments

The authors would like to thank the IETF TCPM WG for their feedback, in particular: Oliver Bonaventure, Bob Briscoe, Ted Faber, John Leslie, Richard Scheffenegger, and Alexander Zimmerman.

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

## Authors' Addresses

Joe Touch  
USC/ISI  
4676 Admiralty Way  
Marina del Rey, CA 90292-6695 USA

Phone: +1 (310) 448-9151  
Email: touch@isi.edu

Wesley M. Eddy  
MTI Systems  
US

Email: wes@mti-systems.com



TCP Maintenance and Minor Extensions (TCPM) WG  
Internet-Draft  
Intended status: Standards Track  
Expires: May 14, 2015

A. Zimmermann  
NetApp, Inc.  
L. Schulte  
Aalto University  
C. Wolff  
A. Hannemann  
credativ GmbH  
November 10, 2014

Detection and Quantification of Packet Reordering with TCP  
draft-zimmermann-tcpm-reordering-detection-02

Abstract

This document specifies an algorithm for the detection and quantification of packet reordering for TCP. In the absence of explicit congestion notification from the network, TCP uses only packet loss as an indication of congestion. One of the signals TCP uses to determine loss is the arrival of three duplicate acknowledgments. However, this heuristic is not always correct, notably in the case when paths reorder packets. This results in degraded performance.

The algorithm for the detection and quantification of reordering in this document uses information gathered from the TCP Timestamps Option, the TCP SACK Option and its DSACK extension. When a reordering event is detected, the algorithm calculates a reordering extent by determining the number of segments the reordered segment was late with respect to its position in the sequence number space. Additionally, the algorithm computes a second reordering extent that is relative to the amount of outstanding data and thus provides a better estimation of the reordering delay when other sender state changes.

Status of This Memo

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

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 14, 2015.

#### Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

#### Table of Contents

1. Introduction . . . . .	3
2. Terminology . . . . .	4
3. Basic Concepts . . . . .	5
4. The Algorithm . . . . .	5
4.1. Initialization During Connection Establishment . . . . .	6
4.2. Receiving Acknowledgments . . . . .	6
4.3. Receiving Acknowledgment Closing Hole . . . . .	7
4.4. Receiving Duplicate Selective Acknowledgment . . . . .	8
4.5. Reordering Extent Computation . . . . .	8
4.6. Retransmitting Segment . . . . .	9
4.7. Placeholder for Response Algorithm . . . . .	9
4.8. Retransmission Timeout . . . . .	9
5. Protocol Steps in Detail . . . . .	9
6. Discussion of the Algorithm . . . . .	12
6.1. Reasoning for the Relative Reordering Extent . . . . .	12
6.2. Calculation of the Relative Reordering Extent . . . . .	13
6.3. Persistent Reception of Selective Acknowledgments . . . . .	13
6.4. Unreliable ACK reception . . . . .	15
6.5. Packet Duplication . . . . .	15
7. Related Work . . . . .	16
8. IANA Considerations . . . . .	17
9. Security Considerations . . . . .	17
10. Acknowledgments . . . . .	17
11. References . . . . .	17
11.1. Normative References . . . . .	17
11.2. Informative References . . . . .	18

Appendix A. Changes from previous versions of the draft	20
A.1. Changes from draft-zimmermann-tcpm-reordering-detection-01	21
A.2. Changes from draft-zimmermann-tcpm-reordering-detection-00	21
Authors' Addresses	21

## 1. Introduction

When the Transmission Control Protocol (TCP) [RFC0793] decides that the oldest outstanding segment is lost, it performs a retransmission and changes the sending rate [RFC5681]. This occurs either when the Retransmission Timeout (RTO) timer expires for a segment [RFC6298], or when three duplicate acknowledgments (ACKs) for a segment have been received (Fast Retransmit/Fast Recovery) [RFC5681]. The assumption behind Fast Retransmit is that non-congestion events that can cause duplicate ACKs to be generated (packet duplication, packet reordering and packet corruption) are infrequent. However, a number of Internet measurement studies have shown that packet reordering is not a rare phenomenon [Pax97], [BPS99], [BS02], [ZM04], [GPL04], [JIDKT07] and has negative performance implications on TCP [BA02], [ZKFP03].

From TCP's perspective, the result of packet reordering on the forward-path is the reception of out-of-order segments by the TCP receiver. In response to every received out-of-order segment, the TCP receiver immediately sends a duplicate ACK. (Note: [RFC5681] recommends that delayed ACKs not be used when the ACK is triggered by an out-of-order segment.) The sender side, if the number of consecutively received duplicate ACKs exceeds the duplicate acknowledgment threshold (DupThresh), retransmits the first unacknowledged segment [RFC5681] and continues with a loss recovery algorithm such as NewReno [RFC6582] or the Selective Acknowledgment (SACK) based loss recovery [RFC6675]. If a segment arrives due to reordering more than three segments (the default value of DupThresh [RFC5681]) too late at the TCP receiver, the sender is not able to distinguish this reordering event from a segment loss, resulting in an unnecessary retransmission and rate reduction.

Since DupThresh is defined in segments rather than bytes [RFC5681], TCP usually quantifies packet reordering in terms of segments. Informally, the reordering extent [RFC4737] is defined as the maximum distance in segments between the reception of a reordered segment and the earliest segment received with a larger sequence number. If a segment is received in-order, its reordering extent is undefined [RFC4737].

Another approach taken by this specification quantifies the reordering extend for a packet not only through an absolute value, but also through a measure that is relative to the amount of outstanding data, in an attempt to approximate a time-based measure. The presented scheme can thereby easily be adapted to the Stream Control Transmission Protocol (SCTP) [RFC2960], since SCTP uses congestion control algorithms similar to TCP.

Overall, this document describes mechanisms to detect reordering on the forward-path during a TCP connection, and provides these samples as an input for an additional reaction algorithm.

The remainder of this document is organized as follows. Section 3 provides a high-level description of the packet reordering detection mechanisms. In Section 4, the algorithm is specified. In Section 5, each step of the algorithm is discussed in detail. Section 6 provides a discussion of several design decisions of the algorithm. Section 7 discusses related work. Section 9 discusses security concerns.

## 2. Terminology

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 [RFC2119].

The reader is expected to be familiar with the TCP state variables given in [RFC0793] (SEG.SEQ, SND.UNA), [RFC5681] (FlightSize), and [RFC6675] (DupThresh, SACK scoreboard). SND.FACK (forward acknowledgment) is used to describe the highest sequence number - plus one - that has been either cumulatively or selectively acknowledged by the receiver and subsequently seen by the sender [MM96]. Further, the term 'acceptable acknowledgment' is used as defined in [RFC0793]. That is, an ACK that increases the connection's cumulative ACK point by acknowledging previously unacknowledged data. The term 'duplicate acknowledgment' is used as defined in [RFC6675], which is different from the definition of duplicate acknowledgment in [RFC5681].

This specification defines the four TCP sender states 'open', 'disorder', 'recovery', and 'loss' as follows. As long as no duplicate ACK is received and no segment is considered lost, the TCP sender is in the 'open' state. Upon the reception of the first consecutive duplicate ACK, TCP will enter the 'disorder' state. After receiving DupThresh duplicate ACKs, the TCP sender switches to the 'recovery' state and executes standard loss recovery procedures like Fast Retransmit and Fast Recovery [RFC5681]. Upon a retransmission timeout, the TCP sender enters the 'loss' state. The



'recovery' state can only be reached by a transition from the 'disorder' state, the 'loss' state can be reached from any other state.

### 3. Basic Concepts

The following specification depends on the TCP Timestamps option [RFC1323], the TCP Selective Acknowledgment (SACK) [RFC2018] option and the latter's Duplicate Selective Acknowledgment (DSACK) extension [RFC2883]. The reader is assumed to be familiar with the algorithms specified in these documents.

Reordering is quantified by an absolute and a relative reordering extent. If a hole in the SACK scoreboard of the TCP sender is closed either cumulatively by an acceptable ACK or selectively by a new SACK, then the absolute reordering extent is computed as the number of segments in the SACK scoreboard between the sequence number of the reordered segment and the highest selectively or cumulatively acknowledged sequence number. The relative reordering extent is then computed as the ratio between the absolute reordering extent and the FlightSize stored when entering the 'disorder' state.

If the hole that was closed in the SACK scoreboard corresponds to a segment that was not retransmitted, or if the retransmission of such a segment can be determined as a spurious retransmission by means of the Eifel detection algorithm [RFC3522], then the calculated reordering extent is immediately valid. Otherwise, if the verification of the Eifel detection algorithm has not been possible, the reordering extent is stored for a possibly subsequent DSACK. If no such DSACK is received in the next two round-trip times (RTTs), the reordering extents are discarded.

### 4. The Algorithm

Given that usually both the Nagle algorithm [RFC0896] [RFC1122] and the TCP Selective Acknowledgment Option [RFC2018] are enabled, a TCP sender MAY employ the following algorithm to detect and quantify the current perceived packet reordering in the network.

Without the Nagle algorithm, there is no straight forward way to accurately calculate the number of outstanding segments in the network (and, therefore, no good way to derive an appropriate reordering extent) without adding state to the TCP sender. A TCP connection that does not employ the Nagle algorithm SHOULD NOT use this methodology.

If a TCP sender implements the following algorithm, the implementation MUST follow the various specifications provided in

Sections 4.1 to 4.8. The algorithm **MUST** be executed *\*before\** the Transmission Control Block or the SACK scoreboard have been updated by another loss recovery algorithm.

#### 4.1. Initialization During Connection Establishment

After the completion of the TCP connection establishment, the following state variables **MUST** be initialized in the TCP transmission control block:

- (C.1) The variable `Dsack`, which indicates whether a DSACK has been received so far, and the data structure `Samples`, which stores the computed reordering extents, **MUST** be initialized as:

```
Dsack = false
Samples = []
```

- (C.2) If the TCP Timestamps option [RFC1122] has been negotiated, then the variable `Timestamps` **MUST** be activated and the data structure `Retrans_TS`, which stores the value of the `TSval` field of the retransmissions sent during Fast Recovery, **MUST** be initialized. Additionally, the data structure `Retrans_Dsack` **MAY** be used in order to detect reordering longer than RTT with Timestamps and DSACK:

```
Timestamps = true
Retrans_TS = []
Retrans_Dsack = []
```

Otherwise, the Timestamps-based detection **SHOULD** be deactivated:

```
Timestamps = false
```

#### 4.2. Receiving Acknowledgments

For each received ACK that either a) carries SACK information, *\*or\** b) is a full ACK that terminates the current Fast Recovery procedure, *\*or\** c) is an acceptable ACK that is received immediately after a duplicate ACK, execute steps (A.1) to (A.4), otherwise skip to step (A.4).

- (A.1) If a) the ACK carries new SACK information, *\*and\** b) the SACK scoreboard is empty (i.e., the TCP sender has received no SACK information from the receiver), then the TCP sender **MUST** save the amount of current outstanding data:

```
FlightSizePrev = FlightSize
```

- (A.2) If the received ACK either a) cumulatively acknowledges at most SMSS bytes, *or* b) selectively acknowledges at most SMSS bytes in the sequence number space in the SACK scoreboard, then:

The TCP sender MUST execute steps (S.1) to (S.4)

- (A.3) If a) Timestamps == false *and* b) the received ACK carries a DSACK option [RFC2883] and the segment identified by the DSACK option can be marked according to step (A.1) to (A.4) of [RFC3708] as a valid duplicate, then:

The TCP sender MUST execute steps (D.1) to (D.3)

- (A.4) The TCP sender MUST terminate the processing of the ACK by this algorithm and MUST continue with the default processing of the ACK.

#### 4.3. Receiving Acknowledgment Closing Hole

- (S.1) If (a) the newly cumulatively or selectively acknowledged segment SEG is a retransmission *and* b) both equations Dsack == false and Timestamps == false hold, then the TCP sender MUST skip to step (A.4).

- (S.2) Compute the relative and absolute reordering extent ReorExtR, ReorExtA:

The TCP sender MUST execute steps (E.1) to (E.4)

- (S.3) If a) the newly acknowledged segment SEG was not retransmitted before *or* b) both equations Timestamps == true and Retrans\_TS[SEG.SEQ] > ACK.TSecr hold, i.e., the ACK acknowledges the original transmission and not a retransmission, then hand over the reordering extents to an additional reaction algorithm:

The TCP sender MUST execute step (P)

- (S.4) If a) the previous step (S.3) was not executed *and* b) both equations Dsack == true and Timestamps == false hold, save the reordering extents for the newly acknowledged segment SEG for at least two RTTs:

Samples[SEG.SEQ].ReorExtR = ReorExtR  
Samples[SEG.SEQ].ReorExtA = ReorExtA

- (S.5) If a) the newly acknowledged segment SEG was retransmitted before exactly once \*and\* b) both equations `Dsack == true` and `Timestamps == true` hold \*and\* c) `Retrans_TS[SEG.SEQ] == ACK.TSecr`, then save `FlightSizePrev` for this segment in order to be calculate the metrics in case a DSACK arrives, i.e. reordering delay is greater than RTT:

`Retrans_Dsack[SEG.SEQ] = FlightSizePrev`

#### 4.4. Receiving Duplicate Selective Acknowledgment

- (D.1) If no DSACK has been received so far, the sender MUST set:

`Dsack = true`

- (D.2) If a) the previous step (D.1) was not executed \*and\* a reordering extent was calculated for the segment SEG identified by the DSACK option, then the TCP sender MUST restore the values of the variables `ReorExtR` and `ReorExtA` and delete the corresponding entries in the data structure:

`ReorExtR = Samples[SEG.SEQ].ReorExtR`  
`ReorExtA = SAMPLES[SEG.SEQ].ReorExtA`

- (D.3) If a) step (D.1) was not executed \*and\* b) `FlightSizePrev` was saved in step (S.4) for the segment, then the TCP sender MUST calculate the reordering extent for the segment with the E series of steps by using the `FlightSizePrev` saved for this segment and afterwards delete the corresponding entries:

`FlightSizePrev_saved = Retrans_Dsack[SEG.SEQ]`

- (D.4) Hand the new reordering extents over to an additional reaction algorithm:

The TCP sender SHOULD execute step (P)

#### 4.5. Reordering Extent Computation

- (E.1) `SEG.SEQ` is the sequence number of the newly cumulatively or selectively acknowledged segment SEG.
- (E.2) `SND.FACK` is the highest either cumulatively or selectively acknowledged sequence number so far plus one.
- (E.3) The TCP sender MUST compute the absolute reordering extent `ReorExtA` as

$$\text{ReorExtA} = (\text{SND.FACK} - \text{SEG.SEQ}) / \text{SMSS}$$

- (E.4) The TCP sender MUST compute the relative reordering extent `ReorExtR` as

$$\text{ReorExtR} = \text{ReorExtA} * (\text{SMSS} / \text{FlightSizePrev})$$

#### 4.6. Retransmitting Segment

If the TCP Timestamps option [RFC1323] is used to detect packet reordering, the TCP sender must save the TCP Timestamps option of all retransmitted segments during Fast Recovery.

- (RET) If a) a segment `SEG` is retransmitted during Fast Recovery, \*and\* b) the equation `Timestamps = true` holds, the TCP sender MUST save the value of the `TSval` field of the retransmitted segment:

$$\text{Retrans\_TS}[\text{SEG.SEQ}] = \text{SEG.TSval}$$

#### 4.7. Placeholder for Response Algorithm

- (P) This is a placeholder for an additional reaction algorithm that takes further action using the results of this algorithm, for example, the adjustment of the `DupThresh` based on relative and absolute reordering extent `ReorExtR` and `ReorExtA`.

#### 4.8. Retransmission Timeout

The expiration of the retransmission timer should be interpreted as an indication of a change in path characteristics, and the TCP sender should consider all saved reordering extents as outdated and delete them.

- (RTO) If an retransmission timeout (RTO) occurs, a TCP sender SHOULD reset the following variables:

```
Samples = []
Retrans_TS = []
FlightSizePrev = 0
```

### 5. Protocol Steps in Detail

The reception of an ACK represents the starting point for the detection scheme above. For each received SACK, DSACK or acceptable ACK that prompts the TCP sender to enter the 'disorder' state, to remain in the 'disorder' state or to leave either the 'disorder' or 'recovery' states towards the 'open' state, steps (A.1) to (A.4) are

performed. All other received ACKs are not relevant for the detection of packet reordering and can be ignored. If the TCP sender changes from the 'open' to the 'disorder' state due to the reception of a duplicate ACK (i.e., the SACK scoreboard is empty and an ACK arrives carrying new SACK information), the current amount of outstanding data, FlightSize, is stored for the subsequent calculation of the relative reordering extent (step (A.1)).

Whenever a received acceptable ACK or SACK closes a hole in the sequence number space of the SACK scoreboard either partially or completely, this is an indication of packet reordering in the network (step (A.2)). The prerequisite for an accurate quantification of the reordering is that only one segment is newly acknowledged (maximum SMSS bytes of data). If more than one segment per ACK is acknowledged, either by reordering on the reverse path or the loss of ACKs, the order in which the segments have been received by the TCP receiver is no longer accurately determinable so that in this case a reordering extent is not calculated. Finally, if the received ACK carries a DSACK option that identifies a segment that was retransmitted only once, then this is sufficient to conclude reordering (step (A.3)), so that a previously calculated reordering extent can be passed to another algorithm (steps (D.3) and (P)).

With just the information provided by the ACK field or SACK information above SND.UNA, the TCP sender is unable to distinguish whether the ACK that finally acknowledges retransmitted data (either cumulatively or selectively) was sent in response to the original segment or a retransmission of the segment. This is described as the retransmission ambiguity problem in [KP87]. Therefore, the detection and quantification of reordering depends on other means to distinguish between acknowledgments for transmission and retransmission to detect if a retransmission was spurious. If neither a DSACK has been received (Dsack == false) so far nor the TCP Timestamps option has been enabled on connection establishment (Timestamps == false) then there is no possibility for the TCP sender to identify spurious retransmissions. Hence, the processing of the received ACK by the detection algorithm must be terminated for retransmitted segments (step (S.1)). Otherwise, if the segment that corresponds to the closed hole in the sequence number space of the SACK scoreboard has not been retransmitted or the retransmission can be identified by the Eifel detection algorithm [RFC3522] as a spurious retransmission, the previously calculated reordering extent is valid (step (S.2)) and an additional reaction algorithm can be executed (steps (S.3) and (P)).

For the use of the Eifel detection it is necessary to store the TCP Timestamps option of all retransmissions sent during Fast Recovery (step (Ret)). However, if the use of the Eifel detection algorithm

is not possible (Timestamps == false), the extent of a possible reordering is stored for the possibility of a subsequent arrival of a DSACK (step (P.4)). If no such DSACK is received in the next two round-trip times, the reordering extent is discarded. Since the DSACK extension is not negotiated during connection establishment [RFC2883], the reordering extent is only stored if a DSACK was previously received for the TCP connection (DSACK == true, step (D.1)).

Regardless of whether packet reordering is detected by using the SACK-based methodology, the DSACK-based methodology, or the TCP Timestamps option, quantification of the reordering will always be done when closing a hole in the sequence number space of the SACK scoreboard (step (A.2), step (P.2)). The only difference is the time of detection, which is in the case of DSACK-based methodology at least one RTT after the time of the quantification. The absolute reordering extent ReorExtA results from the number of segments in the SACK scoreboard between the sequence number of the newly acknowledged segment and the highest either cumulatively or selectively acknowledged sequence number so far plus one (SND.FACK) (step (E.3)).

In the case that the reordering delay is longer than RTT, the reordering can not be detected by timestamps or DSACK alone, but both algorithms are needed: when a packet is retransmitted, but no reordering could be detected when it was acknowledged, then it might be possible that a DSACK arrives for this packet. Then, the reordering extent was longer than RTT and the reordering extent has to be calculated at the point in time the DSACK arrives (step D.3). Therefore, we save the FlightSizePrev for a retransmitted segment when it is acked and no reordering is detected (step S.5).

It is worth noting that the absolute reordering extent includes all segments (bytes) between the closed hole and the highest acknowledged sequence number so far, i.e., it also includes segments (bytes) that are not selectively acknowledged. The reason is that if packet reordering is considered from a temporal perspective, it is irrelevant whether there are lost segments or not. The important fact is that the lost segments have been sent after the delayed segment and before the highest acknowledged segment, which is expressed by the metric. In step (E.4), the relative reordering extent ReorExtR is then calculated by the ratio between the absolute reordering extent ReorExtA and the amount of outstanding data stored by step (A.1).

## 6. Discussion of the Algorithm

The focus of the following discussion is on the quantification of reordering by the relative reordering extent and to elaborate on possible sources of error, which may lead to an inaccurate detection and quantification of reordering in the network.

### 6.1. Reasoning for the Relative Reordering Extent

A problem that arises with the way of quantifying reordering solemnly by the absolute reordering extent is that even in the presence of constant reordering, reordering extents may vary if the transmission rate of the TCP sender changes. Therefore, by using a DupThresh that directly reflects the measured reordering extent, spurious retransmissions cannot be fully avoided.

The following example illustrates this issue. Assume a path with a reordering probability of 1%, a reordering delay of 20 ms, and a bottleneck bandwidth of 3 Mb/s. Because segments that are delayed by reordering arrive 20 ms too late, the TCP receiver can receive a maximum of  $((20 * 3 * 10^3) / 8) = 7500$  bytes out-of-order before the reordered segment arrives. Hence, with a Sender Maximum Segment Size (SMSS) of 1460 bytes, the largest possible reordering extent is close to 5 segments. If the bottleneck bandwidth changes from 3 Mb/s to 4 Mb/s, the maximum reordering extent will increase to 7 segments, although the reordering delay remains constant.

This simple example shows that even with constant reordering, spurious retransmissions cannot be completely avoided if the DupThresh directly reflects the reordering extent. On the other hand, the reordering extent and the resulting DupThresh can sometimes also be much too high and do not correspond to the actual packet reordering present on the path. For example, a slow start overshoot [Hoe96], [MM96], [MSMO97] at the end of slow start might induce such a problem.

An obvious solution to the problem would be to quantify packet reordering not by calculating a reordering extent, but by using the reordering late time offset [RFC4737]. Since the reordering late time offset is not specified in segments but captures the difference between the expected and actual reception time of a reordered segment, this way of quantifying reordering is independent of the current transmission rate. Disadvantages of this approach are however a higher complexity and a worse integration into the TCP specification, since an implementation would require additional timers, whereas TCP itself is self-clocked.



## 6.2. Calculation of the Relative Reordering Extent

Generally, the characteristics of a relative reordering extent should be that if packet reordering on a path is constant in terms of rate and delay, the relative reordering extent should also be constant, regardless of the current transmission rate of the TCP sender. The scheme proposed in this document is to calculate the relative reordering by getting the ratio between absolute reordering (the amount of data the reordered segment was received too late) and the amount of outstanding data stored when TCP sender was entering the 'disorder' state (the maximum amount of data a reordered segment can be received too late). Therefore, the relative reordering extent expresses the portion of currently outstanding data that is selectively acknowledged before the reordered segment is cumulatively acknowledged. If the transmission rate changes, the absolute reordering extent changes as well, but together with the amount of outstanding data, and hence the relative reordering extent stays constant.

A characteristic of the calculation of the relative reordering extent on the basis of currently outstanding amount of data is that the FlightSize reflects the bandwidth-delay-product and not the transmission rate. As a consequence, the relative reordering extent is not independent of the RTT. If the RTT of the communication path changes, the amount of outstanding data changes as well, but the absolute reordering extent remains constant. Hence, the relative reordering extent adapts. In principle it is possible to design an algorithm to compute the relative reordering extent independently of the RTT and to reflect only the characteristics of packet reordering of the path. But since the calculation would be far from trivial and introducing more complexity, this is considered to be future research.

## 6.3. Persistent Reception of Selective Acknowledgments

Especially on paths with a high bandwidth-delay-product, it is possible that even with a minor packet reordering, several segments in a single window of data are delayed. If, in addition, the sequence numbers of those segments are widely spaced in the sequence number space and the delay caused by packet reordering is sufficiently high, this might lead to a constant reception of out-of-order data. Hence, for each received segment, regardless of whether a hole in the sequence number space of the receive window is closed or not, an ACK is sent that carries SACK information. From TCP sender's perspective, this persistent receiving of new SACK information leads to the situation that the TCP sender enters the 'disorder' state when receiving the first SACK and never leaves it

again during the connection lifetime if no segment is lost in between.

In case of the above reordering detection and quantification scheme, the persistent reception of SACK blocks causes the amount of outstanding data, which is stored when the TCP sender enters the 'disorder' state, to never be updated, since FlightSize is only saved in step (A.1) when the SACK scoreboard is empty. If the transmission rate of the TCP sender, and therefore also the maximum amount of data a reordered segment can be received too late, changes significantly during its stay in the 'disorder' state, the actual amount of reordering is not accurately determined by the relative reordering extent. A decrease of the transmission rate would result in an overestimation of the reordering extent and vice versa.

A simple solution to the problem would be to store the maximum offset in terms of sequence number space by which a reordered segment can be received too late only when entering the 'disorder' state, but individually for every potentially reordered segment, that is, for every hole in the sequence number space of the SACK scoreboard. (Note: The maximum offset in terms of sequence number space by which a reordered segment can be received too late is strictly speaking the amount of data that have been transmitted later than the reordered segment. This amount of data can only be expressed by FlightSize within the 'open' state and not within the 'disorder' state, since the cumulative ACK point may not advance).

The problem with this simple idea is that for a new hole in the SACK scoreboard, it is not possible to determine whether it is a result of packet reordering or loss, and therefore it results in increased memory usage (to store the amount of data for each hole). Additionally, the packet reordering would be inaccurately quantified if the transmission rate changes significantly for a short amount of time. For example, if the amount of outstanding data is low when entering the 'disorder' state is entered, the execution of Careful Extended Limited Transmit (as described in [I-D.zimmermann-tcpm-reordering-reaction] [RFC4653]) leads to a significant short-term change of the transmission rate. When the amount of data by which the reordering segment can be delayed is determined individually for every new hole, it leads to an overestimation of the relative reordering extent, since the maximum amount of data possible is 'artificially' reduced by Careful Extended Limited Transmit.

A solution to this problem is to store the maximum offset in terms of sequence number space by which a reordered segment can be received too late not for every segment individually (which does not guarantee an accurate calculation of the relative reordering extent) but only

sufficiently often, e.g., once per RTT. The identification of what frequency would be adequate, though, is neither trivial nor universally applicable, since a concrete solution depends on the transmission behavior of the used TCP in the 'disorder' state and whether it is more beneficial for an additional reordering response algorithm to over- or underestimate the packet reordering on the path. If, for example, TCP-aNCR [I-D.zimmermann-tcpm-reordering-reaction] is used as additional reordering response algorithm, the maximum offset in terms of sequence number space by which a reordered segment can be received too late is not only stored when entering the 'disorder' state but also updated every RTT (every cwnd worth of data transmitted without a loss) while the TCP sender stays in the 'disorder' state.

#### 6.4. Unreliable ACK reception

ACK loss and ACK reordering are a cause for inaccuracies in samples.

#### 6.5. Packet Duplication

Although the problem of packet duplication in today's Internet [JIDKT07], [MMMR08] is negligible, it may happen in rare cases that segments on the path to the TCP receiver are duplicated. If a segment is duplicated on the path, the first incoming segment causes the receiver to send either an acceptable ACK or a SACK, depending on whether the segment is the next expected one or not. Each subsequent identical segment then causes either a duplicate ACK or a DSACK, respectively, depending on whether the DSACK extension [RFC3708] is implemented or not.

If by a combination of packet loss and packet duplication the case occurs that a Fast Retransmit for a lost segment is duplicated on the path, the TCP sender is not able to distinguish this from packet reordering. The first received ACK closes a hole in the sequence number space of the SACK scoreboard, while the second received ACK is a valid DSACK. Although both cases are indistinguishable from a theoretical point of view, the TCP sender can take measures to ensure as far as possible that the DSACK received was not the result of packet duplication.

For this purpose, step (A.3) of the above detection method checks via the steps (A.1) to (A.4) of [RFC3708] whether the segment identified by the DSACK option is marked as a valid duplicate. Unfortunately, the steps of [RFC3708] do not check that more DSACKs have been received than retransmissions have been sent, which is a characteristic of suffering both packet reordering and packet duplication at the same time. By simply counting the received

DSACKs, for example, as additional step (A.5) in [RFC3708], this corner case can be covered as well.

## 7. Related Work

Because of retransmission ambiguity problem [KP87], which describes TCP sender's inability to distinguish whether the first acceptable ACK that arrives after a retransmit was sent in response to the original transmit or the retransmit, two different approaches can generally be taken to detect and quantify packet reordering. First, for transmissions (non-retransmitted segments), the detection is usually conducted by detecting a closed hole in sequence number space of the SACK scoreboard. Second, for retransmissions, the detection of packet reordering is accompanied by the detection of the spurious Fast Retransmits.

Within the IETF, several proposals have been published in the RFC series to detect and quantify packet reordering. With [RFC4737] the IPPM Working Group [IPPM] defines several metrics to evaluate whether a network path has maintained packet order on a packet-by-packet basis. [RFC4737] gives concrete, well-defined metrics, along with a methodology for applying the metric to a generic packet stream. The metric discussed in this document has a different purpose from the IPPM metrics; this document discusses a TCP specific reordering metric calculated on the TCP sender's SACK scoreboard.

Besides the IPPM work, several other proposals have been developed to detect spurious retransmissions with TCP. The Eifel detection algorithm [RFC3522] uses the TCP Timestamps option to determine whether the ACK for a given retransmit is for the original transmission or a retransmission. The F-RTO scheme [RFC5682] slightly alters TCP's sending pattern immediately following a retransmission timeout to indicate whether the retransmitted segment was needed. Finally, the DSACK-based algorithm [RFC3708] uses the TCP SACK option [RFC2018] with the DSACK extension [RFC2883] to identify unnecessary retransmissions. The mechanism for detecting packet reordering outlined in this document rely on the detection schemes of those documents (except F-RTO that only works for spurious retransmits triggered by TCP's retransmission timer), although they do not provide metrics for the reordering extent whereas the algorithm described in this document does.

RR-TCP [ZKFP03] describes a reordering detection and quantification scheme that is also based on holes in the sequence number space of the SACK scoreboard and the reception of DSACKs. For every hole in the SACK scoreboard, RR-TCP calculates a reordering extent. If the segment was retransmitted before an ACK was received, it waits for a DSACK that proves that the segment was spuriously retransmitted. The

reordering sample in such a case is the mean between the sample calculated due to the hole in the sequence number space and the sample calculated in responding to the received DSACK.

The Linux kernel [Linux] implements a reordering detection based on SACK, DSACK and TCP Timestamps option as well. The detection and quantification of non-retransmitted segments with SACK or for retransmitted segments with TCP Timestamps option operates much like the scheme described in this document, with the exception of the DSACK detection. First, Linux does not store any information (e.g., reordering extent) below the cumulative ACK point, so that DSACKs below the cumulative ACK point are ignored (for the purpose for reordering quantification). Second, Linux also does not store any information about a possible reordering event when a hole in the sequence number space of the SACK scoreboard is closed. Therefore, for a DSACK reporting a duplicate above the cumulative ACK, Linux needs to approximate the reordering on arrival of a DSACK by the distance between the DSACK and the highest selectively acknowledged segment.

## 8. IANA Considerations

This memo includes no request to IANA.

## 9. Security Considerations

The described algorithm neither improves nor degrades the current security of TCP, since this document only detects and quantifies reordering and does not change the TCP behavior. General security considerations for SACK based loss recovery are outlined in [RFC6675].

## 10. Acknowledgments

The authors thank the flowgrind [Flowgrind] authors and contributors for their performance measurement tool, which give us a powerful tool to analyze TCP's congestion control and loss recovery behavior in detail.

## 11. References

### 11.1. Normative References

[I-D.zimmermann-tcpm-reordering-reaction]  
Zimmermann, A., Schulte, L., Wolff, C., and A. Hannemann,  
"An adaptive Robustness of TCP to Non-Congestion Events",  
draft-zimmermann-tcpm-reordering-reaction-01 (work in  
progress), November 2013.

- [MM96] Mathis, M. and J. Mahdavi, "Forward Acknowledgement: Refining TCP Congestion Control", ACM SIGCOMM 1996 Proceedings, in ACM Computer Communication Review 26 (4), pp. 281-292, October 1996.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
- [RFC1323] Jacobson, V., Braden, B., and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2883] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, July 2000.
- [RFC3522] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC 3522, April 2003.
- [RFC3708] Blanton, E. and M. Allman, "Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions", RFC 3708, February 2004.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.

## 11.2. Informative References

- [BA02] Blanton, E. and M. Allman, "On Making TCP More Robust to Packet Reordering", ACM Computer Communication Review vol.32, no. 1, pp. 20-30, January 2002.
- [BPS99] Bennett, J., Partridge, C., and N. Shectman, "Packet reordering is not pathological network behavior", IEEE/ACM Transactions on Networking vol. 7, no. 6, pp. 789-798, December 1999.
- [BS02] Bellardo, J. and S. Partridge, "Measuring Packet Reordering", Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement (IMW'02) pp. 97-105, November 2002.

- [Flowgrind] "Flowgrind Home Page", <<http://www.flowgrind.net>>.
- [GPL04] Gharai, L., Perkins, C., and T. Lehman, "Packet Reordering, High Speed Networks and Transport Protocol Performance", Proceedings of the 13th IEEE International Conference on Computer Communications and Networks (ICCCN'04) pp. 73-78, October 2004.
- [Hoe96] Hoe, J., "Improving the Start-up Behavior of a Congestion Control Scheme for TCP", Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'96) pp. 270-280, August 1996.
- [IPPM] "IP Performance Metrics (IPPM) Working Group", <<http://www.ietf.org/html.charters/ippm-charter.html>>.
- [JIDKT07] Jaiswal, S., Iannaccone, G., Diot, C., Kurose, J., and D. Towsley, "Measurement and Classification of Out-of-Sequence Packets in a Tier-1 IP Backbone", IEEE/ACM Transactions on Networking vol. 15, no. 1, pp. 54-66, February 2007.
- [KP87] Karn, P. and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols", ACM SIGCOMM Computer Communication Review vol. 17, no. 5, pp. 2-7, November 1987.
- [Linux] "The Linux Project", <<http://www.kernel.org>>.
- [MMMR08] Mellia, M., Meo, M., Muscariello, L., and D. Rossi, "Passive analysis of TCP anomalies", Computer Networks vol. 52, no. 14, pp. 2663-2676, October 2008.
- [MSMO97] Mathis, M., Semke, J., Mahdavi, J., and T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", ACM SIGCOMM Computer Communication Review vol. 27, no. 3, pp. 67-82, July 1997.
- [Pax97] Paxson, V., "End-to-End Internet Packet Dynamics", IEEE/ACM Transactions on Networking vol. 7, no.3, pp. 277-292, June 1997.
- [RFC0896] Nagle, J., "Congestion control in IP/TCP internetworks", RFC 896, January 1984.

- [RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.
- [RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.
- [RFC4653] Bhandarkar, S., Reddy, A., Allman, M., and E. Blanton, "Improving the Robustness of TCP to Non-Congestion Events", RFC 4653, August 2006.
- [RFC4737] Morton, A., Ciavattone, L., Ramachandran, G., Shalunov, S., and J. Perser, "Packet Reordering Metrics", RFC 4737, November 2006.
- [RFC5682] Sarolahti, P., Kojo, M., Yamamoto, K., and M. Hata, "Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP", RFC 5682, September 2009.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.
- [RFC6582] Henderson, T., Floyd, S., Gurtov, A., and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 6582, April 2012.
- [RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, August 2012.
- [ZKFP03] Zhang, M., Karp, B., Floyd, S., and L. Peterson, "RR-TCP: A Reordering-Robust TCP with DSACK", Proceedings of the 11th IEEE International Conference on Network Protocols (ICNP'03) pp. 95-106, November 2003.
- [ZM04] Zhou, X. and P. Miegheem, "Reordering of IP Packets in Internet", Lecture Notes in Computer Science vol. 3015, pp. 237-246, April 2004.

#### Appendix A. Changes from previous versions of the draft

This appendix should be removed by the RFC Editor before publishing this document as an RFC.



## A.1. Changes from draft-zimmermann-tcpm-reordering-detection-01

- o Moved reasoning for relative reordering extent to discussion
- o Extended algorithm for calculation of reordering extents greater than RTT (steps C.2, S.5 and D.3)
- o Remove reverse-path reordering from intro

## A.2. Changes from draft-zimmermann-tcpm-reordering-detection-00

- o Improved the wording throughout the document.
- o Replaced and updated some references.

## Authors' Addresses

Alexander Zimmermann  
NetApp, Inc.  
Sonnenallee 1  
Kirchheim 85551  
Germany

Phone: +49 89 900594712  
Email: alexander.zimmermann@netapp.com

Lennart Schulte  
Aalto University  
Otakaari 5 A  
Espoo 02150  
Finland

Phone: +358 50 4355233  
Email: lennart.schulte@aalto.fi

Carsten Wolff  
credativ GmbH  
Hohenzollernstrasse 133  
Moenchengladbach 41061  
Germany

Phone: +49 2161 4643 182  
Email: carsten.wolff@credativ.de

Arnd Hannemann  
credativ GmbH  
Hohenzollernstrasse 133  
Moenchengladbach 41061  
Germany

Phone: +49 2161 4643 134  
Email: arnd.hannemann@credativ.de

TCP Maintenance and Minor Extensions (TCPM) WG  
Internet-Draft  
Obsoletes: 4653 (if approved)  
Intended status: Experimental  
Expires: May 14, 2015

A. Zimmermann  
NetApp, Inc.  
L. Schulte  
Aalto University  
C. Wolff  
A. Hannemann  
credativ GmbH  
November 10, 2014

Making TCP Adaptively Robust to Non-Congestion Events  
draft-zimmermann-tcpm-reordering-reaction-02

Abstract

This document specifies an adaptive Non-Congestion Robustness (aNCR) mechanism for TCP. In the absence of explicit congestion notification from the network, TCP uses only packet loss as an indication of congestion. One of the signals TCP uses to determine loss is the arrival of three duplicate acknowledgments. However, this heuristic is not always correct, notably in the case when paths reorder packets. This results in degraded performance.

TCP-aNCR is designed to mitigate this performance degradation by adaptively increasing the number of duplicate acknowledgments required to trigger loss recovery, based on the current state of the connection, in an effort to better disambiguate true segment loss from segment reordering. This document specifies the changes to TCP and TCP-NCR (on which this specification is build on) and discusses the costs and benefits of these modifications.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 14, 2015.

## Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Terminology . . . . .	6
3. Basic Concept . . . . .	7
4. Appropriate Detection and Quantification Algorithms . . . . .	7
5. The TCP-aNCR Algorithm . . . . .	8
5.1. Initialization during Connection Establishment . . . . .	8
5.2. Initializing Extended Limited Transmit . . . . .	9
5.3. Executing Extended Limited Transmit . . . . .	10
5.4. Terminating Extended Limited Transmit . . . . .	11
5.5. Entering Loss Recovery . . . . .	13
5.6. Reordering Extent . . . . .	13
5.7. Retransmission Timeout . . . . .	13
6. Protocol Steps in Detail . . . . .	14
7. Discussion of TCP-aNCR . . . . .	16
7.1. Variable Duplicate Acknowledgment Threshold . . . . .	16
7.2. Relative Reordering Extent . . . . .	17
7.3. Reordering during Slow Start . . . . .	18
7.4. Preventing Bursts . . . . .	18
7.5. Persistent receiving of Selective Acknowledgments . . . . .	19
8. Interoperability Issues . . . . .	21
8.1. Early Retransmit . . . . .	21
8.2. Congestion Window Validation . . . . .	21
8.3. Reactive Response to Packet Reordering . . . . .	22
8.4. Buffer Auto-Tuning . . . . .	22
9. Related Work . . . . .	23
10. IANA Considerations . . . . .	24
11. Security Considerations . . . . .	24
12. Acknowledgments . . . . .	25
13. References . . . . .	25
13.1. Normative References . . . . .	25
13.2. Informative References . . . . .	26

Appendix A. Changes from previous versions of the draft . . . .	28
A.1. Changes from draft-zimmermann-tcpm-reordering-reaction-01	28
A.2. Changes from draft-zimmermann-tcpm-reordering-reaction-00	28
Authors' Addresses . . . . .	28

## 1. Introduction

One strength of the Transmission Control Protocol (TCP) [RFC0793] lies in its ability to adjust its sending rate according to the perceived congestion in the network [RFC5681]. In the absence of explicit notification of congestion from the network, TCP uses segment loss as an indication of congestion (i.e., assuming queue overflow). A TCP receiver sends cumulative acknowledgments (ACKs) indicating the next sequence number expected from the sender for arriving segments [RFC0793]. When segments arrive out of order, duplicate ACKs are generated. As specified in [RFC5681], a TCP sender uses the arrival of three duplicate ACKs as an indication of segment loss. The TCP sender retransmits the segment assumed lost and reduces the sending rate, based on the assumption that the loss was caused by resource contention on the path. The TCP sender does not assume loss on the first or second duplicate ACK, but waits for three duplicate ACKs to account for minor packet reordering. However, the use of this constant threshold of duplicate ACKs leads to performance degradation if the extent of the packet reordering in the network increases [RFC4653].

Whenever interoperability with the TCP congestion control and loss recovery standard [RFC5681] is a prerequisite, increasing the duplicate acknowledgment threshold (DupThresh) is the method of choice to a priori prevent any negative impact - in particular, a spurious Fast Retransmit and Fast Recovery phase - that packet reordering has on TCP. However, this procedure also delays a Fast Retransmit by increasing the DupThresh, and therefore has costs and risks, too. According to [ZKFP03], these are: (1) a delayed response to congestion in the network, (2) a potential expiration of the retransmission timer, and (3) a significant increase in the end-to-end delay for lost segments.

In the current TCP standard, congestion control and loss recovery are tightly coupled: when the oldest outstanding segment is declared lost, a retransmission is triggered, and the sending rate is reduced on the assumption that the loss is due to resource contention [RFC5681]. Therefore, any change to DupThresh causes not only a change to the loss recovery, but also to the congestion control response. TCP-NCR [RFC4653] addresses this problem by defining two extensions to TCP's Limited Transmit [RFC3042] scheme: Careful and Aggressive Extended Limited Transmit.

The first variant of the two, Careful Limited Transmit, sends one previously unsent segment in response to duplicate acknowledgments for every two segments that are known to have left the network. This effectively halves the sending rate, since normal TCP operation sends one new segment for every segment that has left the network. Further, the halving starts immediately and is not delayed until a retransmission is triggered. In the case of packet reordering (i.e., not segment loss), TCP-NCR restores the congestion control state to its previous state after the event.

The second variant, Aggressive Limited Transmit, transmits one previously unsent data segment in response to duplicate acknowledgments for every segment known to have left the network. With this variant, while waiting to disambiguate the loss from a reordering event, ACK-clocked transmission continues at roughly the same rate as before the event started. Retransmission and the sending rate reduction happen per [RFC5681] [RFC6675], albeit after a delay caused by the increased DupThresh. Although this approach delays legitimate rate reductions (possibly slightly, and temporarily aggravating overall congestion on the network), the scheme has the advantage of not reducing the transmission rate in the face of packet reordering.

A basic requirement for preventing an avoidable expiration of the retransmission timer is to generally ensure that an increased DupThresh can potentially be reached in time so that Fast Retransmit is triggered and Fast Recovery is completed before the RTO expires. Simply increasing DupThresh before retransmitting a segment can make TCP brittle to packet or ACK loss, since such loss reduces the number of duplicate ACKs that will arrive at the sender from the receiver. For instance, if cwnd is 10 segments and one segment is lost, a DupThresh of 10 will never be met, because duplicate ACKs corresponding to at most 9 segments will arrive at the sender. To mitigate this issue, the TCP-NCR [RFC4653] modification makes two fundamental changes to the way [RFC5681] [RFC6675] currently operates.

First, as mentioned above, TCP-NCR [RFC4653] extends TCP's Limited Transmit [RFC3042] scheme to allow for the sending of new data segment while the TCP sender stays in the 'disorder' state and disambiguate loss and reordering. This new data serves to increase the likelihood that enough duplicate ACKs arrive at the sender to trigger loss recovery, if it is appropriate. Second, DupThresh is increased from the current fixed value of three [RFC5681] to a value indicating that approximately a congestion window's worth of data has left the network. Since cwnd represents the amount of data a TCP sender can transmit in one round-trip time (RTT), this corresponds to

approximately the largest amount of time a TCP sender can wait before the costly retransmission timeout may be triggered.

Of vital importance is that TCP-NCR [RFC4653] holds DupThresh not constant, but dynamically adjusts it on each SACK to the current amount of outstanding data, which depends not only on the congestion window, but also on the receiver's advertised window. Thus, it is guaranteed that the outstanding data generates a sufficient number of duplicate ACKs for reaching DupThresh and a transition to the 'recovery' state. This is important in cases where there is no new data available to send.

Regarding the problem of packet reordering, TCP-NCR's [RFC4653] decision of waiting to receive notice that cwnd bytes have left the network before deciding whether the root cause is loss or reordering is essentially a trade-off between making the best decision regarding the cause of the duplicate ACKs and responsiveness, and represents a good compromise between avoiding spurious Fast Retransmits and avoiding unnecessary RTOs. On the other hand, if there is no visible packet reordering on the network path - which today is the rule and not the exception - or the delay caused by the reordering is very low, delaying Fast Retransmit is unnecessary in the case of congestion, and data is delivered to the application up to one RTT later. Especially for delay-sensitive applications, such as a terminal session over SSH, this is generally undesirable. By dynamically adapting DupThresh not only to the amount of outstanding data but also to the perceived packet reordering on the network path, this issue can be offset. This is the key idea behind the TCP-aNCR algorithm.

This document specifies a set of TCP modifications to provide an adaptive Non-Congestion Robustness (aNCR) mechanism for TCP. The TCP-aNCR modifications lend themselves to incremental deployment. Only the TCP implementation on the sender side requires modification. The changes themselves are modest. TCP-aNCR is built on top of the TCP Selective Acknowledgments Option [RFC2018] and the SACK-based loss recovery scheme given in [RFC6675] and represents an enhancement of the original TCP-NCR mechanism [RFC4653]. Currently, TCP-aNCR is an independent approach of making TCP more robust to packet reordering. It is not clear if upcoming versions of this draft TCP-aNCR will obsolete TCP-NCR or not.

It should be noted that the TCP-aNCR algorithm in this document could be easily adapted to the Stream Control Transmission Protocol (SCTP) [RFC2960], since SCTP uses congestion control algorithms similar to TCP (and thus has the same reordering robustness issues).

The remainder of this document is organized as follows. Section 3 provides a high-level description of the TCP-aNCR mechanism. Section 4 defines TCP-aNCR's requirements for an appropriate detection and quantification algorithm. Section 5 specifies the TCP-aNCR algorithm and Section 6 discusses each step of the algorithm in detail. Section 7 provides a discussion of several design decisions behind TCP-aNCR. Section 8 discusses interoperability issues related to introducing TCP-aNCR. Finally, related work is presented in Section 9 and security concerns in Section 11.

## 2. Terminology

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 [RFC2119].

The reader is expected to be familiar with the TCP state variables described in [RFC0793] (SND.NXT), [RFC5681] (cwnd, rwnd, ssthresh, FlightSize, IW), [RFC6675] (pipe, DupThresh, SACK scoreboard), and [RFC6582] (recover). Further, the term 'acceptable acknowledgment' is used as defined in [RFC0793]. That is, an ACK that increases the connection's cumulative ACK point by acknowledging previously unacknowledged data. The term 'duplicate acknowledgment' is used as defined in [RFC6675], which is different from the definition of duplicate acknowledgment in [RFC5681].

This specification defines the four TCP sender states 'open', 'disorder', 'recovery', and 'loss' as follows. As long as no duplicate ACK is received and no segment is considered lost, the TCP sender is in the 'open' state. Upon the reception of the first consecutive duplicate ACK, TCP will enter the 'disorder' state. After receiving DupThresh duplicate ACKs, the TCP sender switches to the 'recovery' state and executes standard loss recovery procedures like Fast Retransmit and Fast Recovery [RFC5681]. Upon a retransmission timeout, the TCP sender enters the 'loss' state. The 'recovery' state can only be reached by a transition from the 'disorder' state, the 'loss' state can be reached from any other state.

The following specification depends on the standard TCP congestion control and loss recovery algorithms and the SACK-based loss recovery scheme given in [RFC5681], respectively [RFC6675]. The algorithm presents an enhancement of TCP-NCR [RFC4653]. The reader is assumed to be familiar with the algorithms specified in these documents.



### 3. Basic Concept

The general idea behind the TCP-aNCR algorithm is to extend the TCP-NCR algorithm [RFC4653], so that - based on an appropriate packet reordering detection and quantification algorithm (see Section 4) - TCP congestion control and loss recovery [RFC5681] is adaptively adjusted to the actual perceived packet reordering on the network path.

TCP-NCR [RFC4653] increases DupThresh from the current fixed value of three duplicate ACKs [RFC5681] to approximately until a congestion window of data has left the network. Since cwnd represents the amount of data a TCP sender can transmit in one RTT, the choice to trigger a retransmission only after a cwnd's worth of data is known to have left the network represents roughly the largest amount of time a TCP sender can wait before the RTO may be triggered. The approach chosen in TCP-aNCR is to take TCP-NCR's DupThresh as an upper bound for an adjustment of the DupThresh that is adaptive to the actual packet reordering on the network path.

Using TCP-NCR's DupThresh as an upper bound decouples the avoidance of spurious Fast Retransmits from the avoidance of unnecessary retransmission timeouts. Therefore, the adaptive adjustment of the DupThresh to current perceived packet reordering can be conducted without taking any retransmission timeout avoidance strategy into account. This independence allows TCP-aNCR to quickly respond to perceived packet reordering by setting its DupThresh so that it always corresponds to the minimum of the maximum possible (TCP-NCR's DupThresh) and the maximum measured reordering extent since the last RTO. The reordering extent used by TCP-aNCR is by itself not a static absolute reordering extent, but a relative reordering extent (see Section 4).

### 4. Appropriate Detection and Quantification Algorithms

If the TCP-aNCR algorithm is implemented at the TCP sender, it MUST be implemented together with an appropriate packet reordering detection and quantification algorithm that is specified in a standards track or experimental RFC.

Designers of reordering detection algorithms who want their algorithms to work together with the TCP-aNCR algorithm SHOULD reuse the variable 'ReorExtR' (relative reordering extent) with the semantics and defined values specified in [I-D.zimmermann-tcpm-reordering-detection]. A 'ReorExtR' given by the detection algorithm holds a value ranging from 0 to 1 which holds the new measured reordering sample as a fraction of the data in

flight. TCP-aNCR then saves this new fraction if it is greater than the current value.

## 5. The TCP-aNCR Algorithm

When both the Nagle algorithm [RFC0896] [RFC1122] and the TCP Selective Acknowledgment Option [RFC2018] are enabled for a connection, a TCP sender MAY employ the following TCP-aNCR algorithm to dynamically adapt TCP's congestion control and loss recovery [RFC5681] to the currently perceived packet reordering on the network path.

Without the Nagle algorithm, there is no straightforward way to accurately calculate the number of outstanding segments in the network (and, therefore, no good way to derive an appropriate DupThresh) without adding state to the TCP sender. A TCP connection that does not use the Nagle algorithm SHOULD NOT use TCP-aNCR. The adaptation of TCP-aNCR to an implementation that carefully tracks the sequence numbers transmitted in each segment is considered future work.

A necessary prerequisite for TCP-aNCR's adaptability is that a TCP sender has enabled an appropriate detection and quantification algorithm that complies with the requirements defined in Section 4. If such an algorithm is either non-existent or not used, the behavior of TCP-aNCR is completely analogous to the TCP-NCR algorithm as defined in [RFC4653]. If a TCP sender does implement TCP-aNCR, the implementation MUST follow the various specifications provided in Sections 5.1 to 5.7.

### 5.1. Initialization during Connection Establishment

After the completion of the TCP connection establishment, the following state constants and variables MUST be initialized in the TCP transmission control block for the given TCP connection:

- (C.1) Depending on which variant of Extended Limited Transmit should be executed, the constant LT\_F MUST be initialized as follows.  
For Careful Extended Limited Transmit:

LT\_F = 2/3

For Aggressive Extended Limited Transmit:

LT\_F = 1/2

This constant reflects the fraction of outstanding data (including data sent during Extended Limited Transmit) that

must be SACKed before a retransmission is at the latest triggered.

- (C.2) If TCP-aNCR should adaptively adjust the DupThresh to the current perceived packet reordering on the network path, then the variable 'ReorExtR', which stores the maximum relative reordering extent, MUST initialized as:

ReorExtR = 0

Otherwise the dynamically adaptation of TCP-aNCR SHOULD be disabled by setting

ReorExtR = -1

A relative reordering extent of 0 results in the standard DupThresh of three duplicate ACKs, as defined in [RFC5681]. A fixed relative reordering extent of -1 results in the TCP-NCR behavior from [RFC4653].

## 5.2. Initializing Extended Limited Transmit

If the SACK scoreboard is empty upon the receipt of a duplicate ACK (i.e., the TCP sender has received no SACK information from the receiver), a TCP sender MUST enter Extended Limited Transmit by initialize the following five state variables in the TCP Transmission Control Block:

- (I.1) The TCP sender MUST save the current outstanding data:

FlightSizePrev = FlightSize

- (I.2) The TCP sender MUST save the highest sequence number transmitted so far:

recover = SND.NXT - 1

Note: The state variable 'recover' from [RFC6582] can be reused, since NewReno TCP uses 'recover' at the initialization of a loss recovery procedure, whereas TCP-aNCR uses 'recover' \*before\* loss recovery.

- (I.3) The TCP sender MUST initialize the variable 'skipped' that tracks the number of segments for which an ACK does not trigger a transmission during Careful Limited Transmit:

skipped = 0

During Aggressive Limited Transmit, 'skipped' is not used.

- (I.4) The TCP sender MUST set DupThresh based on the current FlightSize:

$$\text{DupThresh} = \max (\text{LT\_F} * (\text{FlightSize} / \text{SMSS}), 3)$$

The lower bound of DupThresh = 3 is kept from [RFC5681] [RFC6675].

- (I.5) If (ReorExtR != -1) holds, then the TCP sender MUST set DupThresh based on the relative reordering extent 'ReorExtR':

$$\begin{aligned} \text{DupThresh} = \\ \max (\min (\text{DupThresh}, \\ \text{ReorExtR} * (\text{FlightSizePrev} / \text{SMSS})), 3) \end{aligned}$$

In addition to the above steps, the incoming ACK MUST be processed with the (E) series of steps in Section 5.3.

### 5.3. Executing Extended Limited Transmit

On each ACK that a) arrives after TCP-aNCR has entered the Extended Limited Transmit phase (as outlined in Section 5.2) \*and\* b) carries new SACK information, \*and\* c) does \*not\* advance the cumulative ACK point, the TCP sender MUST use the following procedure.

- (E.1) The TCP sender MUST update the SACK scoreboard and uses the SetPipe() procedure from [RFC6675] to set the 'pipe' variable (which represents the number of bytes still considered "in the network"). Note: the current value of DupThresh MUST be used by SetPipe() to produce an accurate assessment of the amount of data still considered in the network.
- (E.2) The TCP sender MUST initialize the variable 'burst' that tracks the number of segments that can at most be sent per ACK to the size of the Initial Window (IW) [RFC5681]:

$$\text{burst} = \text{IW}$$

- (E.3) If a) (cwnd - pipe - skipped >= 1 \* SMSS) holds, \*and\* b) the receive window (rwnd) allows to send SMSS bytes of previously unsent data, \*and\* c) there are SMSS bytes of previously unsent data available for transmission, then the TCP sender MUST transmit one segment of SMSS bytes. Otherwise, the TCP sender MUST skip to step (E.7).

- (E.4) The TCP sender MUST increment 'pipe' by SMSS bytes and MUST decrement 'burst' by SMSS bytes to reflect the newly transmitted segment:

```
pipe = pipe + SMSS
burst = burst - SMSS
```

- (E.5) If Careful Limited Transmit is used, 'skipped' MUST be incremented by SMSS bytes to ensure that the next SMSS bytes of SACKed data processed do not trigger a Limited Transmit transmission.

```
skipped = skipped + SMSS
```

- (E.6) If (burst > 0) holds, the TCP sender MUST return to step (E.3) to ensure that as many bytes as appropriate are transmitted. Otherwise, if more than IW bytes were SACKed by a single ACK, the TCP sender MUST skip to step (E.7). The additional amount of data becomes available again by the next received duplicate ACK and the re-execution of SetPipe().

- (E.7) The TCP sender MUST save the maximum amount of data that is considered to have been in the network during the last RTT:

```
pipe_max = max (pipe, pipe_max)
```

- (E.8) The TCP sender MUST set DupThresh based on the current FlightSize:

```
DupThresh = max (LT_F * (FlightSize / SMSS), 3)
```

The lower bound of DupThresh = 3 is kept from [RFC5681] [RFC6675].

- (E.9) If (ReorExtR != -1) holds, then the TCP sender MUST set DupThresh based on the relative reordering extent 'ReorExtR':

```
DupThresh =
    max (min (DupThresh,
              ReorExtR * (FlightSizePrev / SMSS)), 3)
```

#### 5.4. Terminating Extended Limited Transmit

On the receipt of a duplicate ACK that a) arrives after TCP-aNCR has entered the Extended Limited Transmit phase (as outlined in Section 5.2) \*and\* b) advances the cumulative ACK point, the TCP sender MUST use the following procedure.

The arrival of an acceptable ACK that advances the cumulative ACK point while in Extended Limited Transmit, but before loss recovery is triggered, signals that a series of duplicate ACKs was caused by reordering and not congestion. Therefore, Extended Limited Transmit will be either terminated or re-entered.

(T.1) If the received ACK extends not only the cumulative ACK point, but *\*also\** carries new SACK information (i.e., the ACK is both an acceptable ACK and a duplicate ACK), the TCP sender *MUST* restart Extended Limited Transmit and *MUST* go to step (T.2). Otherwise, the TCP sender *MUST* terminate it and *MUST* skip to step (T.3).

(T.2) If the Cumulative Acknowledgment field of the received ACK covers more than 'recover' (i.e.,  $\text{SEG.ACK} > \text{recover}$ ), Extended Limited Transmit has transmitted one cwnd worth of data without any losses and the TCP sender *MUST* update the following state variables by

```
FlightSizePrev = pipe_max  
pipe_max = 0
```

and *MUST* go to step (I.2) to re-start Extended Limited Transmit. Otherwise if  $(\text{SEG.ACK} \leq \text{recover})$  holds, the TCP sender *MUST* go to step (I.3). This ensures that in the event of a loss the cwnd reduction is based on a current value of FlightSizePrev.

The following steps are executed only if the received ACK does *\*not\** carry SACK information. Extended Limited Transmit will be terminated.

(T.3) A TCP sender *MUST* set ssthresh to:

```
ssthresh = max (cwnd, ssthresh)
```

This step provides TCP-aNCR with a sense of "history". If the next step (T.4) reduces the congestion window, this step ensures that TCP-aNCR will slow-start back to the operating point that was in effect before Extended Limited Transmit.

(T.4) A TCP sender *MUST* reset cwnd to:

```
cwnd = FlightSize + SMSS
```

This step ensures that cwnd is not significantly larger than the amount of data outstanding, a situation that would cause a line rate burst.

- (T.5) A TCP is now permitted to transmit previously unsent data as allowed by `cwnd`, `FlightSize`, application data availability, and the receiver's advertised window.

#### 5.5. Entering Loss Recovery

The receipt of an ACK that results in deeming the oldest outstanding segment is lost via the algorithms in [RFC6675] terminates Extended Limited Transmit and initializes the loss recovery according to [RFC6675]. One slight change to either [RFC6675], or, if Proportional Rate Reduction (PRR) algorithm is used, to [RFC6937] MUST be made, however.

- (Ret) If the PRR algorithm is used to calculate how many bytes should be sent in response to each ACK, the initialization of 'RecoverFS' in Section 3 of [RFC6937] MUST be changed to:

`RecoverFS = FlightSizePrev`

Otherwise, if the standard Fast Recovery algorithm is used, step (4.2) of [RFC6675] MUST be changed in Section 5 to:

`ssthresh = cwnd = (FlightSizePrev / 2)`

This change ensures that the congestion control modifications are made with respect to the amount of data in the network before `FlightSize` was increased by Extended Limited Transmit.

Once the algorithm in [RFC6675] takes over from Extended Limited Transmit, the `DupThresh` value MUST be held constant until the loss recovery phase terminates.

#### 5.6. Reordering Extent

Whenever the additional detection and quantification algorithm (see Section 4) detects and quantifies a new reordering event, the TCP sender MUST update the state variable 'ReorExtR'.

- (Ext) Let 'ReorExtR\_New' the newly determined relative reordering extent:

`ReorExtR = min (max (ReorExtR, ReorExtR_New), 1)`

#### 5.7. Retransmission Timeout

The expiration of the retransmission timer SHOULD be interpreted as an indication of a path characteristics change, and the TCP sender SHOULD reset `DupThresh` to the default value of three.

(RTO) If an RTO occurs and (`ReorExtR != -1`) (i.e. TCP-aNCR is used and not TCP-NCR), then a TCP sender SHOULD reset 'ReorExtR':

`ReorExtR = 0`

## 6. Protocol Steps in Detail

Upon the receipt of the first duplicate ACK in the 'open' state (the SACK scoreboard is empty), the TCP sender starts to execute TCP-aNCR by entering the 'disorder' state and the initialization of Extended Limited Transmit. First, the TCP sender saves the current amount of outstanding data as well as the highest sequence number transmitted so far (`SND.NXT - 1`) (steps (I.1) and (I.2)). In addition, if the TCP connection uses the careful variant of the Extended Careful Limited Transmit (step (C.1)), the 'skipped' variable, which tracks the number of segments for which an ACK does not trigger a transmission during Careful Limited Transmit, is initialized with zero (step (I.3)). The last step during the initialization is the determination of `DupThresh`. Depending on whether TCP-aNCR has been configured during the connection establishment to adaptively adjust to the currently perceived packet reordering on the path (step (C.2)), `DupThresh` is either determined exclusively based on the current `FlightSize` (as TCP-NCR [RFC4653] does) or, in addition, also based on the relative extent reordering (steps (I.4) and (I.5)).

Depending on which variant of Extended Limited Transmit should be executed, the constant `LT_F` must be set accordingly (step (C.1)). This constant reflects the fraction of outstanding data (including data sent during Extended Limited Transmit) that must be SACKed before a retransmission is triggered at the latest (which is the case when a `DupThresh` that is based on relative reordering extent is larger than TCP-NCR's `DupThresh`). Since Aggressive Limited Transmit sends a new segment for every segment known to have left the network, a total of approximately `cwnd` segments will be sent, and therefore ideally a total of approximately  $2 * \text{cwnd}$  segments will be outstanding when a retransmission is finally triggered. `DupThresh` is then set to  $\text{LT\_F} = 1/2$  of  $2 * \text{cwnd}$  (or about 1 RTT's worth of data) (see step (I.4)). The factor is different for Careful Limited Transmit, because the sender only transmits one new segment for every two segments that are SACKed and therefore will ideally have a total of maximum of  $1.5 * \text{cwnd}$  segments outstanding when the retransmission is triggered. Hence, the required threshold is  $\text{LT\_F} = 2/3$  of  $1.5 * \text{cwnd}$  to delay the retransmission by roughly 1 RTT.

For each duplicate ACK received in the 'disorder' state, which is not an acceptable ACK, i.e., it carries new SACK information, but does not advance the cumulative ACK point, Extended Limited Transmit is executed. First, the SACK scoreboard is updated and based on the



current value of DupThresh, the amount of outstanding data (step (E.1)). Furthermore, the state variable 'burst' that indicates the number of segments that can be sent at most for of each received ACK is initialized to the size of the initial window [RFC6928] (step E.2)). If more than IW bytes were SACKed by a single ACK, the additional amount of data becomes available again by the next received duplicate ACK and the re-execution of SetPipe() (step (E.1)).

Next, if new data is available for transmission and both the congestion window and the receiver window allow to send SMSS bytes of previously unsent data, a segment of SMSS bytes is sent (step (E.3)). Subsequently, the corresponding state variables 'pipe', 'burst' and - optionally - 'skipped' are updated (steps (E.4) and (E.5)). If, due to the current size of the congestion and receiver windows (step (E.2)), due to the current value of 'burst' (step (E.5)), no further segment may be sent, the processing of the ACK is terminated. Provided that the amount of data that is currently considered to be in the network is greater than the previously stored one, this new value is stored for later use (step (E.7)). Finally, to take into account the new data sent, DupThresh is updated (steps (E.6) and (E.7)).

The arrival of an acceptable ACK in the 'disorder' state that advances the cumulative ACK point during Extended Limited Transmit signals that a series of duplicate ACKs was caused by reordering and not congestion. Therefore, the receipt of an acceptable ACK that does not carry any SACK information terminates Extended Limited Transmit (step (T.1)). The slow start threshold is set to the maximum of its current value and the current value of cwnd (step (T.3)). Cwnd itself is set to the current value of FlightSize plus one segment (step (T.4)). As a result, the congestion window is not significantly larger than the current amount of outstanding data, so that a burst of data is effectively prevented. If new data is available for transmission and both the new values of cwnd and rwnd allow to send SMSS bytes of previously unsent data, a segment is send (step (T.5)).

On the other hand, if the received ACK acknowledges new data not only cumulatively but also selectively - the ACK carries new SACK information - Extended Limited Transmit is not terminated but re-entered (step (T.1)). If the Cumulative Acknowledgment field of the received ACK covers more than 'recover', one cwnd worth of data has been transmitted during Extended Limited Transmit without any packet loss. Therefore, FlightSizePrev, the amount of outstanding data saved at the beginning of Extended Limited Transmit (step (I.1)), is considered outdated (step (T.2)). This step ensures that in the event of packet loss, the reduction of the cwnd is based on an up-to-

date value, which reflects the number of bytes outstanding in the network (see Section 7). Finally, regardless of whether or not 'recover' is covered, Extended Limited Transmit is re-entered.

The second case that leads to a termination of Extended Limited Transmit is the receipt of an ACK that signals via the algorithm in [RFC6675] that the oldest outstanding segment is considered lost. If either DupThresh or more duplicate ACKs are received, or the oldest outstanding segment is deemed lost via the function IsLost() of [RFC6675], Extended Limited Transmit is terminated and SACK-based loss recovery is entered [RFC6675]. Once the algorithm in [RFC6675] takes over from Extended Limited Transmit, the DupThresh value MUST be held constant until loss recovery is terminated. The process of loss recovery itself is not changed by TCP-aNCR. The only exception is a slight change to either RFC 6675 [RFC6675] or RFC 6937 [RFC6937], depending on whether the PRR algorithm or the traditional Fast Recovery algorithm is used during loss recovery. This change ensures that the adjustment made by the congestion control - the cwnd reduction - is made with respect to the initial amount of outstanding data while Limited Transmit Extended is executed (step (Ret)). The use of FlightSize at this point would no longer be valid since the amount of outstanding data may double by executing Extended Limited Transmit.

## 7. Discussion of TCP-aNCR

The specification of TCP-aNCR represents an incremental update of RFC 4653 [RFC4653]. All changes made by TCP-aNCR can be divided into two categories. On one hand, they implement TCP-aNCR's ability to dynamically adapted TCP congestion control and loss recovery [RFC5681] to the currently perceived packet reordering on the network path. These include the use of a variable DupThresh and the use of a relative reordering extent. On the other hand, the changes that basically correct weaknesses of the original TCP-NCR algorithm and which are independent of TCP-aNCR adaptability. These include packet reordering during slow start, the prevention of bursts, and the persistent receipt of SACKs.

### 7.1. Variable Duplicate Acknowledgment Threshold

The central point of the TCP-aNCR algorithm is the usage of a DupThresh that is adaptable to the perceived packet reordering on the network path. Based on the actual amount of outstanding data, TCP-NCR's DupThresh represents roughly the largest amount of time a Fast Retransmit can safely be delayed before a costly retransmission timeout may be triggered. Therefore, to avoid an RTO, TCP-aNCR's reordering-aware DupThresh is an upper bound of the one calculated in TCP-NCR (steps (I.5) and (E.9)). This decouples the avoidance of

spurious Fast Retransmits from the avoidance of RTOs. It allows TCP-aNCR to react fast and efficiently to packet reordering. The DupThresh always corresponds to the minimum of the largest possible and largest detected reordering. With constant packet reordering in terms of the rate and delay, TCP-aNCR gives a DupThresh based on the relative reordering extent with an optimal delay for every bandwidth-delay-product. If TCP-aNCR should not adaptively adjust the DupThresh to the current perceived packet reordering on the network path (because for example an appropriate detection and quantification algorithm is not implemented), the dynamically adaptation of TCP-aNCR can be disabled, so that TCP-aNCR behaves like TCP-NCR [RFC4653].

## 7.2. Relative Reordering Extent

Whenever a new reordering event is detected and presented to TCP-aNCR in the form of a relative reordering extend 'ReorExtR', TCP-aNCR saves and uses the new 'ReorExtR' if it is larger than the old one (step (EXT)). The upper bound of 1 assures that no excessively large value is used. A 'ReorExtR' larger than one means that more than FlightSize bytes would have been received out-of-order before the reordered segment is received. The delay caused by the reordering is thus longer than the RTT of the TCP connection. Since the RTT is roughly the time a Fast Retransmit can safely be delayed before the retransmission has to be to avoid an RTO, a maximum 'ReorExtR' of one seems to be a suitable value.

The expiration of the retransmission timer is interpreted by TCP-aNCR as an indication of a change in path characteristics, hence, the saved 'ReorExtR' is assumed to be outdated and will be invalidated (step (RTO)). As a consequence, the relative reordering extent 'ReorExtR' increases monotonically between two successive retransmission timeouts and corresponds to the maximum measured reordering extent since the last RTO. Other approaches would be an exponentially-weighted moving average (EWMA) or a histogram of the last n reordering extents. The main drawback of an EWMA is however that on average half of the detected reordering events would be larger than the saved reordering extend. Thus, only half of the spurious retransmits could be avoided. Applying an histogram could largely avoid the disadvantages of an EWMA, however, it would result in a not acceptable increase in memory usage.

In combination with the invalidation after an RTO, the advantage of using maximum is the low complexity as well as its fast convergence to the actual maximum reordering on the network path. As a result, the negative impact that packet reordering has on TCP's congestion control and loss recovery can be avoided. A disadvantage of using a maximum is that if the delay caused by the reordering decreases over the lifetime of the TCP connection, a Fast Retransmit is

unnecessarily long delayed. Nevertheless, since the negative impact reordering has on TCP's congestion control and loss recovery is more substantial than the disadvantage of a longer delay, a decrease of the ReorExtR between RTOs is considered inappropriate.

### 7.3. Reordering during Slow Start

The arrival of an acceptable ACK during Extended Limited Transmit signals that previously received duplicate ACKs are the result of packet reordering and not congestion, so that Extended Limited Transmit is completed accordingly. Upon the termination of Extended Limited Transmit, and especially when using the Careful variant, TCP-NCR (as well as TCP-aNCR) may be in a situation where the entire cwnd is not being utilized. Therefore, to mitigate a potential burst of segments, in step (T.2) TCP-NCR sets the slow start threshold to the FlightSize that was saved at the beginning of Extended Limited Transmit [RFC4653]. This step should ensure that TCP-NCR slow starts back to the operating point in use before Extended Limited Transmit.

Unfortunately, the assignment in step (T.2) is only correct if the TCP sender already was in congestion avoidance at the time Extended Limited Transmit was entered. Otherwise, if the TCP sender was instead in slow start, the value of ssthresh is greater than the saved FlightSize so that slow start prematurely concludes. This behavior can leave much of the network resources idle, and a long time may be needed in order to use the full capacity. To mitigate this issue, TCP-aNCR sets the slow start threshold to the maximum of its current value and the current cwnd (step (T.3)). This continues slow start after a reordering event happening during slow start.

### 7.4. Preventing Bursts

In cases where a new single SACK covers more than one segment - this can happen either due to packet loss or packet reordering on the ACK path - TCP-NCR [RFC4653] sends an undesirable burst of data. TCP-aNCR solves this problem by limiting the burst size - the maximum of data that can be sent in response to a single SACK - to the Initial Window [RFC5681] while executing Extended Limited Transmit (steps (E.2), (E.4), and (E.6)). Since IW represents the amount of data that a TCP sender is able to send into the network safely without knowing its characteristics, it is a reasonable value for the burst size, too. If more than IW bytes were SACKed by a single ACK, the additional amount of data becomes available again by the next received duplicate ACK. Thus, the transmission of new segments is spread over the next received ACKs, so that micro bursts - a characteristic of packet reordering in the reverse path - are largely compensated.

Another situation that causes undesired bursts of segments with TCP-NCR is the receipt of an acceptable ACK during Careful Extended Limited Transmit. If multiple segments from a single window of data are delayed by packet reordering, typically the first acceptable ACK after entering the 'disorder' state acknowledges data not only cumulatively but also selectively. Hence, Extended Limited Transmit is not terminated but re-started. If the segments are delayed by the reordering for almost one RTT, then the amount of outstanding data in the network ('pipe') is approximately half the amount of data saved at the beginning of Extended Limited Transmit (FlightSizePrev). If the sequence numbers of the delayed segments are close to each other in the sequence number space, the acceptable ACK acknowledges only a small amount of data, so that FlightSize is still large. As a result, TCP-NCR sets the cwnd to FlightSizePrev in step (T.1). Since 'pipe' is only half of FlightSizePrev due to Careful Extended Limited Transmit, TCP-NCR sends a burst of almost half a cwnd worth of data in the subsequent step (T.3).

Note: Even in the case the sequence numbers of the delayed segments are not close to each other in the sequence number space and cwnd is set in step (T.1) to FlightSize + SMSS, a burst of data will emerge due to re-entering Extended Limited Transmit, because TCP-NCR sets 'skipped' to zero in step (I.2) and uses FlightSizePrev in step (E.2).

TCP-aNCR prevents such a burst by making a clear differentiation between terminating Extended Limited Transmit and a restarting Extended Limited Transmit (step T.1). Only the first case causes the congestion window to be set to the current FlightSize plus one segment. In the latter case, when re-entering Extended Limited Transmit, the congestion window is not adjusted and the original (T.1) of the TCP-NCR specification is omitted. The transmission of new data is then only performed after re-entering Extended Limited Transmit in step (E.2) of the TCP-aNCR specification, where the actual burst mitigation takes place.

#### 7.5. Persistent receiving of Selective Acknowledgments

In some inconvenient cases it could happen that a TCP sender persistently receives SACK information due to reordering on the network path, e.g., if the segments are often and/or lengthy delayed by the packet reordering. With TCP-NCR, the persistent reception of SACKs causes Extended Limited Transmit to be entered with the first received duplicate ACK but never to be terminated if no packet loss occurs - for every received ACK, TCP-NCR either follows steps (E.1) to (E.6) or steps (T.1) to (T.4). In particular, TCP-NCR executes a) for every acceptable ACK step (T.4) and b) at any time step (I.1)

again. Hence, the amount of outstanding data saved at the beginning of Extended Limited Transmit, `FlightSizePrev`, is never updated.

An emerging problem in this context is that during Extended Limited Transmit TCP-NCR determines the transmission of new segments in step (E.2) solely on the basis of `FlightSizePrev`, so that an interim increase of the `cwnd` is not considered (according to [RFC5681], the congestion window is increased for every received acceptable ACK that advances the cumulative ACK point, no matter if it carries SACK information or not). As a result, TCP-NCR can only very slowly determine the available capacity of the communication path.

TCP-aNCR addresses this problem by limiting the amount of data that is allowed to be sent into the network during Extended Limited Transmit not on the basis of `FlightSizePrev`, but on the size of the congestion window. The equation in step E.3 of the TCP-aNCR specification is therefore equal to the one used in [RFC6675] (except for the 'skipped' variable). If an acceptable ACK is received during the execution of Extended Limited Transmit, re-entering Extended Limited Transmit makes any increase in `cwnd` immediately available. Hence, even in the case when persistently receiving SACKs, the available capacity of the communication path can be determined quickly.

Another problem resulting from persistently receiving SACKs, and which is related to the increase in `cwnd` in response to received acceptable ACKs, is the reduction of `cwnd` due to a packet loss. When a packet is considered lost, the congestion control adjustment is done with respect to the amount of outstanding data at the beginning of Extended Limited Transmit, `FlightSizePrev` (step (Ret)). As in the previous case, an increase in `cwnd` is again not taken into account. A simple solution to the problem would be to perform the window reduction not on the basis of `FlightSizePrev` but analogous to step (E.2) based on the current size of `cwnd`.

A problem with this solution is that `cwnd` can potentially be increased, although the TCP connection is limited by the application and not by `cwnd`. Although [RFC2861] specifies that an increase of `cwnd` is only applicable if `cwnd` is fully utilized, this behavior is not specified by any standards track document. But even this conservative increase behavior is guaranteed to not be conservative enough. If, from a single window of data, both segments are delayed but also lost, `cwnd` would first be increased in response to each received acceptable ACKs, while subsequently reduced due to the lost segments, which would not result in a halving of the `cwnd` any more.

The solution proposed by TCP-aNCR reuses the state variable 'recover' from [RFC6582] and adapts the approach taken by NewReno TCP and SACK

TCP to detect, with help of the state variable, the end of one loss recovery phase properly, allowing to recover multiple losses from a single window of data efficiently. Therefore, by entering the 'disorder' state and the starting Extended Limited Transmit, TCP-aNCR saves the highest sequence number sent so far in 'recover'. If a received acceptable ACK covers more than 'recover', one cwnd's worth of data has been transmitted during Extended Limited Transmit without any packet loss. Hence, FlightSizePrev can be updated by 'pipe\_max', which reflects the maximum amount of data that is considered to have been in the network during the last RTT. This update takes an interim increase in cwnd into account, so that in case of packet loss, the reduction in cwnd can be based on the current value of FlightSizePrev.

## 8. Interoperability Issues

TCP-aNCR requires that both the TCP Selective Acknowledgment Option [RFC2018] as well as a SACK-based loss recovery scheme compatible to one given in [RFC6675] are used by the TCP sender. Hence, compatibility to both specifications is REQUIRED.

### 8.1. Early Retransmit

The specification of TCP-aNCR in this document and the Early Retransmit algorithm specified in [RFC5827] define orthogonal methods to modify DupThresh. Early Retransmit allows the TCP sender to reduce the number of duplicate ACKs required to trigger a Fast Retransmit below the standard DupThresh of three, if FlightSize is less than 4\*SMSS and no new segment can be sent. In contrast, TCP-aNCR allows, starting from the minimum of three duplicate ACKs, to increase the DupThresh beyond the standard of three duplicate ACKs to make TCP more robust to packet reordering, if the amount of outstanding data is sufficient to reach the increased DupThresh to trigger Fast Retransmit and Fast Recovery.

### 8.2. Congestion Window Validation

The increase of the congestion window during application-limited periods can lead to an invalidation of the congestion window, in that it no longer reflects current information about the state of the network, if the congestion window might never have been fully utilized during the last RTT. According to [RFC2861], the congestion window should, first, only be increased during slow-start or congestion avoidance if the cwnd has been fully utilized by the TCP sender and, second, gradually be reduced during each RTT in which the cwnd was not fully used.

A problem that arises in this context is that during Careful Extended Limited Transmit, `cwnd` is not fully utilized due to the variable 'skipped' (see step (E.3)), so that - strictly following [RFC2861] - the congestion window should not be increased upon the receipt of an acceptable ACK. A trivial solution of this problem is to include the variable 'skipped' in the calculation of [RFC2861] to determine whether the congestion window is fully utilized or not.

### 8.3. Reactive Response to Packet Reordering

As a proactive scheme with the aim to a priori prevent the negative impact that packet reordering has on TCP, TCP-aNCR can conceptually be combined with any reactive response to packet reordering, which attempts to mitigate the negative effects of reordering a posteriori. This is because the modifications of TCP-aNCR to the standard TCP congestion control and loss recovery [RFC6675] are implemented in the 'disorder' state and are performed by the TCP sender before it enters loss recovery, while reactive responses to packet reordering operate generally after entering loss recovery, by undoing the unnecessarily changes to the congestion control state.

If unnecessary changes to the congestion control state are undone after loss recovery, which is typically the case if a spurious Fast Retransmit is detected based on the DSACK option [RFC3708][RFC4015], since first ACK carrying a DSACK option usually arrives at a TCP sender only after loss recovery has already terminated, it might happen that the restoring of the original value of the congestion window is done at a time at which the TCP sender is already back in again in the 'disorder' state and executing Extended Limited Transmit. While this is basically compatible with the TCP-aNCR specification - the undo simply represents an increase of the congestion window - however, some care must be taken that the combination of the algorithms does not lead to unwanted behavior.

### 8.4. Buffer Auto-Tuning

Although all modifications of the TCP-aNCR algorithm are implemented in the TCP sender, the receiver also potentially has a part to play. If some segments from a single window of data are delayed by the packet reordering in the network, all segments that are received in out-of-order have to be queued in the receive buffer until the holes in sequence number space have been closed and the data can be delivered to the receiving application. In the worst case, which occurs if the TCP sender uses Aggressive Limited Transmit and the reordering delay is close to the RTT, TCP-aNCR increases the receiver's buffering requirement by up to an extra `cwnd`. Therefore, to maximize the benefits from TCP-aNCR, receivers should advertise a large window - ideally by using buffer auto-tuning algorithms - to



absorb the extra out-of-order data. In the case that the additional buffer requirements are not met, the use of the above algorithm takes into account the reduced advertised window - with a corresponding loss in robustness to packet reordering.

## 9. Related Work

Over the past few years, several solutions have been proposed to improve the performance of TCP in the face of packet reordering. These schemes generally fall into one of two categories (with some overlap): mechanisms that try to prevent spurious retransmits from happening (proactive schemes) and mechanisms that try to detect spurious retransmits and undo the needless congestion control state changes that have been taken (reactive schemes).

[I-D.blanton-tcp-reordering], [ZKFP03] and [LM05] attempt to prevent packet reordering from triggering spurious retransmits by using various algorithms to approximate the DupThresh required to disambiguate loss and reordering over a given network path at a given time. This basic principle is also used in TCP-aNCR. While [I-D.blanton-tcp-reordering] describes four basic approaches on how to increase the DupThresh and discusses pros and cons of these approaches, presents [ZKFP03] a relatively complex algorithm that saves the reordering extents in a histogram and calculates the DupThresh in a way that a certain percentage of samples is smaller than the DupThresh. [LM05] uses an EWMA for the same purpose. Both algorithms do not prevent all the spurious retransmissions by design.

In contrast to the above mentioned algorithms Linux [Linux] implements a proactive scheme by setting the DupThresh to the highest detected reordering and resets only upon an RTO. To avoid a costly retransmission timeout due to the increased DupThresh Linux implements first an extension of the Limited Transmit algorithm, second limits the DupThresh to an upper bound of 127 duplicate ACKs, and third prematurely enters loss recovery if too few segments are in-flight to reach the DupThresh and no additional segments can send. Especially the last change is commendable since, besides TCP-NCR, none of the described algorithms in this section mention a similar concern.

[BHLLO06] and [BSRV04] presents proactive schemes based on timers by which the DupThresh is ignored altogether. After the timer is expired TCP initialize the loss recovery. In [BSRV04] this timer has a length of one RTT and is started when the first duplicate ACK is received, whereas the approach taken in [BHLLO06] solely relies on timers to detect packet loss without taking into account any other congestion signals such as duplicate ACKs. It assigns each segment

send a timestamp and retransmits the segment if the corresponding timer fires.

TCP-NCR [RFC4653] tries to prevent spurious retransmits similar to [I-D.blanton-tcp-reordering] or [ZKFP03] as it delays a retransmission to disambiguate loss and reordering. However, TCP-NCR takes a simplified approach by simply delay a retransmission by an amount based on the current cwnd (in comparison to standard TCP), while the other schemes use relatively complex algorithms in an attempt to derive a more precise value for DupThresh that depends on the current patterns of packet reordering. Many of the features offered by TCP-NCR have been taken into account while designing TCP-aNCR.

Besides the proactive schemes, several other schemes have been developed to detect and mitigate needless retransmissions after the fact. The Eifel detection algorithm [RFC3522], the detection based on DSACKs [RFC3708], and F-RTO scheme [RFC5682] represent approaches to detect spurious retransmissions, while the Eifel response algorithm [RFC4015], [I-D.blanton-tcp-reordering], and Linux [Linux] present respectively implement algorithms to mitigate the changes these events made to the congestion control state. As discussed in Section 8.3 TCP-aNCR could be used in conjunction with these algorithms, with TCP-aNCR attempting to prevent spurious retransmits and some other scheme kicking in if the prevention failed.

## 10. IANA Considerations

This memo includes no request to IANA.

## 11. Security Considerations

By taking dedicated actions so that the perceived packet reordering in the network is either underestimating or overestimating by the use of an relative and absolute reordering, an attacker or misbehaving TCP receiver has in regards to TCP's congestion control two options to bias a TCP-aNCR sender. An underestimation of the present packet reordering in the network occurs, if for example, a misbehaving TCP receiver already acknowledges segments while they are actually still in-flight, causing holes premature are closed in the sequence number space of the SACK scoreboard. With regard to TCP-aNCR the result of an underestimated packet reordering is a too small DupThresh, resulting in a premature loss recovery execution. In context of TCP's congestion control the effects of such attacks are limited since the lower bound of TCP-aNCR's DupThresh is the default value of three duplicate ACKs [RFC5681], so that in worst case TCP-aNCR behaves equal to TCP SACK [RFC6675].

In contrast to an underestimation, an overestimation of the packet reordering in the network occurs, if for example, a misbehaving TCP receiver still further send SACKs for subsequent segments before it sends an acceptable ACK for the actually already received delayed segment, so that the hole in the sequence number space of the SACK scoreboard is later closed. In the context of TCP-aNCR the result of such an overestimation is a too large DupThresh, so that in the case of a packet loss TCP's loss recovery is executed later than necessary. Similar to the previous case, the effects of delayed entry into the loss recovery are limited because on the one hand TCP-NCR's DupThresh is used as an upper bound for TCP-aNCR's variable DupThresh so that the entrance to the loss recovery and the adaptation of the congestion window may be delayed at most one RTT. On the other hand, such a limited delay of the congestion control adjustment has even in the worst case only a limited impact on the performance of TCP connection and has generally been regarded as safe for use on the Internet [BBFS01].

## 12. Acknowledgments

The authors would like to thank Daniel Slot for his TCP-NCR implementation in Linux. We also thank the flowgrind [Flowgrind] authors and contributors for here performance measurement tool, which give us a powerful tool to analyze TCP's congestion control and loss recovery behavior in detail.

## 13. References

### 13.1. Normative References

- [I-D.zimmermann-tcpm-reordering-detection]  
Zimmermann, A., Schulte, L., Wolff, C., and A. Hannemann, "Detection and Quantification of Packet Reordering with TCP", draft-zimmermann-tcpm-reordering-detection-01 (work in progress), November 2013.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3042] Allman, M., Balakrishnan, H., and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, January 2001.

- [RFC4653] Bhandarkar, S., Reddy, A., Allman, M., and E. Blanton, "Improving the Robustness of TCP to Non-Congestion Events", RFC 4653, August 2006.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.
- [RFC6582] Henderson, T., Floyd, S., Gurtov, A., and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 6582, April 2012.
- [RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, August 2012.
- [RFC6928] Chu, J., Dukkupati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, April 2013.
- [RFC6937] Mathis, M., Dukkupati, N., and Y. Cheng, "Proportional Rate Reduction for TCP", RFC 6937, May 2013.

### 13.2. Informative References

- [BBFS01] Bansal, D., Balakrishnan, H., Floyd, S., and S. Shenker, "Dynamic Behavior of Slowly Responsive Congestion Control Algorithms", Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'01) pp. 263-274, September 2001.
- [BHLLO06] Bohacek, S., Hespanha, J., Lee, J., Lim, C., and K. Obraczka, "A New TCP for Persistent Packet Reordering", IEEE/ACM Transactions on Networking vol. 2, no. 14, pp. 369-382, April 2006.
- [BSRV04] Bhandarkar, S., Sadry, N., Reddy, A., and N. Vaidya, "TCP-DCR: A Novel Protocol for Tolerating Wireless Channel Errors", IEEE Transactions on Mobile Computing vol. 4, no. 5., pp. 517-529, September 2005.
- [Flowgrind] "Flowgrind Home Page", <<http://www.flowgrind.net>>.
- [I-D.blanton-tcp-reordering] Blanton, E., Dimond, R., and M. Allman, "Practices for TCP Senders in the Face of Segment Reordering", draft-blanton-tcp-reordering-00 (work in progress), February 2003.

- [LM05] Leung, C. and C. Ma, "Enhancing TCP Performance to Persistent Packet Reordering", KICS Journal of Communications and Networks vol. 7, no. 3, pp. 385-393, September 2005.
- [Linux] "The Linux Project", <<http://www.kernel.org>>.
- [RFC0896] Nagle, J., "Congestion control in IP/TCP internetworks", RFC 896, January 1984.
- [RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.
- [RFC2861] Handley, M., Padhye, J., and S. Floyd, "TCP Congestion Window Validation", RFC 2861, June 2000.
- [RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.
- [RFC3522] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC 3522, April 2003.
- [RFC3708] Blanton, E. and M. Allman, "Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions", RFC 3708, February 2004.
- [RFC4015] Ludwig, R. and A. Gurtov, "The Eifel Response Algorithm for TCP", RFC 4015, February 2005.
- [RFC5682] Sarolahti, P., Kojo, M., Yamamoto, K., and M. Hata, "Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP", RFC 5682, September 2009.
- [RFC5827] Allman, M., Avrachenkov, K., Ayesta, U., Blanton, J., and P. Hurtig, "Early Retransmit for TCP and Stream Control Transmission Protocol (SCTP)", RFC 5827, May 2010.
- [ZKFP03] Zhang, M., Karp, B., Floyd, S., and L. Peterson, "RR-TCP: A Reordering-Robust TCP with DSACK", Proceedings of the 11th IEEE International Conference on Network Protocols (ICNP'03) pp. 95-106, November 2003.

## Appendix A. Changes from previous versions of the draft

This appendix should be removed by the RFC Editor before publishing this document as an RFC.

## A.1. Changes from draft-zimmermann-tcpm-reordering-reaction-01

- o Specify interaction between TCP-aNCR and PRR.
- o Fix typo in DupThresh calculation (steps I.5 and E.9).

## A.2. Changes from draft-zimmermann-tcpm-reordering-reaction-00

- o Improved the wording throughout the document.
- o Replaced and updated some references.

## Authors' Addresses

Alexander Zimmermann  
NetApp, Inc.  
Sonnenallee 1  
Kirchheim 85551  
Germany

Phone: +49 89 900594712  
Email: alexander.zimmermann@netapp.com

Lennart Schulte  
Aalto University  
Otakaari 5 A  
Espoo 02150  
Finland

Phone: +358 50 4355233  
Email: lennart.schulte@aalto.fi

Carsten Wolff  
credativ GmbH  
Hohenzollernstrasse 133  
Moenchengladbach 41061  
Germany

Phone: +49 2161 4643 182  
Email: carsten.wolff@credativ.de

Arnd Hannemann  
credativ GmbH  
Hohenzollernstrasse 133  
Moenchengladbach 41061  
Germany

Phone: +49 2161 4643 134  
Email: arnd.hannemann@credativ.de

TCP Maintenance and Minor Extensions  
(TCPM) WG  
Internet-Draft  
Obsoletes: 675 721 879 1078 6013  
(if approved)  
Updates: 4614bis (if approved)  
Intended status: Informational  
Expires: January 30, 2015

A. Zimmermann  
NetApp, Inc.  
W. Eddy  
MTI Systems  
L. Eggert  
NetApp, Inc.  
July 29, 2014

Moving Undeployed TCP Extensions to Historic and Informational Status --  
An addition to RFC 6247  
draft-zimmermann-tcpm-undeployed-01

## Abstract

This document reclassifies several TCP extensions that have either been superseded or never seen widespread use to Historic status. The affected RFCs are RFC 675, RFC 721, RFC 879, RFC 1078, and RFC 6013. Additionally, it reclassifies RFC 813, RFC 814, RFC 816, RFC 817, RFC 872, RFC 896, and RFC 964 to Informational status. Most of those RFCs are today part of RFC 1122.

## Status of this Memo

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

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

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

This Internet-Draft will expire on January 30, 2015.

## Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of



publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## 1. Introduction

TCP has a long history. Over time, many RFCs accumulated that described aspects of the TCP protocol, implementation, and extensions. Some of these have become outdated or simply have never seen widespread deployment. Section 6 and 7.1 of the TCP Roadmap document [I-D.ietf-tcpm-tcp-rfc4614bis] already classifies a number of TCP extensions as "historic" and describes the reasons for doing so, but it does not instruct the RFC Editor and IANA to change the status of these RFCs in the RFC database and the relevant IANA registries. The sole purpose of this document is to do just that. Please refer to Section 6 and 7.1 of [I-D.ietf-tcpm-tcp-rfc4614bis] for justification.

## 2. RFC Editor Considerations

The RFC Editor is requested to change the status of the following RFCs to Historic [RFC2026]:

- o [RFC0675] on "Specification of Internet Transmission Control Program"
- o [RFC0721] on "Out-of-Band Control Signals in a Host-to-Host Protocol"
- o [RFC0879] on "TCP Maximum Segment Size and Related Topics"
- o [RFC1078] on "TCP port service Multiplexer (TCPMUX)"
- o [RFC6013] on "TCP Cookie Transactions"

The RFC Editor is requested to change the status of the following RFCs to Informational [RFC2026]:

- o [RFC0813] on "Window and Acknowledgement Strategy in TCP"
- o [RFC0814] on "Name, addresses, ports, and routes"

- o [RFC0816] on "Fault Isolation and Recovery"
- o [RFC0817] on "Modularity and efficiency in protocol implementation"
- o [RFC0872] on "TCP-on-a-LAN"
- o [RFC0896] on "Congestion Control in IP/TCP Internetworks"
- o [RFC0964] on "Some problems with the specification of the Military Standard Transmission Control Protocol"

### 3. Security Considerations

This document introduces no new security considerations. Each RFC listed in this document attempts to address the security considerations of the specification it contains.

### 4. References

#### 4.1. Normative References

- [RFC0675] Cerf, V., Dalal, Y., and C. Sunshine, "Specification of Internet Transmission Control Program", RFC 675, December 1974.
- [RFC0721] Garlick, L., "Out-of-Band Control Signals in a Host-to-Host Protocol", RFC 721, September 1976.
- [RFC0813] Clark, D., "Window and Acknowledgement Strategy in TCP", RFC 813, July 1982.
- [RFC0814] Clark, D., "Name, addresses, ports, and routes", RFC 814, July 1982.
- [RFC0816] Clark, D., "Fault isolation and recovery", RFC 816, July 1982.
- [RFC0817] Clark, D., "Modularity and efficiency in protocol implementation", RFC 817, July 1982.
- [RFC0872] Padlipsky, M., "TCP-on-a-LAN", RFC 872, September 1982.
- [RFC0879] Postel, J., "TCP maximum segment size and related topics", RFC 879, November 1983.

- [RFC0896] Nagle, J., "Congestion control in IP/TCP internetworks", RFC 896, January 1984.
- [RFC0964] Sidhu, D. and T. Blumer, "Some problems with the specification of the Military Standard Transmission Control Protocol", RFC 964, November 1985.
- [RFC1078] Lottor, M., "TCP port service Multiplexer (TCPMUX)", RFC 1078, November 1988.
- [RFC6013] Simpson, W., "TCP Cookie Transactions (TCPCT)", RFC 6013, January 2011.

#### 4.2. Informative References

- [I-D.ietf-tcpm-tcp-rfc4614bis]  
Duke, M., Braden, R., Eddy, W., Blanton, E., and A. Zimmermann, "A Roadmap for Transmission Control Protocol (TCP) Specification Documents", draft-ietf-tcpm-tcp-rfc4614bis-05 (work in progress), April 2014.
- [RFC2026] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

#### Authors' Addresses

Alexander Zimmermann  
NetApp, Inc.  
Sonnenallee 1  
Kirchheim 85551  
Germany

Phone: +49 89 900594712  
Email: alexander.zimmermann@netapp.com

Wesley M. Eddy  
MTI Systems  
3000 Aerospace Parkway  
Cleveland, OH 44135

Phone: 216-433-6682  
Email: wes@mti-systems.com

Lars Eggert  
NetApp, Inc.  
Sonnenallee 1  
Kirchheim 85551  
Germany

Phone: +49 89 900594306  
Email: lars@netapp.com

