

TCP Maintenance Working Group  
Internet-Draft  
Intended status: Experimental  
Expires: August 29, 2013

N. Dukkhipati  
N. Cardwell  
Y. Cheng  
M. Mathis  
Google, Inc  
February 25, 2013

Tail Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses  
draft-dukkhipati-tcpm-tcp-loss-probe-01.txt

## Abstract

Retransmission timeouts are detrimental to application latency, especially for short transfers such as Web transactions where timeouts can often take longer than all of the rest of a transaction. The primary cause of retransmission timeouts are lost segments at the tail of transactions. This document describes an experimental algorithm for TCP to quickly recover lost segments at the end of transactions or when an entire window of data or acknowledgments are lost. Tail Loss Probe (TLP) is a sender-only algorithm that allows the transport to recover tail losses through fast recovery as opposed to lengthy retransmission timeouts. If a connection is not receiving any acknowledgments for a certain period of time, TLP transmits the last unacknowledged segment (loss probe). In the event of a tail loss in the original transmissions, the acknowledgment from the loss probe triggers SACK/FACK based fast recovery. TLP effectively avoids long timeouts and thereby improves TCP performance.

## 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 August 29, 2013.

## Copyright Notice

Copyright (c) 2013 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. Terminology . . . . .	5
2. Loss probe algorithm . . . . .	5
2.1. Pseudocode . . . . .	6
2.2. FACK threshold based recovery . . . . .	8
3. Detecting recovered losses . . . . .	9
3.1. TLP Loss Detection: The Basic Idea . . . . .	9
3.2. TLP Loss Detection: Algorithm Details . . . . .	9
4. Discussion . . . . .	11
4.1. Unifying loss recoveries . . . . .	12
4.2. Recovery of any N-degree tail loss . . . . .	12
5. Experiments with TLP . . . . .	14
6. Related work . . . . .	16
7. Security Considerations . . . . .	17
8. IANA Considerations . . . . .	17
9. References . . . . .	18
Authors' Addresses . . . . .	19

## 1. Introduction

Retransmission timeouts are detrimental to application latency, especially for short transfers such as Web transactions where timeouts can often take longer than all of the rest of a transaction. This document describes an experimental algorithm, Tail Loss Probe (TLP), to invoke fast recovery for losses that would otherwise be only recoverable through timeouts.

The Transmission Control Protocol (TCP) has two methods for recovering lost segments. First, the fast retransmit algorithm relies on incoming duplicate acknowledgments (ACKs), which indicate that the receiver is missing some data. After a required number of duplicate ACKs have arrived at the sender, it retransmits the first unacknowledged segment and continues with a loss recovery algorithm such as the SACK-based loss recovery [RFC6675]. If the fast retransmit algorithm fails for any reason, TCP uses a retransmission timeout as the last resort mechanism to recover lost segments. If an ACK for a given segment is not received in a certain amount of time called retransmission timeout (RTO), the segment is resent [RFC6298].

Timeouts can occur in a number of situations, such as the following:

- (1) Drop tail at the end of transactions. Example: consider a transfer of five segments sent on a connection that has a congestion window of ten. Any degree of loss in the tail, such as segments four and five, will only be recovered via a timeout.
- (2) Mid-transaction loss of an entire window of data or ACKs. Unlike (1) there is more data waiting to be sent. Example: consider a transfer of four segments to be sent on a connection that has a congestion window of two. If the sender transmits two segments and both are lost then the loss will only be recovered via a timeout.
- (3) Insufficient number of duplicate ACKs to trigger fast recovery at sender. The early retransmit mechanism [RFC5827] addresses this problem in certain special circumstances, by reducing the number of duplicate ACKs required to trigger a fast retransmission.
- (4) An unexpectedly long round-trip time (RTT), such that the ACKs arrive after the RTO timer expires. The F-RTO algorithm [RFC5682] is designed to detect such spurious retransmission timeouts and at least partially undo the consequences of such events.

Measurements on Google Web servers show that approximately 70% of retransmissions for Web transfers are sent after the RTO timer expires, while only 30% are handled by fast recovery. Even on servers exclusively serving YouTube videos, RTO based retransmissions

account for about 46% of the retransmissions. If the losses are detectable from the ACK stream (through duplicate ACKs or SACK blocks) then early retransmit, fast recovery and proportional rate reduction are effective in avoiding timeouts [IMC11PRR]. Timeout retransmissions that occur in recovery and disorder state (a state indicating that a connection has received some duplicate ACKs), account for just 4% of the timeout episodes. On the other hand 96% of the timeout episodes occur without any preceding duplicate ACKs or other indication of losses at the sender [IMC11PRR]. Early retransmit and fast recovery have no hope of repairing losses without these indications. Efficiently addressing situations that would cause timeouts without any prior indication of losses is a significant opportunity for additional improvements to loss recovery.

To get a sense of just how long the RTOs are in relation to connection RTTs, following is the distribution of RTO/RTT values on Google Web servers. [percentile, RTO/RTT]: [50th percentile, 4.3]; [75th percentile, 11.3]; [90th percentile, 28.9]; [95th percentile, 53.9]; [99th percentile, 214]. Large RTOs, typically caused by variance in measured RTTs, can be a result of intermediate queuing, and service variability in mobile channels. Such large RTOs make a huge contribution to the long tail on the latency statistics of short flows. Note that simply reducing the length of RTO does not address the latency problem for two reasons: first, it increases the chances of spurious retransmissions. Second and more importantly, an RTO reduces TCP's congestion window to one and forces a slow start. Recovery of losses without relying primarily on the RTO mechanism is beneficial for short TCP transfers.

The question we address in this document is: Can a TCP sender recover tail losses of transactions through fast recovery and thereby avoid lengthy retransmission timeouts? We specify an algorithm, Tail Loss Probe (TLP), which sends probe segments to trigger duplicate ACKs with the intent of invoking fast recovery more quickly than an RTO at the end of a transaction. TLP is applicable only for connections in Open state, wherein a sender is receiving in-sequence ACKs and has not detected any lost segments. TLP can be implemented by modifying only the TCP sender, and does not require any TCP options or changes to the receiver for its operation. For convenience, this document mostly refers to TCP, but the algorithms and other discussion are valid for Stream Control Transmission Protocol (SCTP) as well.

This document is organized as follows. Section 2 describes the basic Loss Probe algorithm. Section 3 outlines an algorithm to detect the cases when TLP plugs a hole in the sender. The algorithm makes the sender aware that a loss had occurred so it performs the appropriate congestion window reduction. Section 4 discusses the interaction of TLP with early retransmit in being able to recover any degree of tail

losses. Section 5 discusses the experimental results with TLP on Google Web servers. Section 6 discusses related work, and Section 7 discusses the security considerations.

### 1.1. 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].

## 2. Loss probe algorithm

The Loss probe algorithm is designed for a sender to quickly detect tail losses without waiting for an RTO. We will henceforth use tail loss to generally refer to either drops at the tail end of transactions or a loss of an entire window of data/ACKs. TLP works for senders with SACK enabled and in Open state, i.e. the sender has so far received in-sequence ACKs with no SACK blocks. The risk of a sender incurring a timeout is high when the sender has not received any ACKs for a certain portion of time but is unable to transmit any further data either because it is application limited (out of new data to send), receiver window (rwnd) limited, or congestion window (cwnd) limited. For these circumstances, the basic idea of TLP is to transmit probe segments for the specific purpose of eliciting additional ACKs from the receiver. The initial idea was to send some form of zero window probe (ZWP) with one byte of new or old data. The ACK from the ZWP would provide an additional opportunity for a SACK block to detect loss without an RTO. Additional losses can be detected subsequently and repaired as SACK based fast recovery proceeds. However, in practice sending a single byte of data turned out to be problematic to implement and more fragile than necessary. Instead we use a full segment to probe but have to add complexity to compensate for the probe itself masking losses.

Define probe timeout (PTO) to be a timer event indicating that an ACK is overdue on a connection. The PTO value is set to  $\max(2 * \text{SRTT}, 10\text{ms})$ , where SRTT is the smoothed round-trip time [RFC6298], and is adjusted to account for delayed ACK timer when there is only one outstanding segment.

The basic version of the TLP algorithm transmits one probe segment after a probe timeout if the connection has outstanding unacknowledged data but is otherwise idle, i.e. not receiving any ACKs or is cwnd/rwnd/application limited. The transmitted segment, aka loss probe, can be either a new segment if available and the receive window permits, or a retransmission of the most recently sent segment, i.e., the segment with the highest sequence number. When

there is tail loss, the ACK from the probe triggers fast recovery. In the absence of loss, there is no change in the congestion control or loss recovery state of the connection, apart from any state related to TLP itself.

TLP MUST NOT be used for non-SACK connections. SACK feedback allows senders to use the algorithm described in section 3 to infer whether any segments were lost.

## 2.1. Pseudocode

We define the terminology used in specifying the TLP algorithm:

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

**RTO:** The transport's retransmission timeout (RTO) is based on measured round-trip times (RTT) between the sender and receiver, as specified in [RFC6298] for TCP.

**PTO:** Probe timeout is a timer event indicating that an ACK is overdue. Its value is constrained to be smaller than or equal to an RTO.

**SRTT:** smoothed round-trip time computed like in [RFC6298].

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

**Consecutive PTOs:** back-to-back PTOs all scheduled for the same tail packets in a flight. The (N+1)st PTO is scheduled after transmitting the probe segment for Nth PTO.

The TLP algorithm works as follows:

(1) Schedule PTO after transmission of new data in Open state:

Check for conditions to schedule PTO outlined in step 2 below.

FlightSize > 1: schedule PTO in  $\max(2 \cdot \text{SRTT}, 10\text{ms})$ .

FlightSize == 1: schedule PTO in  $\max(2 \cdot \text{SRTT}, 1.5 \cdot \text{SRTT} + \text{WCDelAckT})$ .

If RTO is earlier, schedule PTO in its place:  $\text{PTO} = \min(\text{RTO}, \text{PTO})$ .

WCDelAckT stands for worst case delayed ACK timer. When FlightSize is 1, PTO is inflated additionally by WCDelAckT time to compensate for a potential long delayed ACK timer at the receiver. The RECOMMENDED value for WCDelAckT is 200ms.

A PTO value of  $2 \times \text{SRTT}$  allows a sender to wait long enough to know that an ACK is overdue. Under normal circumstances, i.e. no losses, an ACK typically arrives in one RTT. But choosing PTO to be exactly an RTT is likely to generate spurious probes given that even end-system timings can easily push an ACK to be above an RTT. We chose PTO to be the next integral value of RTT. If RTO is smaller than the computed value for PTO, then a probe is scheduled to be sent at the RTO time. The RTO timer is rearmed at the time of sending the probe, as is shown in Step 3 below. This ensures that a PTO is always sent prior to a connection experiencing an RTO.

(2) Conditions for scheduling PTO:

- (a) Connection is in Open state.
- (b) Connection is either cwnd limited or application limited.
- (c) Number of consecutive PTOs  $\leq 2$ .
- (d) Connection is SACK enabled.

Implementations MAY use one or two consecutive PTOs.

(3) When PTO fires:

- (a) If a new previously unsent segment exists:
    - > Transmit new segment.
    - > FlightSize += SMSS. cwnd remains unchanged.
  - (b) If no new segment exists:
    - > Retransmit the last segment.
  - (c) Increment statistics counter for loss probes.
  - (d) If conditions in (2) are satisfied:
    - > Reschedule next PTO.
- Else:
- > Rearm RTO to fire at epoch 'now+RTO'.

The reason for retransmitting the last segment in Step (b) is so that the ACK will carry SACK blocks and trigger either SACK-based loss recovery [RFC6675] or FACK threshold based fast recovery [FACK]. On transmission of a TLP, a MIB counter is incremented to keep track of the total number of loss probes sent.

(4) During ACK processing:

Cancel any existing PTO.

If conditions in (2) allow:

- > Reschedule PTO relative to the ACK receipt time.

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

(1) Sender transmits segments 1-10: 1, 2, 3, ..., 8, 9, 10. There is no more new data to transmit. A PTO is scheduled to fire in 2 RTTs, after the transmission of the 10th segment.

(2) Receives acknowledgements (ACKs) for segments 1-5; segments 6-10 are lost and no ACKs are received. Note that the sender (re)schedules its PTO timer relative to the last received ACK, which is the ACK for segment 5 in this case. The sender sets the PTO interval using the calculation described in step (1) of the algorithm.

(3) When PTO fires, sender retransmits segment 10.

(4) After an RTT, SACK for packet 10 arrives. The ACK also carries SACK holes for segments 6, 7, 8 and 9. This triggers FACK threshold based recovery.

(5) Connection enters fast recovery and retransmits remaining lost segments.

## 2.2. FACK threshold based recovery

At the core of TLP is its reliance on FACK threshold based algorithm to invoke Fast Recovery. In this section we specify this algorithm.

Section 3.1 of the Forward Acknowledgement (FACK) Paper [FACK] describes an alternate algorithm for triggering fast retransmit, based on the extent of the SACK scoreboard. Its goal is to trigger fast retransmit as soon as the receiver's reassembly queue is larger than the dupack threshold, as indicated by the difference between the forward most SACK block edge and SND.UNA. This algorithm quickly and reliably triggers fast retransmit in the presence of burst losses -- often on the first SACK following such a loss. Such a threshold based algorithm also triggers fast retransmit immediately in the presence of any reordering with extent greater than the dupack threshold.

FACK threshold based recovery works by introducing a new TCP state variable at the sender called SND.FACK. SND.FACK reflects the forward-most data held by the receiver and is updated when a SACK block is received acknowledging data with a higher sequence number than the current value of SND.FACK. SND.FACK reflects the highest sequence number known to have been received plus one. Note that in non-recovery states, SND.FACK is the same as SND.UNA. The following snippet is the pseudocode for FACK threshold based recovery.

```
If (SND.FACK - SND.UNA) > dupack threshold:  
    -> Invoke Fast Retransmit and Fast Recovery.
```



### 3. Detecting recovered losses

If the only loss was the last segment, there is the risk that the loss probe itself might repair the loss, effectively masking it from congestion control. To avoid interfering with mandatory congestion control [RFC5681] it is imperative that TLP include a mechanism to detect when the probe might have masked a loss and to properly reduce the congestion window (cwnd). An algorithm to examine subsequent ACKs to determine whether the original segment was lost is described here.

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

#### 3.1. TLP Loss Detection: The Basic Idea

Consider a TLP retransmission "episode" where a sender retransmits N consecutive TLP packets, all for the same tail packet in a flight. Let us say that an episode ends when the sender receives an ACK above the SND.NXT at the time of the episode. We want to make sure that before the episode ends the sender receives N "TLP dupacks", indicating that all N TLP probe segments were unnecessary, so there was no loss/hole that needed plugging. If the sender gets less than N "TLP dupacks" before the end of the episode, then probably the first TLP packet to arrive at the receiver plugged a hole, and only the remaining TLP packets that arrived at the receiver generated dupacks.

Note that delayed ACKs complicate the picture, since a delayed ACK will imply that the sender receives one fewer ACK than would normally be expected. To mitigate this complication, before sending a TLP loss probe retransmission, the sender should attempt to wait long enough that the receiver has sent any delayed ACKs that it is withholding. The sender algorithm, described in section 2.1 features such a delay.

If there is ACK loss or a delayed ACK, then this algorithm is conservative, because the sender will reduce cwnd when in fact there was no packet loss. In practice this is acceptable, and potentially even desirable: if there is reverse path congestion then reducing cwnd is prudent.

#### 3.2. TLP Loss Detection: Algorithm Details

##### (1) State

**TLPRTxOut:** the number of unacknowledged TLP retransmissions in current TLP episode. The connection maintains this integer counter that tracks the number of TLP retransmissions in the current episode for which we have not yet received a "TLP dupack". The sender initializes the TLPRTxOut field to 0.

**TLPHighRxt:** the value of SND.NXT at the time of TLP retransmission. The TLP sender uses TLPHighRxt to record SND.NXT at the time it starts doing TLP transmissions during a given TLP episode.

## (2) Initialization

When a connection enters the ESTABLISHED state, or suffers a retransmission timeout, or enters fast recovery, it executes the following:

```
TLPRTxOut = 0;
TLPHighRxt = 0;
```

## (3) Upon sending a TLP retransmission:

```
if (TLPRTxOut == 0)
    TLPHighRxt = SND.NXT;
TLPRTxOut++;
```

## (4) Upon receiving an ACK:

### (a) Tracking ACKs

We define a "TLP dupack" as a dupack that has all the regular properties of a dupack that can trigger fast retransmit, plus the ACK acknowledges TLPHighRxt, and the ACK carries no new SACK information (as noted earlier, TLP requires that the receiver supports SACK). This is the kind of ACK we expect to see for a TLP transmission if there were no losses. More precisely, the TLP sender considers a TLP probe segment as acknowledged if all of the following conditions are met:

- (a) TLPRTxOut > 0
- (b) SEG.ACK == TLPHighRxt
- (c) the segment contains no SACK blocks for sequence ranges above TLPHighRxt
- (d) the ACK does not advance SND.UNA
- (e) the segment contains no data
- (f) the segment is not a window update

If all of those conditions are met, then the sender executes the following:

```
TLPRtxOut--;
```

(b) Marking the end of a TLP episode and detecting losses

If an incoming ACK is after `TLPHighRxt`, then the sender deems the TLP episode over. At that time, the TLP sender executes the following:

```
isLoss = (TLPRtxOut > 0) &&  
         (segment does not carry a DSACK for TLP retransmission);  
TLPRtxOut = 0  
if (isLoss)  
    EnterRecovery();
```

In other words, if the sender detects an ACK for data beyond the TLP loss probe retransmission then (in the absence of reordering on the return path of ACKs) it should have received any ACKs that will indicate whether the original or any loss probe retransmissions were lost. An exception is the case when the segment carries a Duplicate SACK (DSACK) for the TLP retransmission. If the `TLPRtxOut` count is still non-zero and thus indicates that some TLP probe segments remain unacknowledged, then the sender should presume that at least one segment was lost, so it should enter fast recovery using the proportional rate reduction algorithm [IMC11PRR].

(5) Senders must only send a TLP loss probe retransmission if all the conditions from section 2.1 are met and the following condition also holds:

```
(TLPRtxOut == 0) || (SND.NXT == TLPHighRxt)
```

This ensures that there is at most one sequence range with outstanding TLP retransmissions. The sender maintains this invariant so that there is at most one TLP retransmission "episode" happening at a time, so that the sender can use the algorithm described above in this section to determine when the episode is over, and thus when it can infer whether any data segments were lost.

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

#### 4. Discussion

In this section we discuss two properties related to TLP.

#### 4.1. Unifying loss recoveries

The existing loss recovery algorithms in TCP have a discontinuity: A single segment loss in the middle of a packet train can be recovered via fast recovery while a loss at the end of the train causes an RTO. Example: consider a train of segments 1-10, loss of segment five can be recovered quickly through fast recovery, while loss of segment ten can only be recovered through a timeout. In practice, the difference between losses that trigger RTO versus those invoking fast recovery has more to do with the position of the losses as opposed to the intensity or magnitude of congestion at the link.

TLP unifies the loss recovery mechanisms regardless of the position of a loss, so now with TLP a segment loss in the middle of a train as well as at the tail end can now trigger the same fast recovery mechanisms.

#### 4.2. Recovery of any N-degree tail loss

The TLP algorithm, when combined with a variant of the early retransmit mechanism described below, is capable of recovering any tail loss for any sized flow using fast recovery.

We propose the following enhancement to the early retransmit algorithm described in [RFC5827]: in addition to allowing an early retransmit in the scenarios described in [RFC5827], we propose to allow a delayed early retransmit [IMC11PRR] in the case where there are three outstanding segments that have not been cumulatively acknowledged and one segment that has been fully SACKed.

Consider the following scenario, which illustrates an example of how this enhancement allows quick loss recovery in a new scenario:

- (1) scoreboard reads: A \_ \_ \_
- (2) TLP retransmission probe of the last (fourth) segment
- (3) the arrival of a SACK for the last segment changes scoreboard to: A \_ \_ S
- (4) early retransmit and fast recovery of the second and third segments

With this enhancement to the early retransmit mechanism, then for any degree of N-segment tail loss we get a quick recovery mechanism instead of an RTO.

Consider the following taxonomy of tail loss scenarios, and the ultimate outcome in each case:

	number of losses	scoreboard after TLP retrans ACKed	mechanism	final outcome
(1)	AAAL	AAAA	TLP loss detection	all repaired
(2)	AALL	AALS	early retransmit	all repaired
(3)	ALLL	ALLS	early retransmit	all repaired
(4)	LLLL	LLLS	FAK fast recovery	all repaired
(5)	>=5 L	..LS	FAK fast recovery	all repaired

key:

A = ACKed segment

L = lost segment

S = SACKed segment

Let us consider each tail loss scenario in more detail:

(1) With one segment lost, the TLP loss probe itself will repair the loss. In this case, the sender's TLP loss detection algorithm will notice that a segment was lost and repaired, and reduce its congestion window in response to the loss.

(2) With two segments lost, the TLP loss probe itself is not enough to repair the loss. However, when the SACK for the loss probe arrives at the sender, then the early retransmit mechanism described in [RFC5827] will note that with two segments outstanding and the second one SACKed, the sender should retransmit the first segment. This retransmit will repair the single remaining lost segment.

(3) With three segments lost, the TLP loss probe itself is not enough to repair the loss. However, when the SACK for the loss probe arrives at the sender, then the enhanced early retransmit mechanism described in this section will note that with three segments outstanding and the third one SACKed, the sender should retransmit the first segment and enter fast recovery. The early retransmit and fast recovery phase will, together, repair the the remaining two lost segments.

(4) With four segments lost, the TLP loss probe itself is not enough to repair the loss. However, when the SACK for the loss probe arrives at the sender, then the FACK fast retransmit mechanism [FACK] will note that with four segments outstanding and the fourth one SACKed, the sender should retransmit the first segment and enter fast recovery. The fast retransmit and fast recovery phase will, together, repair the the remaining two lost segments.

(5) With five or more segments lost, events precede much as in case (4). The TLP loss probe itself is not enough to repair the loss.

However, when the SACK for the loss probe arrives at the sender, then the FACK fast retransmit mechanism [FACK] will note that with five or more segments outstanding and the segment highest in sequence space SACKed, the sender should retransmit the first segment and enter fast recovery. The fast retransmit and fast recovery phase will, together, repair the remaining lost segments.

In summary, the TLP mechanism, in conjunction with the proposed enhancement to the early retransmit mechanism, is able to recover from a tail loss of any number of segments without resort to a costly RTO.

## 5. Experiments with TLP

In this section we describe experiments and measurements with TLP performed on Google Web servers using Linux 2.6. The experiments were performed over several weeks and measurements were taken across a wide range of Google applications. The main goal of the experiments is to instrument and measure TLP's performance relative to the baseline. The experiment and baseline were using the same kernels with an on/off switch to enable TLP.

Our experiments include both the basic TLP algorithm of Section 2 and its loss detection component in Section 3. All other algorithms such as early retransmit and FACK threshold based recovery are present in the both the experiment and baseline. There are three primary metrics we are interested in: impact on TCP latency (average and tail or 99th percentile latency), retransmission statistics, and the overhead of probe segments relative to the total number of transmitted segments. TCP latency is the time elapsed between the server transmitting the first byte of the response to it receiving an ACK for the last byte.

The table below shows the percentiles and average latency improvement of key Web applications, including even those responses without losses, measured over a period of one week. The key takeaway is: the average response time improved up to 7% and the 99th percentile improved by 10%. Nearly all of the improvement for TLP is in the tail latency (post-90th percentile). The varied improvements across services are due to different response-size distributions and traffic patterns. For example, TLP helps the most for Images, as these are served by multiple concurrently active TCP connections which increase the chances of tail segment losses.

Application	Average	99%
Google Web Search	-3%	-5%
Google Maps	-5%	-10%
Google Images	-7%	-10%

TLP also improved performance in mobile networks -- by 7.2% for Web search and Instant and 7.6% for Images transferred over Verizon network. To see why and where the latency improvements are coming from, we measured the retransmission statistics. We broke down the retransmission stats based on nature of retransmission -- timeout retransmission or fast recovery. TLP reduced the number of timeouts by 15% compared to the baseline, i.e.  $(\text{timeouts\_tlp} - \text{timeouts\_baseline}) / \text{timeouts\_baseline} = 15\%$ . Instead, these losses were either recovered via fast recovery or by the loss probe retransmission itself. The largest reduction in timeouts is when the sender is in the Open state in which it receives only insequence ACKs and no duplicate ACKs, likely because of tail losses. Correspondingly, the retransmissions occurring in the slow start phase after RTO reduced by 46% relative to baseline. Note that it is not always possible for TLP to convert 100% of the timeouts into fast recovery episodes because a probe itself may be lost. Also notable in our experiments is a significant decrease in the number of spurious timeouts -- the experiment had 61% fewer congestion window undo events. The Linux TCP sender uses either DSACK or timestamps to determine if retransmissions are spurious and employs techniques for undoing congestion window reductions. We also note that the total number of retransmissions decreased 7% with TLP because of the decrease in spurious retransmissions, and because the TLP probe itself plugs a hole.

We also quantified the overhead of probe packets. The probes accounted for 0.48% of all outgoing segments, i.e.  $(\text{number of probe segments} / \text{number of outgoing segments}) * 100 = 0.48\%$ . This is a reasonable overhead when contrasted with the overall retransmission rate of 3.2%. 10% of the probes sent are new segments and the rest are retransmissions, which is unsurprising given that short Web responses often don't have new data to send. We also found that in about 33% of the cases, the probes themselves plugged the only hole at receiver and the loss detection algorithm reduced the congestion window. 37% of the probes were not necessary and resulted in a duplicate acknowledgment.

Besides the macro level latency and retransmission statistics, we report some measurements from TCP's internal state variables at the

point when a probe segment is transmitted. The following distribution shows the FlightSize and congestion window values when a PTO is scheduled. We note that cwnd is not the limiting factor and that nearly all of the probe segments are sent within the congestion window.

percentile	10%	25%	50%	75%	90%	99%
FlightSize	1	1	2	3	10	20
cwnd	5	10	10	10	17	44

We have also experimented with a few variations of TLP: multiple probe segments versus single probe for the same tail loss episode, and several values for WCDelAckT. Our experiments show that sending just one probe suffices to get most (~90%) of latency benefits. The experiment results reported in this section and our current implementation limits number of probes to one, although the draft itself allows up to two consecutive probes. We chose the worst case delayed ack timer to be 200ms. When FlightSize equals 1 it is important to account for the delayed ACK timer in the PTO value, in order to bring down the number of unnecessary probe segments. With delays of 0ms and 50ms, the probe overhead jumped from 0.48% to 3.1% and 2.2% respectively. We have also experimented with transmitting 1-byte probe retransmissions as opposed to an entire MSS retransmission probe. While this scheme has the advantage of not requiring the loss detection algorithm outlined in Section 3, it turned out to be problematic to implement correctly in certain TCP stacks. Additionally, retransmitting 1-byte probe costs one more RTT to recover single packet tail losses, which is detrimental for short transfer latency.

## 6. Related work

TCP's long and conservative RTO recovery has long been identified as the major performance bottleneck for latency-demanding applications. A well-studied example is online gaming that requires reliability and low latency but small bandwidth. [GRIWODZ06] shows that repeated long RTO is the dominating performance bottleneck for game responsiveness. The authors in [PETLUND08] propose to use linear RTO to improve the performance, which has been incorporated in the Linux kernel as a non-default socket option for such thin streams. [MONDAL08] further argues exponential RTO backoff should be removed because it is not necessary for the stability of Internet. In contrast, TLP does not change the RTO timer calculation or the exponential back off. TLP's approach is to keep the behavior after RTO conservative for stability but allows a few timely probes before concluding the network is badly congested and cwnd should fall to 1.



As noted earlier in the Introduction the F-RTO [RFC5682] algorithm reduces the number of spurious timeout retransmissions and the Early Retransmit [RFC5827] mechanism reduces timeouts when a connection has received a certain number of duplicate ACKs. Both are complementary to TLP and can work alongside. Rescue retransmission introduced in [RFC6675] deals with loss events such as AL\*SL\* (using the same notation as section 4). TLP covers wider range of events such as AL\*. We experimented with rescue retransmission on Google Web servers, but did not observe much performance improvement. When the last segment is lost, it is more likely that a number of contiguous segments preceding the segment are also lost, i.e. AL\* is common. Timeouts that occur in the fast recovery are rare.

[HURTIG13] proposes to offset the elapsed time of the pending packet when re-arming the RTO timer. It is possible to apply the same idea for the TLP timer as well. We have not yet tested such a change to TLP.

Tail Loss Probe is one of several algorithms designed to maximize the robustness of TCPs self clock in the presence of losses. It follows the same principles as Proportional Rate Reduction [IMC11PRR] and TCP Laminar [Laminar].

On a final note we note that Tail loss probe does not eliminate 100% of all RTOs. RTOs still remain the dominant mode of loss recovery for short transfers. More work in future should be done along the following lines: transmitting multiple loss probes prior to finally resorting to RTOs, maintaining ACK clocking for short transfers in the absence of new data by clocking out old data in response to incoming ACKs, taking cues from applications to indicate end of transactions and use it for smarter tail loss recovery.

## 7. Security Considerations

The security considerations outlined in [RFC5681] apply to this document. At this time we did not find any additional security problems with Tail loss probe.

## 8. IANA Considerations

This document makes no request of IANA.

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

## 9. References

- [RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, August 2012.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.
- [RFC5827] Allman, M., Ayesta, U., Wang, L., Blanton, J., and P. Hurtig, "Early Retransmit for TCP and Stream Control Transmission Protocol (SCTP)", RFC 5827, April 2010.
- [RFC5682] Sarolahti, P., Kojo, M., Yamamoto, K., and M. Hata, "Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP", RFC 5682, September 2009.
- [IMC11PRR] Mathis, M., Dukkkipati, N., Cheng, Y., and M. Ghobadi, "Proportional Rate Reduction for TCP", Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference , 2011.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.
- [FACK] Mathis, M. and M. Jamshid, "Forward acknowledgement: refining TCP congestion control", ACM SIGCOMM Computer Communication Review, Volume 26, Issue 4, Oct. 1996. , 1996.
- [RFC2883] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, July 2000.
- [RFC2018] Mathis, M. and J. Mahdavi, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.
- [GRIWODZ06] Griwodz, C. and P. Halvorsen, "The fun of using TCP for an MMORPG", NOSSDAV , 2006.

[PETLUND08]

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

[MONDAL08]

Mondal, A. and A. Kuzmanovic, "Removing Exponential Backoff from TCP", ACM SIGCOMM Computer Communication Review , 2008.

[Laminar] Mathis, M., "Laminar TCP and the case for refactoring TCP congestion control", July 2012.

[HURTIG13]

Hurtig, P., Brunstrom, A., Petlund, A., and M. Welzl, "TCP and SCTP RTO Restart", draft-ietf-tcpm-rtorestart-00 (work in progress), February 2013.

#### Authors' Addresses

Nandita Dukkipati  
Google, Inc  
1600 Amphitheater Parkway  
Mountain View, California 93117  
USA

Email: nanditad@google.com

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

Email: ncardwell@google.com

Yuchung Cheng  
Google, Inc  
1600 Amphitheater Parkway  
Mountain View, California 93117  
USA

Email: ycheng@google.com

Matt Mathis  
Google, Inc  
1600 Amphitheater Parkway  
Mountain View, California 93117  
USA

Email: mattmathis@google.com



TCP Maintenance Working Group  
Internet-Draft  
Intended status: Experimental  
Expires: January 16, 2014

T. Flach  
USC  
N. Dukkipati  
Y. Cheng  
B. Raghavan  
Google  
July 15, 2013

TCP Instant Recovery: Incorporating Forward Error Correction in TCP  
draft-flach-tcpm-fec-00.txt

## Abstract

Ordinary TCP loss recovery takes at least one round-trip time and as such can increase application-perceived latency, especially for short flows such as Web transactions. TCP Instant Recovery (TCP-IR) is an experimental algorithm that allows a receiving end to recover lost packets without retransmissions, thus potentially saving at least one full round-trip time compared to standard TCP. TCP-IR achieves this by judiciously injecting encoded data segments within a TCP stream. This document describes the TCP-IR algorithm at the sending and receiving ends, along with the required protocol 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 January 16, 2014.

## Copyright Notice

Copyright (c) 2013 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
1.1. Terminology . . . . .	3
2. Protocol Overview . . . . .	3
3. Protocol Details . . . . .	5
3.1. TCP-IR Option . . . . .	6
3.2. Negotiation . . . . .	7
3.3. Encoding Types . . . . .	8
3.4. TCP-IR Sender . . . . .	9
3.5. TCP-IR Receiver . . . . .	9
3.6. Processing Acknowledgements . . . . .	11
4. Interaction with middleboxes . . . . .	11
5. Implementation and Performance . . . . .	12
6. Related Work . . . . .	13
7. Security Considerations . . . . .	13
8. IANA Considerations . . . . .	13
9. References . . . . .	13
Authors' Addresses . . . . .	14

## 1. Introduction

TCP Instant Recovery (TCP-IR) enables a receiver to recover lost data segments instantly without the need for retransmissions from a sender. Standard TCP retransmission-based loss recovery takes at least one RTT for loss detection and recovery.

The main motivation for TCP-IR is to reduce the tail latency of Web transactions. Most Web transfers are short and could finish within a few round-trip times (RTTs), but losses can add multiple RTTs to transfer times and increase the variance in Web page download times. The goal of TCP-IR is to reduce loss detection and recovery to zero RTT while still employing TCP's congestion control principles.

Recovery mechanisms, such as fast recovery and retransmission timeout, are fundamentally RTT dependent. Regardless of how fast network bandwidth grows, the number of RTTs that it takes to recover lost packets does not change. TCP-IR employs forward error correction (FEC) to scale the recovery time inversely with bandwidth and make it independent of RTT. It explicitly trades some network

bandwidth to reduce RTTs for short transfers. Most bandwidth in the Internet is used by large flows such as video, and thus short, latency-sensitive traffic can benefit using a small degree of FEC without hurting bulk flow throughput.

In this document, we specify the TCP-IR mechanism, which requires both sender and receiver changes, to achieve 1-RTT recovery for commonly observed loss scenarios. Instead of complete redundancy for every segment, we employ FEC within TCP. The sender transmits extra FEC segments, which encode previously transmitted segments, so that the receiver can repair a small number of losses. While the use of FEC for transport has been explored in the past, to our knowledge this is the first specification to place FEC within TCP in a way that is incrementally deployable across today's networks with middleboxes.

## 1.1. 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].

## 2. Protocol Overview

The key idea in TCP-IR is the judicious introduction of a small number of checksum or XOR segments into TCP's data stream such that a receiver can immediately recover lost segment(s) without the need for retransmissions. The core design challenge is in injecting out of band XOR segments within the regular data stream. We outline the main aspects of the protocol below.

Several of the design choices we made are rooted in our measurements and observations of Internet loss patterns as documented in [RECOVERY-SIGCOMM13]. We find that among the flows experiencing losses, most flows lose only one or two consecutive packets, commonly at the tail of a burst. As an example, for bursts of at most 10 packets, ~35% experienced exactly one loss, and an additional 10% experienced exactly two losses. Further, the latter packets for any given burst size are more likely to be lost. Given these findings, we chose a simple XOR-based encoding scheme that can perform instant recovery of a small amount (one or two segments) of packet loss.

A TCP-IR sender and receiver first negotiate the use of instant recovery in the initial handshake. If both hosts support the use of instant recovery, every packet in the connection includes a TCP-IR option.

TCP-IR sender:



1. Periodically in every round-trip time, a TCP-IR sender places the XOR of newly transmitted segments into a single MSS-length checksum packet. The XOR is only computed for new segments not previously included in checksums.
2. Regardless of the sizes of original segments, the sender computes the XORs along MSS byte boundaries. Because every packet carries a payload of at most MSS bytes, such an encoding guarantees that the receiver can instantly recover any single packet loss.
3. The encoded XOR packet uses the same sequence number as the first segment it encodes. The encoded packet carries a flag in the TCP-IR option signaling that the payload is encoded. A receiver uses the flag to disambiguate an encoded packet from a regular (re)transmission, since both segments carry the same sequence number. The option also includes the number of bytes that the payload encodes.

There is no reliability provided for the XOR segments.

TCP-IR receiver:

1. A receiver first establishes if the payload of the received segment is encoded, by checking a flag in the TCP-IR option.
2. Once the receiver establishes that the payload is encoded, it obtains the encoded range of bytes by using the sequence number of the TCP-IR packet and the the number of bytes encoded.
3. The receiver checks for holes in the encoded range. If it received the entire sequence range, the receiver drops the encoded packet. Otherwise, if it is missing at most MSS contiguous bytes, the receiver uses the encoded payload to recover the lost sequence range and forwards it to the regular reception routine, thus allowing 0-RTT recovery.
4. For the purpose of recovering lost segments, a receiver buffers the last fifteen in-order MSS blocks that it ACKed, even if the application layer has already consumed these blocks. Because an encoded packet is the XOR of at most sixteen MSS segments, the receiver can recover any single lost packet by computing the XOR of the encoded payload and the buffered data in the encoding range.
5. If too much data is missing for the encoded packet to recover, the receiver sends a duplicate ACK. This ACK informs the sender that a recovery failed and also denotes the byte ranges lost via the TCP-IR option. The sender marks the byte ranges as lost and triggers a fast retransmit and recovery.

6. TCP-IR does not circumvent congestion control. If the receiver were to simply ACK a recovered packet, it would mask the loss and prevent congestion control during a known loss episode. To perform congestion window reduction upon a successful recovery at the receiver, TCP-IR uses a mechanism similar to explicit congestion notification (ECN). Upon a successful recovery, the receiver enables an R\_SUCC flag in the TCP-IR option in each outgoing ACK. The sender in turn triggers a congestion window reduction and sets an R\_ACK flag in the TCP-IR option of the next packet sent to the receiver. Once the receiver observes R\_ACK in an incoming packet, indicating that the sender reduced the congestion window, it disables R\_SUCC for future packets.

Figure 1 gives an example of TCP-IR in action.

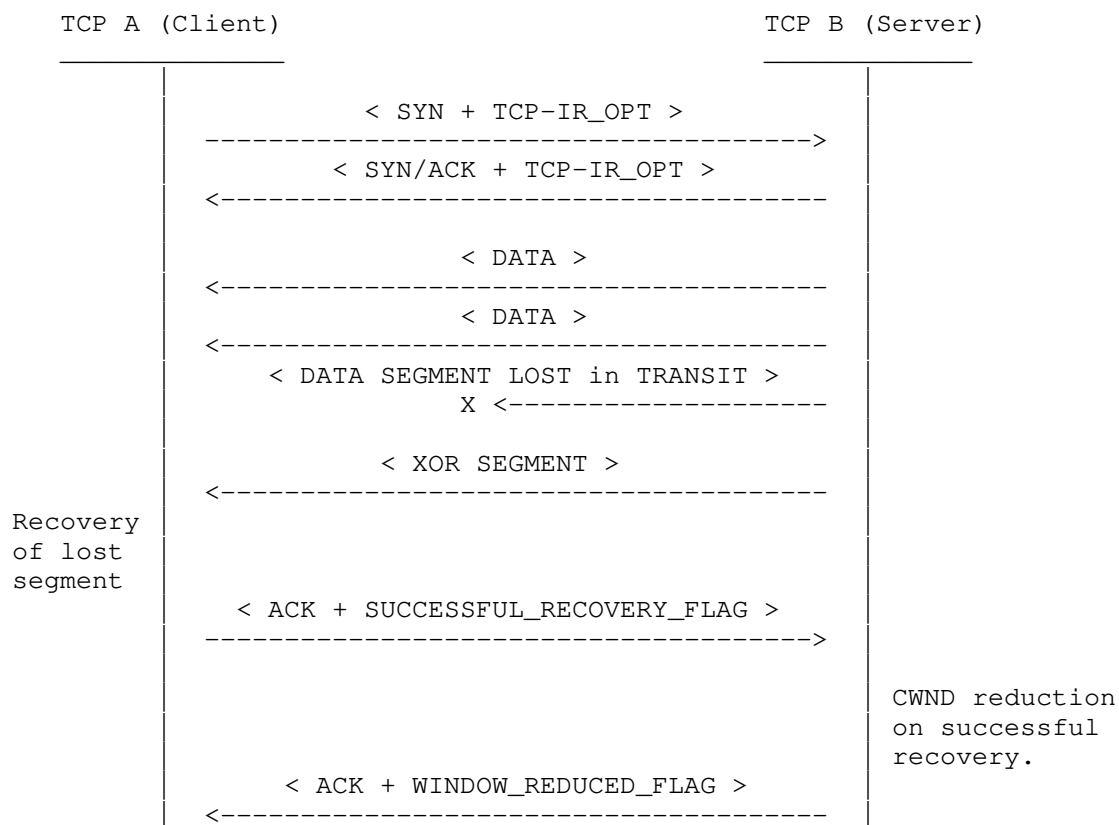


Figure 1: Sample flow using TCP-IR

### 3. Protocol Details

In the following, we describe the TCP-IR option design, negotiation of instant recovery, the supported encoding schemes, and finally the sender and receiver side algorithms.

### 3.1. TCP-IR Option

Both the server and the client use a new option to perform the following:

- o Negotiate the use of TCP-IR, including the encoding type.
- o Distinguish encoded packets from regular packets.
- o Communicate the number of encoded bytes in an XOR packet.
- o Acknowledge the recovery of segments and congestion window reductions.
- o Indicate the loss of segments.

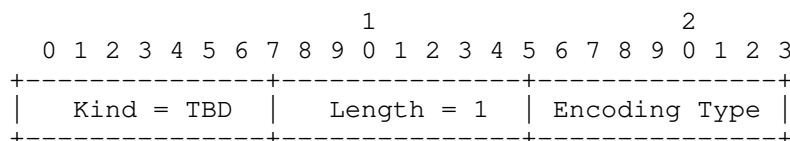


Figure 2: TCP-IR Option format for packets with SYN flag set

During the initial handshake (for packets with the SYN flag set), the option has the format shown in Figure 2. It contains the following fields:

Kind (8 bits)

This MUST be set to the option number for TCP-IR to be determined by IANA.

Length (8 bits)

This MUST be set to the length of the TCP option in octets; its value MUST be 1.

Encoding Type (8 bits)

This SHOULD be set to a value corresponding to a supported encoding type (see Section 3.3).

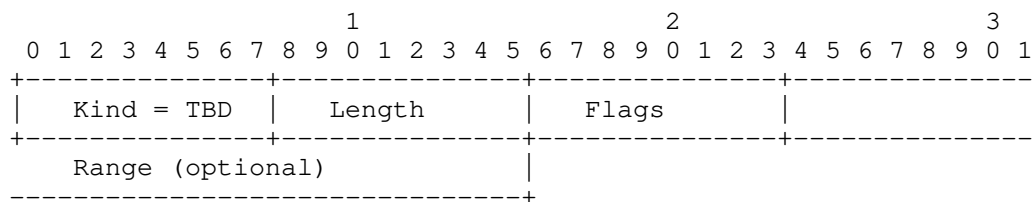


Figure 3: TCP-IR Option format (except for packets with SYN flag set)

For all other packets, the option has the format shown in Figure 3. It contains the following fields:

**Kind (8 bits)**

This MUST be set to the option number for TCP-IR to be determined by IANA.

**Length (8 bits)**

This MUST be set to the length of the TCP option in octets; its value MUST be 1, or 4 (if the "Range" field is appended).

**Flags (8 bits)**

The field can carry the following flags (each represented by one bit):

Bit	Flag Name	Description
0	R_CWR	Congestion Window Reduction Acknowledgement
1	R_SUCCESS	Recovery successful
2	R_FAIL	Recovery failed
3	ENCODED	Packet is encoded
4-7		Unused

The unused bits SHOULD NOT be set.

**Range (24 bits, optional)**

This field is only used if either the ENCODED or R\_FAIL bit in the "Flags" field is set. If the ENCODED bit is set, this field specifies the number of bytes encoded in the payload. If the R\_FAIL bit is set, this field specifies the number of bytes considered lost (see Section 3.4 and Section 3.6).

### 3.2. Negotiation

TCP-IR MUST be explicitly negotiated during the initial handshake. If the negotiation succeeds, both endpoints can send and receive TCP-IR packets. More specifically TCP-IR is enabled if:

1. The receiver sends the SYN packet carrying the TCP-IR option. The encoding type field MUST carry a valid encoding type (see Section 3.3).
2. The sender responds with a SYN/ACK carrying the TCP-IR option. The encoding type field MUST carry the same encoding type as the option in the corresponding SYN packet.
3. All following packets (transmitted by either sender or receiver) MUST carry the TCP-IR option.

If any endpoint receives a packet (after negotiation succeeds) that does not carry the TCP-IR option, the connection MUST be reset. This is necessary because a receiver can no longer distinguish between regular and TCP-IR packets. We recommend tracking these cases to avoid TCP-IR negotiation on future connections.

### 3.3. Encoding Types

The client can select the encoding type to be used by the TCP-IR module but both endpoints have to support it and agree on it during the initial handshake (see Section 3.2).

Currently the following encoding types are supported and are therefore valid values in the "Encoding Type" field of the TCP-IR option during negotiation:

Value	Type	Description
0	Undefined	
1	Basic XOR	TCP-IR packets carry the XOR of every MSS length segment. One TCP-IR packet encodes up to 16 MSS length segments.
2	Interleaved XOR	TCP-IR packets carry the XOR of every other MSS length segment. One TCP-IR packet encodes up to 8 MSS length segments.
3-255	Undefined	

We selected these encoding types to demonstrate the flexibility of TCP-IR with respect to user preferences (like the acceptable amount of redundancy) and connection properties. Basic XOR can recover a single segment loss of up to MSS bytes in the encoding range. Interleaved XOR enables the recovery of two consecutive segments of up to MSS bytes. This makes Interleaved XOR suitable for connections observing bursty losses, but can double the number of generated TCP-IR packets.

The currently supported encoding types use the MSS value to determine the block size for the encoding process. If the MSS value changes after the initial handshake, TCP-IR MUST be disabled for the remainder of the connection.

### 3.4. TCP-IR Sender

All packets after the initial handshake carry the TCP-IR option to ensure that the receiver can always distinguish regular packets from encoded packets (packets carrying a payload encoded by the TCP-IR module), or detect the removal of the option by a middlebox. Encoded packets MUST set the ENCODED bit in the "Flags" field of the TCP-IR option; all other packets MUST NOT set the ENCODED bit.

The TCP-IR packet MUST use the same sequence number as the first byte it encodes. This prevents enforcing reliability for encoded packets as well as the overhead of specifying the index of the first encoded byte in a separate option field.

In addition to that, the option in encoded packets MUST carry the "Range" field. The value in the "Range" field specifies the index of the byte after the last encoded byte in the payload relative to the sequence number of the encoded packet.

TCP-IR adds a short delay in the transmission of encoded packets to reduce the probability of losing both the original transmission and the encoded packet in the same loss burst.

The encoding and transmission routine works as follows:

1. Before a regular data packet is forwarded to the IP layer, the TCP-IR timer is armed (unless the timer is already armed). In our prototype implementation the timer is set to a value of  $RTT/4$ .
2. Once the timer fires, all transmitted segments not encoded before are now encoded according to the negotiated encoding type and the corresponding encoded packets are transmitted immediately. The maximum number of MSS length segments which can be encoded in a single TCP-IR packet depends on the negotiated encoding type (see Section 3.3). As a result, the number of encoded packets created in this step depends on the encoding type and the number of previously un-encoded segments. The option fields in the encoded packet are populated as described in Section 3.1.

### 3.5. TCP-IR Receiver

Receivers distinguish TCP-IR packets from regular packets by checking the ENCODED bit in the "Flags" field of the TCP-IR option. Encoded

packets are forwarded to the TCP-IR reception routine (described below). If the packet does not carry the TCP-IR option it is discarded and TCP-IR is disabled for the remainder of the connection. To inform the sender that TCP-IR can no longer be used, the receiver sends an acknowledgement without the TCP-IR option.

Additionally, the regular reception routine is modified as follows. The last 15 ACKed MSS length segments remain in the buffer, even if the application layer has already consumed these segments. Segments received out-of-order are already buffered by default and cannot be consumed by the application layer. Since a single TCP-IR packet encodes at most 8 (interleaved XOR) or 16 (basic XOR) MSS length segments, any single segment loss (up to MSS length) in the encoding range can be recovered by the decoder.

The reception routine for TCP-IR packets works as follows:

1. The encoding range of the TCP-IR packet is extracted. As mentioned earlier, the sequence number of the packet specifies the sequence number of the first encoded byte. The sequence number plus the value stored in the "Range" field in the TCP-IR option minus 1 specifies the sequence number of the last encoded byte.
2. If all bytes in the encoding range were already received, skip to Step 5.
3. If lost segments in the encoding range can be recovered (in the case of XOR encoding, a loss of at most one MSS length segment in the encoding range can be handled):
  - a. The lost segments are reconstructed. The matching packet headers are appended to the reconstructed segments and the packets are forwarded to the regular reception routine.
  - b. The R\_SUCCESS bit in the "Flags" field of the TCP-IR option is set for all future packets. This includes the (potentially delayed) acknowledgement for the recovered segment. Further details are described in Section 3.6.
4. If none of the segments in the encoding range are recoverable:
  - a. The sequence number of the last byte lost is extracted. The offset between the sequence number of the next expected byte (RCV.NXT) and the last byte lost defines the loss range.
  - b. An acknowledgement is generated with the following requirements for the TCP-IR option.
    - + The R\_FAIL bit in the "Flags" field is set.

- + The option carries the "Range" field. The "Range" field encodes the loss range, as described above. The context is maintained, since the acknowledgement number will be set to RCV.NXT.
5. The TCP-IR packet is discarded.

### 3.6. Processing Acknowledgements

If a receiver instantly recovers losses we want to ensure the sender learns of it so as to not circumvent congestion control [RFC5681]. The R\_SUCCESS bit in the "Flags" field of the TCP-IR option informs the sender that the receiver successfully recovered a lost packet. Once the sender observes the R\_SUCCESS bit in a packet the following steps are executed:

1. The sender reduces its congestion window.
2. The sender sets the R\_CWR bit in the "Flags" field of the TCP-IR option in the next outgoing packet only.
3. The sender does not act on any future observations of the R\_SUCCESS bit being set until SND.UNA advances past the SND.NXT value observed at the time when Step 2 was executed. This ensures the congestion window is not reduced multiple times in the same loss episode.
4. Once the receiver observes the R\_CWR bit being set in any incoming packet, the R\_SUCCESS bit is reset for all future packets.

A failed recovery on the receiver side triggers an explicit acknowledgment sent to the sender to inform it about the segments that are considered lost. This is indicated by the R\_FAIL bit being set in the "Flags" field of the TCP-IR option. If the sender observes this bit being set, the following steps are executed:

1. The sender extracts the loss range from the "Range" field in the TCP-IR option. The sequence numbers of the first and last byte lost are defined by the acknowledgement number of the packet, and the acknowledgement number plus the loss range value.
2. The sender marks the appropriate byte range as lost and triggers Fast Retransmit/Recovery.

Explicit notification of loss ranges has the benefit that lost segments are retransmitted faster, avoiding the extra wait time until the RTO fires.

### 4. Interaction with middleboxes

An important design goal of TCP-IR is compatibility with middleboxes and support for graceful fallback to standard TCP behavior in



situations where middlebox interference prevents proper use of TCP-IR.

Even if hosts negotiate TCP-IR during the initial handshake, it is possible for a middlebox to strip the option from a later packet. To be robust to this, if either host receives a packet without the option, it **MUST** discard the packet and reset the connection. This is necessary since receivers are no longer able to distinguish TCP-IR packets from regular packets.

TCP-IR uses relative sequence numbers to convey metadata (such as the encoding range) between endpoints. This prevents issues in the cases of middleboxes performing sequence number translations.

Some problems caused by middlebox interference (and their solutions in TCP-IR) are not discussed in the current version of this draft:

- o Rewriting of the acknowledgement number if the acknowledged segment was not observed by the middlebox. With TCP-IR this can occur after recovering a lost segment. This issue can be circumvented by retransmitting the recovered segment, even though it is not needed by the other endpoint anymore. This plugs the "sequence hole" in the state of the middlebox.
- o Rewriting payloads of previously seen segments.
- o Packet coalescing and splitting.

## 5. Implementation and Performance

We implemented TCP-IR in Linux TCP in about 1600 lines of code. 20% of the modular implementation includes the parts common to both the sender and receiver, which are option encoding/decoding, and negotiation during connection setup. 50% of the implementation is the receiver components including detection of an encoded packet, decoding the TCP-IR payload, and generating the right acknowledgements upon a successful or failed recovery. The remainder 30% of the implementation is the sender component which consists mainly of payload encoding and transmission.

We conducted two kinds of experiments. The first was in an emulated setting using loss patterns similar to those observed in our measurement of real traffic. We used the netem module to emulate a 200 ms RTT and both random and correlated loss rates of 2%. TCP-IR reduced the latency for short transfers in lossy environments by 28% in the 90th percentile. The benefits diminish as the minimum number of RTTs necessary to complete the transaction increases (due to the message size) because the time to recover from losses no longer dominates the overall transmission time. TCP-IR is better suited for small transfers common in today's Web.

In the second set of experiments, we used the Web-page-replay tool and dummynet to replay HTTP resource transfers for actual Web page downloads through controlled, emulated network conditions. We tested a variety of popular Web sites, and ran separate tests for Web pages tailored for desktop and mobile clients. As an example, with TCP-IR, the New York Times website takes 15% less time in the 90th percentile until the first objects are rendered on the screen.

Details on performance experiments with TCP-IR are in [RECOVERY-SIGCOMM13].

## 6. Related Work

Applying FEC to transport, at nearly every layer, is an old idea. [Coding-IEEE2011] suggested placing network coding in TCP, and [CodedTCP-2013] extended this work by implementing a variant over UDP mainly for high loss rate wireless environments. Among others, [AdaptiveFEC-2004] and Tickoo et al. [LT-TCP-2005] explored extending TCP to incorporate FEC. Finally, Maelstrom is an FEC variant for long-range communication between data centers leveraging the benefits of combining and encoding data from multiple sources into a single stream [Maelstrom-2011]. The focus of all of this work is on the performance aspects of using FEC over lossy links. None address the protocol level changes required in TCP to incorporate FEC.

## 7. Security Considerations

The security considerations outlined in [RFC5681] apply to this document. At this time we did not find any additional security problems with TCP-IR.

## 8. IANA Considerations

The two Options for TCP-IR used during negotiation and subsequently in every packet of the connection require IANA allocate one value from the TCP option Kind namespace. Experimentation prior to the allocation SHOULD follow [EXPOPT] and use experimental option kind 254 and two magic bytes 0xDC60, and migrate to the new option after the allocation accordingly.

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

## 9. References

[RECOVERY-SIGCOMM13]

Flach, T., Dukkkipati, 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", Proceedings of the 2013 ACM SIGCOMM , 2013.

[Coding-IEEE2011]

Sundararajan, J., Shah, D., Medard, M., Jakubczak, S., Mitzenmacher, M., and J. Barros, "Network Coding Meets TCP: Theory and Implementation", Proceedings of the IEEE , 2011.

[CodedTCP-2013]

Kim, M., Cloud, J., ParandehGheibi, A., Urbina, L., Fouli, K., Leith, D., and M. Medard, "Network Coded TCP (CTCP)", arXiv:1212.2291. , 2013.

[AdaptiveFEC-2004]

Baldantoni, L., Lundqvist, H., and G. Karlsson, "Adaptive end-to-end FEC for improving TCP performance over wireless links", IEEE Communications Society , 2004.

[LT-TCP-2005]

Tickoo, O., Subramanian, V., Kalyanaraman, S., and K. Ramakrishnan, "LT-TCP: End-to-End Framework to improve TCP Performance over Networks with Lossy Channels ", Proc. of IWQoS , 2005.

[Maelstrom-2011]

Balakrishnan, M., Marian, T., Birman, K., Weatherspoon, H., and L. Ganesh, "Maelstrom: transparent error correction for communication between data centers ", IEEE/ACM Transactions on Networking , 2011.

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

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

[EXPOPT] Touch, J., "Shared Use of Experimental TCP Options", draft-ietf-tcpm-experimental-options-06 (work in progress), October 2012.

Authors' Addresses

Tobias Flach  
University of Southern California  
941 Bloom Walk  
Los Angeles, California 90089  
USA

Email: [flach@usc.edu](mailto:flach@usc.edu)  
URI: <http://nsl.cs.usc.edu/~tobiasflach>

Nandita Dukkhipati  
Google, Inc.  
1600 Amphitheatre Parkway  
Mountain View, California 94043  
USA

Email: [nanditad@google.com](mailto:nanditad@google.com)

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

Email: [ycheng@google.com](mailto:ycheng@google.com)

Barath Raghavan  
Google, Inc.  
1600 Amphitheatre Parkway  
Mountain View, California 94043  
USA

Email: [barath@google.com](mailto:barath@google.com)

TCP Maintenance and Minor Extensions  
(tcpm)  
Internet-Draft  
Updates: 793 (if approved)  
Intended status: Standards Track  
Expires: August 20, 2013

F. Gont  
UTN-FRH / SI6 Networks  
D. Borman  
Quantum Corporation  
February 16, 2013

On the Validation of TCP Sequence Numbers  
draft-gont-tcpm-tcp-seq-validation-00.txt

Abstract

When TCP receives packets that lie outside of the receive window, the corresponding packets are dropped and either an ACK, RST or no response is generated due to the out-of-window packet, with no further processing of the packet. Most of the time, this works just fine and TCP remains stable, especially when a TCP connection has unidirectional data flow. However, there are three scenarios in which packets that are outside of the receive window should still have their ACK field processed, or else a packet war will take place. The aforementioned issues have affected a number of popular TCP implementations, typically leading to connection failures, system crashes, or other undesirable behaviors. This document describes the three scenarios in which the aforementioned issues might arise, and formally updates RFC 793 such that these potential problems are mitigated.

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 August 20, 2013.

Copyright Notice

Copyright (c) 2013 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. TCP Sequence Number Validation . . . . .	3
3. Scenarios in which Undesirable Behaviors Might Arise . . . . .	4
3.1. TCP simultaneous open . . . . .	4
3.2. TCP self connects . . . . .	5
3.3. TCP simultaneous close . . . . .	6
3.4. Simultaneous Window Probes . . . . .	8
4. Updating RFC 793 . . . . .	9
4.1. TCP sequence number validation . . . . .	9
4.2. TCP self connects . . . . .	14
5. IANA Considerations . . . . .	14
6. Security Considerations . . . . .	14
7. Acknowledgements . . . . .	14
8. References . . . . .	14
8.1. Normative References . . . . .	14
8.2. Informative References . . . . .	15
Authors' Addresses . . . . .	15

## 1. Introduction

TCP processes incoming packets in in-sequence order. Packets that are not in-sequence but have data that lies in the receive window are queued for later processing. Packets that lie outside of the receive window are dropped and either an ACK, RST or no response is generated due to the out-of-window packet, with no further processing of the packet. Most of the time, this works just fine and TCP remains stable, especially when a TCP connection has unidirectional data flow.

However, there are three situations in which packets that are outside of the receive window should still have their ACK field processed. These situations arise during a simultaneous open, simultaneous window probes and a simultaneous close. In all three of these cases, a packet will arrive with a sequence number that is one to the left of the window, but the acknowledgement field has updated information that needs to be processed to avoid entering a packet war, in which both sides of the connection generate a response to the received packet, which just causes the other side to do the same thing. This issue has affected a number of popular TCP implementations, typically leading to connection failures, system crashes, or other undesirable behaviors.

Section 2 provides an overview of the TCP sequence number validation checks specified in RFC 793. Section 3 describes the three scenarios in which the current TCP sequence number validation checks can lead to undesirable behaviors. Section 4 formally updates RFC 793 such that these issues are mitigated.

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

## 2. TCP Sequence Number Validation

Section 3.3 of RFC 793 [RFC0793] specifies (in pp. 25-26) how the TCP sequence number of incoming segments is to be validated. It summarizes the validation of the TCP sequence number with the following table:

Segment Length	Receive Window	Test
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND or RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

RFC 793 states that if an incoming segment is not acceptable, an acknowledgment should be sent in reply (unless the RST bit is set), and that after sending the acknowledgment, the unacceptable segment should be dropped.

Section 3.9 of RFC 793 repeats (in pp. 69-76) the same validation checks when describing the processing of incoming TCP segments meant for connections that are in the SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, or TIME-WAIT states (i.e., any state other than CLOSED, LISTEN, or SYN-SENT).

A key problem with the aforementioned checks is that it assumes that a segment must be processed only if a portion of it overlaps with the receive window. However, there are some cases in which the Acknowledgement information in an incoming segment needs to be processed by TCP even if the contents of the segment does not overlap with the receive window. Otherwise, the TCP state machine may become dead-locked, and this situation may result in undesirable behaviors such as system crashes.

### 3. Scenarios in which Undesirable Behaviors Might Arise

The following subsections describe the three scenarios in which the TCP Sequence Number validation specified in RFC 793 (and described in Section 2 of this document) could result in undesirable behaviors.

#### 3.1. TCP simultaneous open

The following figure illustrates a typical "simultaneous open" attempt.



TCP A		TCP B
1. CLOSED		CLOSED
2. SYN-SENT	--> <SEQ=100><CTL=SYN>	...
3. SYN-RECEIVED	<--- <SEQ=300><CTL=SYN>	<--- SYN-SENT
4.	... <SEQ=100><CTL=SYN>	--> SYN-RECEIVED
5.	--> <SEQ=100><ACK=301><CTL=SYN,ACK>	...
6.	<--- <SEQ=300><ACK=101><CTL=SYN,ACK>	<---
7.	... <SEQ=100><ACK=301><CTL=SYN,ACK>	-->
8.	--> <SEQ=100><ACK=301><CTL=SYN,ACK>	...
9.	<--- <SEQ=300><ACK=101><CTL=SYN,ACK>	<---
10.	... <SEQ=100><ACK=301><CTL=ACK>	-->

#### (Failed) Simultaneous Connection Synchronization

In line 2, TCP A performs an "active open" by sending a SYN segment to TCP B, and enters the SYN-SENT state. In line 3, TCP B performs an "active open" by sending a SYN segment to TCP A, and enters the "SYN-SENT" state; when TCP A receives this SYN segment sent by TCP B, it enters the SYN-RECEIVED state, and its RCV.NXT becomes 301. In line 4, similarly, when TCP B receives the SYN segment sent by TCP A, it enters the SYN-RECEIVED STATE and its RCV.NXT becomes 101. In line 5, TCP A sends a SYN/ACK in response to the received SYN segment from line 3. In line 6, TCP B sends a SYN/ACK in response to the received SYN segment from line 4. In line 7, TCP B receives the SYN/ACK from line 5. In line 8, TCP A receives the SYN/ACK from line 6, which fails the TCP Sequence Number validation check. As a result, the received packet is dropped, and a SYN/ACK is sent in response. In line 9, TCP B processes the SYN/ACK from line 7, which fails the TCP Sequence Number validation check. As a result, the received packet is dropped, and a SYN/ACK is sent in response. In line 10, the SYN/ACK from line 9 arrives at TCP B. The segment exchange from lines 8-10 will continue forever (with both TCP end-points will be stuck in the SYN-RECEIVED state), thus leading to a SYN/ACK war.

### 3.2. TCP self connects

Some systems have been found to be unable to process TCP connection requests in which the source endpoint {Source Address, Source Port}

is the same as the destination end-point {Destination Address, Destination Port}. Such a scenario might arise e.g. if a process creates a socket, bind()s a local end-point (IP address and TCP port), and then issues a connect() to the same end-point as that specified to bind().

While not widely employed in existing applications, such a socket could be employed as a "full-duplex pipe" for Inter-Process Communication (IPC).

This scenario is described in detail in pp. 960-962 of [Wright1994].

The aforementioned scenario has been reported to cause malfunction of a number of implementations [CERT1996], and has been exploited in the past to perform Denial of Service (DoS) attacks [Meltman1997] [CPNI-TCP].

While this scenario is not common in the real world, TCP should nevertheless be able to process them without the need of any "extra" code: a SYN segment in which the source end-point {Source Address, Source Port} is the same as the destination end-point {Destination Address, Destination Port} should result in a "simultaneous open" scenario, such as the one described in page 32 of RFC 793 [RFC0793]. Therefore, those TCP implementations that correctly handle simultaneous opens should already be prepared to handle these unusual TCP segments.

### 3.3. TCP simultaneous close

The following figure illustrates a typical "simultaneous close" attempt, in which the FIN segments sent by each TCP end-point cross each other in the network.

TCP A		TCP B
1. ESTABLISHED		ESTABLISHED
2. FIN-WAIT-1	--> <SEQ=100><ACK=300><CTL=FIN,ACK> ...	
3. CLOSING	<--- <SEQ=300><ACK=100><CTL=FIN,ACK> <--- FIN-WAIT-1	
4.	... <SEQ=100><ACK=300><CTL=FIN,ACK> --> CLOSING	
5.	--> <SEQ=100><ACK=301><CTL=FIN,ACK> ...	
6.	<--- <SEQ=300><ACK=101><CTL=FIN,ACK> <---	
7.	... <SEQ=100><ACK=301><CTL=FIN,ACK> -->	
8.	--> <SEQ=100><ACK=301><CTL=FIN,ACK> ...	
9.	<--- <SEQ=300><ACK=101><CTL=FIN,ACK> <---	
10.	... <SEQ=100><ACK=301><CTL=FIN,ACK> -->	

#### (Failed) Simultaneous Connection Termination

In line 1, we assume that both end-points of the connection are in the ESTABLISHED state. In line 2, TCP A performs an "active close" by sending a FIN segment to TCP B, thus entering the FIN-WAIT-1 state. In line 3, TCP B performs an active close sending a FIN segment to TCP A, thus entering the FIN-WAIT-1 state; when this segment is processed by TCP A, it enters the CLOSING state (and its RCV.NXT becomes 301).

Both FIN segments cross each other on the network, thus resulting in a "simultaneous connection termination" (or "simultaneous close") scenario.

In line 4, the FIN segment sent by TCP A arrives to TCP B, causing it to transition to the CLOSING state (at this point, TCP B's RCV.NXT becomes 101). In line 5, TCP A acknowledges the receipt of the TCP B's FIN segment, and also sets the FIN bit in the outgoing segment (since it has not yet been acknowledged). In line 6, TCP B acknowledges the receipt of TCP A's FIN segment, and also sets the FIN bit in the outgoing segment (since it has not yet been acknowledged). In line 7, the FIN/ACK from line 5 arrives at TCP B. In line 8, the FIN/ACK from line 6 fails the TCP sequence number validation check, and thus elicits a ACK segment (the segment also contains the FIN bit set, since it had not yet been acknowledged). In line 9, the FIN/ACK from line 7 fails the TCP sequence number

validation check, and hence elicits an ACK segment (the segment also contains the FIN bit set, since it had not yet been acknowledged). In line 10, the FIN/ACK from line 8 finally arrives at TCP B.

The packet exchange from lines 8-10 will repeat indefinitely, with both TCP end-points stuck in the CLOSING state, thus leading to a "FIN war": each FIN/ACK segment sent by a TCP will elicit a FIN/ACK from the other TCP, and each of these FIN/ACKs will in turn elicit more FIN/ACKs.

### 3.4. Simultaneous Window Probes

The following figure illustrates a scenario in which the "persist timer" at both TCP end-points expires, and both TCP end-points send a "window probes" that cross each other in the network.

TCP A		TCP B
1. ESTABLISHED		ESTABLISHED
2.	(both TCP windows open)	
3.	--> <SEQ=100><DATA=1><ACK=300><CTL=ACK> ...	
4.	<-- <SEQ=300><DATA=1><ACK=100><CTL=ACK> <--	
5.	... <SEQ=100><DATA=1><ACK=300><CTL=ACK> -->	
6.	--> <SEQ=100><ACK=301><CTL=ACK>	...
7.	<-- <SEQ=300><ACK=101><CTL=ACK>	<--
8.	... <SEQ=100><ACK=301><CTL=ACK>	-->
9.	--> <SEQ=100><ACK=301><CTL=ACK>	...
10.	<-- <SEQ=300><ACK=101><CTL=ACK>	<--
11.	... <SEQ=100><ACK=301><CTL=ACK>	-->

(Failed) Simultaneous Connection Termination

In line 1, we assume that both end-points of the connection are in the ESTABLISHED state; additionally, TCP A's RCV.NXT is 300, while TCP B's RCV.NXT is 100, and the receive window (RCV.WND) at both TCP end-points is 0. In line 2, both TCP windows open. In line 3, the "persist timer" at TCP A expires, and hence TCP A sends a "Window Probe". In line 4, the "persist timer" at TCP B expires, and hence

TCP B sends a "Window Probe".

Both Window Probes cross each other in the network.

When this probe arrives at TCP A, TCP A's RCV.NXT becomes 301, and an ACK segment is sent to advertise the new window (this ACK is shown in line 6). In line 5, TCP A's Window Probe from line 3 arrives at TCP B. TCP B's RCV-WND becomes 101. In line 6, TCP A sends the ACK to advertise the new window. In line 7, TCP B sends an ACK to advertise the new Window. When this ACK arrives at TCP A, the TCP Sequence Number validation fails, since SEG.SEQ=300 and RCV.NXT=301. Therefore, this segment elicits a new ACK (meant to re-synchronize the sequence numbers). In line 8, the ACK from line 6 arrives at TCP B. The TCP sequence number validation for this segment fails, since SEG.SEQ=100 AND RCV.NXT=101. Therefore, this segment elicits a new ACK (meant to re-synchronize the sequence numbers).

Line 9 and line 11 shows the ACK elicited by the segment from line 7, while line 10 shows the ACK elicited by the segment from line 8. The sequence numbers of these ACK segments will be considered invalid, and hence will elicit further ACKs. Therefore, the segment exchange from lines 9-11 will repeat indefinitely, thus leading to an "ACK war".

#### 4. Updating RFC 793

##### 4.1. TCP sequence number validation

The following text from Section 3.3 (pp. 25-26) of [RFC0793]:

A segment is judged to occupy a portion of valid receive sequence space if

$$\text{RCV.NXT} \leq \text{SEG.SEQ} < \text{RCV.NXT} + \text{RCV.WND}$$

or

$$\text{RCV.NXT} \leq \text{SEG.SEQ} + \text{SEG.LEN} - 1 < \text{RCV.NXT} + \text{RCV.WND}$$

The first part of this test checks to see if the beginning of the segment falls in the window, the second part of the test checks to see if the end of the segment falls in the window; if the segment passes either part of the test it contains data in the window.

Actually, it is a little more complicated than this. Due to zero windows and zero length segments, we have four cases for the acceptability of an incoming segment:

Segment Length	Receive Window	Test
0	0	$\text{SEG.SEQ} = \text{RCV.NXT}$
0	>0	$\text{RCV.NXT} \leq \text{SEG.SEQ} < \text{RCV.NXT} + \text{RCV.WND}$
>0	0	not acceptable
>0	>0	$\text{RCV.NXT} \leq \text{SEG.SEQ} < \text{RCV.NXT} + \text{RCV.WND}$ or $\text{RCV.NXT} \leq \text{SEG.SEQ} + \text{SEG.LEN} - 1 < \text{RCV.NXT} + \text{RCV.WND}$

is replaced with:

A segment is judged to occupy a portion of valid receive sequence space if

$$\text{RCV.NXT}-1 \leq \text{SEG.SEQ} < \text{RCV.NXT}+\text{RCV.WND}$$

or

$$\text{RCV.NXT}-1 \leq \text{SEG.SEQ}+\text{SEG.LEN}-1 < \text{RCV.NXT}+\text{RCV.WND}$$

The first part of this test checks to see if the beginning of the segment falls in the window (or one byte to the left of the window), the second part of the test checks to see if the end of the segment falls in the window (or one byte to the left of the window); if the segment passes either part of the test it contains data in the window or control information that needs to be processed by TCP.

Actually, it is a little more complicated than this. Due to zero windows and zero length segments, we have four cases for the acceptability of an incoming segment:

Segment Length	Receive Window	Test
0	0	$\text{RCV.NXT}-1 \leq \text{SEG.SEQ} \leq \text{RCV.NXT}$
0	>0	$\text{RCV.NXT}-1 \leq \text{SEG.SEQ} < \text{RCV.NXT}+\text{RCV.WND}$
>0	0	not acceptable
>0	>0	$\text{RCV.NXT}-1 \leq \text{SEG.SEQ} < \text{RCV.NXT}+\text{RCV.WND}$ or $\text{RCV.NXT}-1 \leq \text{SEG.SEQ}+\text{SEG.LEN}-1 < \text{RCV.NXT}+\text{RCV.WND}$

Additionally, the following text from Section 3.9 (pp.69-70) of [RFC0793]:

Segments are processed in sequence. Initial tests on arrival are used to discard old duplicates, but further processing is done in SEG.SEQ order. If a segment's contents straddle the boundary between old and new, only the new parts should be processed.

There are four cases for the acceptability test for an incoming segment:

Segment Length	Receive Window	Test
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND or RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

If the RCV.WND is zero, no segments will be acceptable, but special allowance should be made to accept valid ACKs, URGs and RSTs.

If an incoming segment is not acceptable, an acknowledgment should be sent in reply (unless the RST bit is set, if so drop the segment and return):

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

After sending the acknowledgment, drop the unacceptable segment and return.

In the following it is assumed that the segment is the idealized segment that begins at RCV.NXT and does not exceed the window. One could tailor actual segments to fit this assumption by trimming off any portions that lie outside the window (including SYN and FIN), and only processing further if the segment then begins at RCV.NXT. Segments with higher beginning sequence numbers may be held for later processing.

is replaced with:



Segments are processed in sequence. Initial tests on arrival are used to discard old duplicates, but further processing is done in SEG.SEQ order. If a segment's contents straddle the boundary between old and new, only the new parts should be processed. Acknowledgement information must still be processed when the contents of the incoming segment are one byte to the left of the receive window.

This is to handle simultaneous opens, simultaneous closes, and simultaneous window probes.

There are four cases for the acceptability test for an incoming segment:

Segment Length	Receive Window	Test
0	0	RCV.NXT-1 =< SEG.SEQ <= RCV.NXT
0	>0	RCV.NXT-1 =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT-1 =< SEG.SEQ < RCV.NXT+RCV.WND or RCV.NXT-1 =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

If the RCV.WND is zero, no segments will be acceptable, but special allowance should be made to accept valid ACKs, URGs and RSTs.

If an incoming segment is not acceptable, an acknowledgment should be sent in reply (unless the RST bit is set, if so drop the segment and return):

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

After sending the acknowledgment, drop the unacceptable segment and return.

In the following it is assumed that the segment is the idealized segment that begins at RCV.NXT and does not exceed the window. One could tailor actual segments to fit this assumption by trimming off any portions that lie outside the window (including SYN and FIN). Segments with higher beginning sequence numbers may be held for later processing. Acknowledgement information must still be processed when the contents of the incoming segment are one byte to the left of the receive window.

#### 4.2. TCP self connects

TCP MUST be able to gracefully handle connection requests (i.e., SYN segments) in which the source end-point (IP Source Address, TCP Source Port) is the same as the destination end-point (IP Destination Address, TCP Destination Port). Such segments MUST result in a TCP "simultaneous open", such as the one described in page 32 of RFC 793 [RFC0793].

Those TCP implementations that correctly handle simultaneous opens are expected to gracefully handle this scenario.

#### 5. IANA Considerations

This document has no IANA actions. The RFC Editor is requested to remove this section before publishing this document as an RFC.

#### 6. Security Considerations

This document describes a problem found in the current validation rules for TCP sequence numbers. The aforementioned problem has affected some popular TCP implementations, typically leads to connection failures, system crashes, or other undesirable behaviors. This document formally updates RFC 793, such that the aforementioned issues are eliminated.

#### 7. Acknowledgements

This document originated from a discussion about this topic (at IETF 73, Minneapolis) between both co-authors of this document.

#### 8. References

##### 8.1. Normative References

- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

## 8.2. Informative References

[CERT1996]

CERT, "CERT Advisory CA-1996-21: TCP SYN Flooding and IP Spoofing Attacks", 1996,  
<<http://www.cert.org/advisories/CA-1996-21.html>>.

[CPNI-TCP]

Gont, F., "CPNI Technical Note 3/2009: Security Assessment of the Transmission Control Protocol (TCP)", 2009, <<http://www.gont.com.ar/papers/tn-03-09-security-assessment-TCP.pdf>>.

[Meltman1997]

Meltman, "new TCP/IP bug in win95. Post to the bugtraq mailing-list", 1996,  
<<http://insecure.org/sploits/land.ip.DOS.html>>.

[Wright1994]

Wright, G. and W. Stevens, "TCP/IP Illustrated, Volume 2: The Implementation", Addison-Wesley, 1994.

## Authors' Addresses

Fernando Gont  
UTN-FRH / SI6 Networks  
Evaristo Carriego 2644  
Haedo, Provincia de Buenos Aires 1706  
Argentina

Phone: +54 11 4650 8472  
Email: [fgont@si6networks.com](mailto:fgont@si6networks.com)  
URI: <http://www.si6networks.com>

David Borman  
Quantum Corporation  
1155 Centre Pointe Drive, Suite 1  
Mendota Heights, MN 55120  
U.S.A.

Phone: 651-688-4394  
Email: [david.borman@quantum.com](mailto:david.borman@quantum.com)



TCP Maintenance (TCPM)  
Internet-Draft  
Intended status: Standards Track  
Expires: November 24, 2013

D. Borman  
Quantum Corporation  
B. Braden  
University of Southern  
California  
V. Jacobson  
Packet Design  
R. Scheffenegger, Ed.  
NetApp, Inc.  
May 23, 2013

TCP Extensions for High Performance  
draft-ietf-tcpm-1323bis-14

Abstract

This document specifies a set of TCP extensions to improve performance over paths with a large bandwidth \* delay product and to provide reliable operation over very high-speed paths. It defines TCP options for scaled windows and timestamps. The timestamps are used for two distinct mechanisms, RTTM (Round Trip Time Measurement) and PAWS (Protection Against Wrapped Sequences).

This document updates and obsoletes RFC 1323.

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 November 24, 2013.

Copyright Notice

Copyright (c) 2013 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 . . . . .	4
1.1.	TCP Performance . . . . .	4
1.2.	TCP Reliability . . . . .	5
1.3.	Using TCP options . . . . .	6
1.4.	Terminology . . . . .	7
2.	TCP Window Scale Option . . . . .	8
2.1.	Introduction . . . . .	8
2.2.	Window Scale Option . . . . .	8
2.3.	Using the Window Scale Option . . . . .	9
2.4.	Addressing Window Retraction . . . . .	10
3.	TCP Timestamps option . . . . .	12
3.1.	Introduction . . . . .	12
3.2.	Timestamps option . . . . .	12
3.3.	The RTTM Mechanism . . . . .	13
3.4.	Updating the RTO value . . . . .	15
3.5.	Which Timestamp to Echo . . . . .	15
4.	PAWS - Protection Against Wrapped Sequence Numbers . . . . .	18
4.1.	Introduction . . . . .	18
4.2.	The PAWS Mechanism . . . . .	18
4.3.	Basic PAWS Algorithm . . . . .	19
4.4.	Timestamp Clock . . . . .	21
4.5.	Outdated Timestamps . . . . .	23
4.6.	Header Prediction . . . . .	23
4.7.	IP Fragmentation . . . . .	25
4.8.	Duplicates from Earlier Incarnations of Connection . . . . .	25
5.	Conclusions and Acknowledgements . . . . .	25
6.	Security Considerations . . . . .	26
7.	IANA Considerations . . . . .	27
8.	References . . . . .	28
8.1.	Normative References . . . . .	28
8.2.	Informative References . . . . .	28
	Appendix A. Implementation Suggestions . . . . .	31
	Appendix B. Duplicates from Earlier Connection Incarnations . . . . .	32
	B.1. System Crash with Loss of State . . . . .	32
	B.2. Closing and Reopening a Connection . . . . .	33
	Appendix C. Summary of Notation . . . . .	34
	Appendix D. Event Processing Summary . . . . .	35
	Appendix E. Timestamps Edge Cases . . . . .	41
	Appendix F. Window Retraction Example . . . . .	41
	Appendix G. RTO calculation modification . . . . .	42
	Appendix H. Changes from RFC 1323 . . . . .	43
	Authors' Addresses . . . . .	45

## 1. Introduction

The TCP protocol [RFC0793] was designed to operate reliably over almost any transmission medium regardless of transmission rate, delay, corruption, duplication, or reordering of segments. Over the years, advances in networking technology has resulted in ever-higher transmission speeds, and the fastest paths are well beyond the domain for which TCP was originally engineered.

This document defines a set of modest extensions to TCP to extend the domain of its application to match the increasing network capability. It is an update to and obsoletes [RFC1323], which in turn is based upon and obsoletes [RFC1072] and [RFC1185].

Changes between [RFC1323] and this document are detailed in Appendix H.

For brevity, the full discussions of the merits and history behind the TCP options defined within this document have been omitted. [RFC1323] should be consulted for reference. It is recommended that a modern TCP stack implements and make use of the extensions described in this document.

### 1.1. TCP Performance

TCP performance problems arise when the bandwidth \* delay product is large. A network having such paths is referred to as "long, fat network" (LFN).

There are two fundamental performance problems with basic TCP over LFN paths:

#### (1) Window Size Limit

The TCP header uses a 16 bit field to report the receive window size to the sender. Therefore, the largest window that can be used is  $2^{16} = 64$  KiB. For LFN paths where the bandwidth \* delay product exceeds 64 KiB, the receive window limits the maximum throughput of the TCP connection over the path, i.e., the amount of unacknowledged data that TCP can send in order to keep the pipeline full.

To circumvent this problem, Section 2 of this memo defines a TCP option, "Window Scale", to allow windows larger than  $2^{16}$ . This option defines an implicit scale factor, which is used to multiply the window size value found in a TCP header to obtain the true window size.



## (2) Recovery from Losses

Packet losses in an LFN can have a catastrophic effect on throughput.

To generalize the Fast Retransmit/Fast Recovery mechanism to handle multiple packets dropped per window, selective acknowledgments are required. Unlike the normal cumulative acknowledgments of TCP, selective acknowledgments give the sender a complete picture of which segments are queued at the receiver and which have not yet arrived.

Selective acknowledgements and their use are specified in separate documents, "TCP Selective Acknowledgment Options" [RFC2018], "An Extension to the Selective Acknowledgement (SACK) Option for TCP" [RFC2883], and "A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP" [RFC6675], and not further discussed in this document.

## 1.2. TCP Reliability

An especially serious kind of error may result from an accidental reuse of TCP sequence numbers in data segments. TCP reliability depends upon the existence of a bound on the lifetime of a segment: the "Maximum Segment Lifetime" or MSL.

Duplication of sequence numbers might happen in either of two ways:

### (1) Sequence number wrap-around on the current connection

A TCP sequence number contains 32 bits. At a high enough transfer rate, the 32-bit sequence space may be "wrapped" (cycled) within the time that a segment is delayed in queues.

### (2) Earlier incarnation of the connection

Suppose that a connection terminates, either by a proper close sequence or due to a host crash, and the same connection (i.e., using the same pair of port numbers) is immediately reopened. A delayed segment from the terminated connection could fall within the current window for the new incarnation and be accepted as valid.

Duplicates from earlier incarnations, case (2), are avoided by enforcing the current fixed MSL of the TCP specification, as explained in Section 4.8 and Appendix B. However, case (1), avoiding the reuse of sequence numbers within the same connection, requires an upper bound on MSL that depends upon the transfer rate, and at high

enough rates, a dedicated mechanism is required.

A possible fix for the problem of cycling the sequence space would be to increase the size of the TCP sequence number field. For example, the sequence number field (and also the acknowledgment field) could be expanded to 64 bits. This could be done either by changing the TCP header or by means of an additional option.

Section 4 presents a different mechanism, which we call PAWS (Protection Against Wrapped Sequence numbers), to extend TCP reliability to transfer rates well beyond the foreseeable upper limit of network bandwidths. PAWS uses the TCP Timestamps option defined in Section 3.2 to protect against old duplicates from the same connection.

### 1.3. Using TCP options

The extensions defined in this document all use TCP options.

When [RFC1323] was published, there was concern that some buggy TCP implementation might be crashed by the first appearance of an option on a non-`<SYN>` segment. However, bugs like that can lead to DOS attacks against a TCP. Research has shown that most TCP implementations will properly handle unknown options on non-`<SYN>` segments ([Medina04], [Medina05]). But it is still prudent to be conservative in what you send, and avoiding buggy TCP implementation is not the only reason for negotiating TCP options on `<SYN>` segments.

The window scale option negotiates fundamental parameters of the TCP session. Therefore, it is only sent during the initial handshake. Furthermore, the window scale option will be sent in a `<SYN,ACK>` segment only if the corresponding option was received in the initial `<SYN>` segment.

The Timestamps option may appear in any data or `<ACK>` segment, adding 12 bytes to the 20-byte TCP header. It is required that this TCP option will be sent on all non-`<SYN>` segments after an exchange of options on the `<SYN>` segments has indicated that both sides understand this extension.

Research has shown that the use of the Timestamps option to arrive at an optimal retransmission timeout value has only limited benefit ([Allman99]). However, there are other uses of the Timestamps option, such as the Eifel mechanism [RFC3522], [RFC4015], and PAWS (see Section 4) which improve overall TCP security and performance. The extra header bandwidth used by this option should be evaluated for the gains in performance and security in an actual deployment.

Appendix A contains a recommended layout of the options in TCP headers to achieve reasonable data field alignment.

Finally, we observe that most of the mechanisms defined in this document are important for LFN's and/or very high-speed networks. For low-speed networks, it might be a performance optimization to NOT use these mechanisms. A TCP vendor concerned about optimal performance over low-speed paths might consider turning these extensions off for low-speed paths, or allow a user or installation manager to disable them.

#### 1.4. 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].

In this document, these words will appear with that interpretation only when in UPPER CASE. Lower case uses of these words are not to be interpreted as carrying [RFC2119] significance.

## 2. TCP Window Scale Option

### 2.1. Introduction

The window scale extension expands the definition of the TCP window to 30 bits and then uses an implicit scale factor to carry this 30-bit value in the 16-bit Window field of the TCP header (SEG.WND in [RFC0793]). The exponent of the scale factor is carried in a TCP option, Window Scale. This option is sent only in a <SYN> segment (a segment with the SYN bit on), hence the window scale is fixed in each direction when a connection is opened.

The maximum receive window, and therefore the scale factor, is determined by the maximum receive buffer space. In a typical modern implementation, this maximum buffer space is set by default but can be overridden by a user program before a TCP connection is opened. This determines the scale factor, and therefore no new user interface is needed for window scaling.

### 2.2. Window Scale Option

The three-byte Window Scale option MAY be sent in a <SYN> segment by a TCP. It has two purposes: (1) indicate that the TCP is prepared to do both send and receive window scaling, and (2) communicate the exponent of a scale factor to be applied to its receive window. Thus, a TCP that is prepared to scale windows SHOULD send the option, even if its own scale factor is 1 and the exponent 0. The scale factor is limited to a power of two and encoded logarithmically, so it may be implemented by binary shift operations. The maximum scale exponent is limited to 14 for a maximum permissible receive window size of 1 GiB ( $2^{(14+16)}$ ).

TCP Window Scale Option (WSopt):

Kind: 3

Length: 3 bytes

Kind=3	Length=3	shift.cnt
1	1	1

This option is an offer, not a promise; both sides MUST send Window Scale options in their <SYN> segments to enable window scaling in either direction. If window scaling is enabled, then the TCP that sent this option will right-shift its true receive-window values by 'shift.cnt' bits for transmission in SEG.WND. The value 'shift.cnt'

MAY be zero (offering to scale, while applying a scale factor of 1 to the receive window).

This option MAY be sent in an initial <SYN> segment (i.e., a segment with the SYN bit on and the ACK bit off). It MAY also be sent in a <SYN,ACK> segment, but only if a Window Scale option was received in the initial <SYN> segment. A Window Scale option in a segment without a SYN bit MUST be ignored.

The window field in a segment where the SYN bit is set (i.e., a <SYN> or <SYN,ACK>) is never scaled.

### 2.3. Using the Window Scale Option

A model implementation of window scaling is as follows, using the notation of [RFC0793]:

- o All windows are treated as 32-bit quantities for storage in the connection control block and for local calculations. This includes the send-window (SND.WND) and the receive-window (RCV.WND) values, as well as the congestion window.
- o The connection state is augmented by two window shift counters, Snd.Wind.Shift and Rcv.Wind.Shift, to be applied to the incoming and outgoing window fields, respectively.
- o If a TCP receives a <SYN> segment containing a Window Scale option, it sends its own Window Scale option in the <SYN,ACK> segment.
- o The Window Scale option is sent with shift.cnt = R, where R is the value that the TCP would like to use for its receive window.
- o Upon receiving a <SYN> segment with a Window Scale option containing shift.cnt = S, a TCP sets Snd.Wind.Shift to S and sets Rcv.Wind.Shift to R; otherwise, it sets both Snd.Wind.Shift and Rcv.Wind.Shift to zero.
- o The window field (SEG.WND) in the header of every incoming segment, with the exception of <SYN> segments, is left-shifted by Snd.Wind.Shift bits before updating SND.WND:

$$\text{SND.WND} = \text{SEG.WND} \ll \text{Snd.Wind.Shift}$$

(assuming the other conditions of [RFC0793] are met, and using the "C" notation "<<" for left-shift).

- o The window field (SEG.WND) of every outgoing segment, with the exception of <SYN> segments, is right-shifted by Rcv.Wind.Shift bits:

$$\text{SEG.WND} = \text{RCV.WND} \gg \text{Rcv.Wind.Shift}$$

TCP determines if a data segment is "old" or "new" by testing whether its sequence number is within  $2^{31}$  bytes of the left edge of the window, and if it is not, discarding the data as "old". To insure that new data is never mistakenly considered old and vice versa, the left edge of the sender's window has to be at most  $2^{31}$  away from the right edge of the receiver's window. Similarly with the sender's right edge and receiver's left edge. Since the right and left edges of either the sender's or receiver's window differ by the window size, and since the sender and receiver windows can be out of phase by at most the window size, the above constraints imply that two times the maximum window size must be less than  $2^{31}$ , or

$$\text{max window} < 2^{30}$$

Since the max window is  $2^S$  (where  $S$  is the scaling shift count) times at most  $2^{16} - 1$  (the maximum unscaled window), the maximum window is guaranteed to be  $< 2^{30}$  if  $S \leq 14$ . Thus, the shift count MUST be limited to 14 (which allows windows of  $2^{30} = 1 \text{ GiB}$ ). If a Window Scale option is received with a shift.cnt value larger than 14, the TCP SHOULD log the error but MUST use 14 instead of the specified value. This is safe as a sender can always choose to only partially use any signaled receive window.

The scale factor applies only to the Window field as transmitted in the TCP header; each TCP using extended windows will maintain the window values locally as 32-bit numbers. For example, the "congestion window" computed by Slow Start and Congestion Avoidance (see [RFC5681]) is not affected by the scale factor, so window scaling will not introduce quantization into the congestion window.

#### 2.4. Addressing Window Retraction

When a non-zero scale factor is in use, there are instances when a retracted window can be offered - see Appendix F for a detailed example. The end of the window will be on a boundary based on the granularity of the scale factor being used. If the sequence number is then updated by a number of bytes smaller than that granularity, the TCP will have to either advertise a new window that is beyond what it previously advertised (and perhaps beyond the buffer), or will have to advertise a smaller window, which will cause the TCP window to shrink. Implementations MUST ensure that they handle a shrinking window, as specified in section 4.2.2.16 of [RFC1122].

For the receiver, this implies that:

- 1) The receiver MUST honor, as in-window, any segment that would have been in-window for any <ACK> sent by the receiver.
- 2) When window scaling is in effect, the receiver SHOULD track the actual maximum window sequence number (which is likely to be greater than the window announced by the most recent <ACK>, if more than one segment has arrived since the application consumed any data in the receive buffer).

On the sender side:

- 3) The initial transmission MUST be within the window announced by the most recent <ACK>.
- 4) On first retransmission, or if the sequence number is out-of-window by less than  $2^{\text{Rcv.Wind.Shift}}$  then do normal retransmission(s) without regard to receiver window as long as the original segment was in window when it was sent.
- 5) Subsequent retransmissions MAY only be sent, if they are within the window announced by the most recent <ACK>.

### 3. TCP Timestamps option

#### 3.1. Introduction

TCP measures the round trip time (RTT), primarily for the purpose of arriving at a reasonable value for the Retransmission Timeout (RTO) timer interval. Accurate and current RTT estimates are necessary to adapt to changing traffic conditions, while a conservative estimate of the RTO interval is necessary to minimize spurious RTOs.

When [RFC1323] was originally written, it was perceived that taking RTT measurements for each segment, and also during retransmissions, would contribute to reduce spurious RTOs, while maintaining the timeliness of necessary RTOs. At the time, RTO was also the only mechanism to make use of the measured RTT. It has been shown, that taking more RTT samples has only a very limited effect to optimize RTOs [Allman99].

This document makes a clear distinction between the round trip time measurement (RTTM) mechanism, and subsequent mechanisms using the RTT signal as input, such as RTO (see Section 3.4).

The Timestamps option is important when large receive windows are used, to allow the use of the PAWS mechanism (see Section 4). Furthermore, the option is useful for all TCP's, since it simplifies the sender and allows the use of additional optimizations such as Eifel ([RFC3522], [RFC4015]) and others.

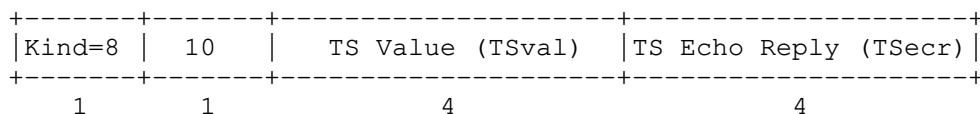
#### 3.2. Timestamps option

TCP is a symmetric protocol, allowing data to be sent at any time in either direction, and therefore timestamp echoing may occur in either direction. For simplicity and symmetry, we specify that timestamps always be sent and echoed in both directions. For efficiency, we combine the timestamp and timestamp reply fields into a single TCP Timestamps option.

TCP Timestamps option (TSopt):

Kind: 8

Length: 10 bytes





The Timestamps option carries two four-byte timestamp fields. The Timestamp Value field (TSval) contains the current value of the timestamp clock of the TCP sending the option.

The Timestamp Echo Reply field (TSecr) is valid if the ACK bit is set in the TCP header; if it is valid, it echoes a timestamp value that was sent by the remote TCP in the TSval field of a Timestamps option. When TSecr is not valid, its value MUST be zero. However, a value of zero does not imply TSecr being invalid. The TSecr value will generally be from the most recent Timestamps option that was received; however, there are exceptions that are explained below.

A TCP MAY send the Timestamps option (TSopt) in an initial <SYN> segment (i.e., segment containing a SYN bit and no ACK bit), and MAY send a TSopt in other segments only if it received a TSopt in the initial <SYN> or <SYN,ACK> segment for the connection.

Once TSopt has been successfully negotiated (sent and received) during the <SYN>, <SYN,ACK> exchange, TSopt MUST be sent in every non-<RST> segment for the duration of the connection, and SHOULD be sent in an <RST> segment (see Section 4.2 for details). If a non-<RST> segment is received without a TSopt, a TCP MUST drop the segment and MAY also send an <ACK> for the last in-sequence segment. A TCP MUST NOT abort a TCP connection because any segment lacks an expected TSopt.

If a TSopt is received on a connection where TSopt was not negotiated in the initial three-way handshake, the TSopt MUST be ignored and the packet processed normally.

In the case of crossing <SYN> segments where one <SYN> contains a TSopt and the other doesn't, both sides MAY send a TSopt in the <SYN,ACK> segment.

TSopt is required for the two mechanisms described in sections 3.3 and 4.2. There are also other mechanisms that rely on the presence of the TSopt, e.g. [RFC3522]. If a TCP stopped sending TSopt at any time during an established session, it interferes with these mechanisms. This update to [RFC1323] describes explicitly the previous assumption (see Section 4.2), that each TCP segment must have TSopt, once negotiated.

### 3.3. The RTTM Mechanism

RTTM places a Timestamps option in every segment, with a TSval that is obtained from a (virtual) "timestamp clock". Values of this clock MUST be at least approximately proportional to real time, in order to measure actual RTT.

These TSval values are echoed in TSecr values in the reverse direction. The difference between a received TSecr value and the current timestamp clock value provides an RTTmeasurement.

When timestamps are used, every segment that is received will contain a TSecr value. However, these values cannot all be used to update the measured RTT. The following example illustrates why. It shows a one-way data flow with segments arriving in sequence without loss. Here A, B, C... represent data blocks occupying successive blocks of sequence numbers, and ACK(A),... represent the corresponding cumulative acknowledgments. The two timestamp fields of the Timestamps option are shown symbolically as <TSval=x,TSecr=y>. Each TSecr field contains the value most recently received in a TSval field.

```

TCP A                                     TCP B

                                <A,TSval=1,TSecr=120> ----->

                                <----- <ACK(A),TSval=127,TSecr=1>

                                <B,TSval=5,TSecr=127> ----->

                                <----- <ACK(B),TSval=131,TSecr=5>

                                . . . . .

                                <C,TSval=65,TSecr=131> ----->

                                <----- <ACK(C),TSval=191,TSecr=65>

                                (etc.)

```

The dotted line marks a pause (60 time units long) in which A had nothing to send. Note that this pause inflates the RTT which B could infer from receiving TSecr=131 in data segment C. Thus, in one-way data flows, RTTM in the reverse direction measures a value that is inflated by gaps in sending data. However, the following rule prevents a resulting inflation of the measured RTT:

RTTM Rule: A TSecr value received in a segment MAY be used to update the averaged RTT measurement only if the segment advances the left edge of the send window, i.e. SND.UNA is increased.

Since TCP B is not sending data, the data segment C does not acknowledge any new data when it arrives at B. Thus, the inflated RTTM measurement is not used to update B's RTTM measurement.

### 3.4. Updating the RTO value

[Ludwig00] and [Floyd05] have highlighted the problem that an unmodified RTO calculation, which is updated with per-packet RTT samples, will truncate the path history too soon. This can lead to an increase in spurious retransmissions, when the path properties vary in the order of a few RTTs, but a high number of RTT samples are taken on a much shorter timescale.

Implementers should note that with timestamps multiple RTTMs can be taken per RTT. The [RFC6298] RTO estimator has weighting factors, alpha and beta, based on an implicit assumption that at most one RTT will be sampled per RTT. When multiple RTTMs per RTT are available to update the RTO estimator, this implicit assumption must be considered. An implementation suggestion is detailed in Appendix G.

### 3.5. Which Timestamp to Echo

If more than one Timestamps option is received before a reply segment is sent, the TCP must choose only one of the TSvals to echo, ignoring the others. To minimize the state kept in the receiver (i.e., the number of unprocessed TSvals), the receiver should be required to retain at most one timestamp in the connection control block.

There are three situations to consider:

#### (A) Delayed ACKs.

Many TCP's acknowledge only every second segment out of a group of segments arriving within a short time interval; this policy is known generally as "delayed ACKs". The data-sender TCP must measure the effective RTT, including the additional time due to delayed ACKs, or else it will retransmit unnecessarily. Thus, when delayed ACKs are in use, the receiver SHOULD reply with the TSval field from the earliest unacknowledged segment.

#### (B) A hole in the sequence space (segment(s) have been lost).

The sender will continue sending until the window is filled, and the receiver may be generating <ACK>s as these out-of-order segments arrive (e.g., to aid "fast retransmit").

The lost segment is probably a sign of congestion, and in that situation the sender should be conservative about retransmission. Furthermore, it is better to overestimate than underestimate the RTT. An <ACK> for an out-of-order segment SHOULD therefore contain the timestamp from the most recent segment that advanced the window.

The same situation occurs if segments are re-ordered by the network.

- (C) A filled hole in the sequence space.

The segment that fills the hole and advances the window represents the most recent measurement of the network characteristics. An RTT computed from an earlier segment would probably include the sender's retransmit time-out, badly biasing the sender's average RTT estimate. Thus, the timestamp from the latest segment (which filled the hole) MUST be echoed.

An algorithm that covers all three cases is described in the following rules for Timestamps option processing on a synchronized connection:

- (1) The connection state is augmented with two 32-bit slots:

TS.Recent holds a timestamp to be echoed in TSecr whenever a segment is sent, and Last.ACK.sent holds the ACK field from the last segment sent. Last.ACK.sent will equal RCV.NXT except when <ACK>s have been delayed.

- (2) If:

SEG.TSval >= TS.recent and SEG.SEQ <= Last.ACK.sent

then SEG.TSval is copied to TS.Recent; otherwise, it is ignored.

- (3) When a TSopt is sent, its TSecr field is set to the current TS.Recent value.

The following examples illustrate these rules. Here A, B, C... represent data segments occupying successive blocks of sequence numbers, and ACK(A),... represent the corresponding acknowledgment segments. Note that ACK(A) has the same sequence number as B. We show only one direction of timestamp echoing, for clarity.

- o Segments arrive in sequence, and some of the <ACK>s are delayed.

By case (A), the timestamp from the oldest unacknowledged segment is echoed.

	TS.Recent
<A, TSval=1> ----->	1
<B, TSval=2> ----->	1
<C, TSval=3> ----->	1
<---- <ACK(C), TSecr=1>	
(etc)	

- o Segments arrive out of order, and every segment is acknowledged.

By case (B), the timestamp from the last segment that advanced the left window edge is echoed, until the missing segment arrives; it is echoed according to Case (C). The same sequence would occur if segments B and D were lost and retransmitted.

	TS.Recent
<A, TSval=1> ----->	1
<---- <ACK(A), TSecr=1>	1
<C, TSval=3> ----->	1
<---- <ACK(A), TSecr=1>	1
<B, TSval=2> ----->	2
<---- <ACK(C), TSecr=2>	2
<E, TSval=5> ----->	2
<---- <ACK(C), TSecr=2>	2
<D, TSval=4> ----->	4
<---- <ACK(E), TSecr=4>	
(etc)	

## 4. PAWS - Protection Against Wrapped Sequence Numbers

### 4.1. Introduction

Section 4.2 describes a simple mechanism to reject old duplicate segments that might corrupt an open TCP connection; we call this mechanism PAWS (Protection Against Wrapped Sequence numbers). PAWS operates within a single TCP connection, using state that is saved in the connection control block. Section 4.8 and Appendix H discuss the implications of the PAWS mechanism for avoiding old duplicates from previous incarnations of the same connection.

### 4.2. The PAWS Mechanism

PAWS uses the same TCP Timestamps option as the RTTM mechanism described earlier, and assumes that every received TCP segment (including data and <ACK> segments) contains a timestamp `SEG.TSval` whose values are monotonically non-decreasing in time. The basic idea is that a segment can be discarded as an old duplicate if it is received with a timestamp `SEG.TSval` less than some timestamp recently received on this connection.

In both the PAWS and the RTTM mechanism, the "timestamps" are 32-bit unsigned integers in a modular 32-bit space. Thus, "less than" is defined the same way it is for TCP sequence numbers, and the same implementation techniques apply. If  $s$  and  $t$  are timestamp values,

$$s < t \quad \text{if } 0 < (t - s) < 2^{31},$$

computed in unsigned 32-bit arithmetic.

The choice of incoming timestamps to be saved for this comparison MUST guarantee a value that is monotonically increasing. For example, we might save the timestamp from the segment that last advanced the left edge of the receive window, i.e., the most recent in-sequence segment. Instead, we choose the value `TS.Recent` introduced in Section 3.5 for the RTTM mechanism, since using a common value for both PAWS and RTTM simplifies the implementation of both. As Section 3.5 explained, `TS.Recent` differs from the timestamp from the last in-sequence segment only in the case of delayed <ACK>s, and therefore by less than one window. Either choice will therefore protect against sequence number wrap-around.

RTTM was specified in a symmetrical manner, so that `TSval` timestamps are carried in both data and <ACK> segments and are echoed in `TSseq` fields carried in returning <ACK> or data segments. PAWS submits all incoming segments to the same test, and therefore protects against duplicate <ACK> segments as well as data segments. (An alternative

non-symmetric algorithm would protect against old duplicate <ACK>s: the sender of data would reject incoming <ACK> segments whose TSecr values were less than the TSecr saved from the last segment whose ACK field advanced the left edge of the send window. This algorithm was deemed to lack economy of mechanism and symmetry.)

TSval timestamps sent on <SYN> and <SYN,ACK> segments are used to initialize PAWS. PAWS protects against old duplicate non-<SYN> segments, and duplicate <SYN> segments received while there is a synchronized connection. Duplicate <SYN> and <SYN,ACK> segments received when there is no connection will be discarded by the normal 3-way handshake and sequence number checks of TCP.

[RFC1323] recommended that <RST> segments NOT carry timestamps, and that they be acceptable regardless of their timestamp. At that time, the thinking was that old duplicate <RST> segments should be exceedingly unlikely, and their cleanup function should take precedence over timestamps. More recently, discussions about various blind attacks on TCP connections have raised the suggestion that if the Timestamps option is present, SEG.TSecr could be used to provide stricter acceptance tests for <RST> segments. While still under discussion, to enable research into this area it is now RECOMMENDED that when generating an <RST>, that if the segment causing the <RST> to be generated contained a Timestamps option, that the <RST> also contain a Timestamps option. In the <RST> segment, SEG.TSecr SHOULD be set to SEG.TSval from the incoming segment and SEG.TSval SHOULD be set to zero. If an <RST> is being generated because of a user abort, and Snd.TS.OK is set, then a Timestamps option SHOULD be included in the <RST>. When an <RST> segment is received, it MUST NOT be subjected to PAWS checks, and information from the Timestamps option MUST NOT be used to update connection state information. SEG.TSecr MAY be used to provide stricter <RST> acceptance checks.

#### 4.3. Basic PAWS Algorithm

The PAWS algorithm REQUIRES the following processing to be performed on all incoming segments for a synchronized connection. Also, PAWS processing MUST take precedence over the regular TCP acceptability check (Section 3.3 in [RFC0793]), which is performed after verification of the received Timestamps option:

- R1) If there is a Timestamps option in the arriving segment, SEG.TSval < TS.Recent, TS.Recent is valid (see later discussion) and the RST bit is not set, then treat the arriving segment as not acceptable:

Send an acknowledgement in reply as specified in [RFC0793] page 69 and drop the segment.

Note: it is necessary to send an <ACK> segment in order to retain TCP's mechanisms for detecting and recovering from half-open connections. For example, see Figure 10 of [RFC0793].

- R2) If the segment is outside the window, reject it (normal TCP processing)
- R3) If an arriving segment satisfies: `SEG.SEQ <= Last.ACK.sent` (see Section 3.5), then record its timestamp in `TS.Recent`.
- R4) If an arriving segment is in-sequence (i.e., at the left window edge), then accept it normally.
- R5) Otherwise, treat the segment as a normal in-window, out-of-sequence TCP segment (e.g., queue it for later delivery to the user).

Steps R2, R4, and R5 are the normal TCP processing steps specified by [RFC0793].

It is important to note that the timestamp **MUST** be checked only when a segment first arrives at the receiver, regardless of whether it is in-sequence or it must be queued for later delivery.

Consider the following example.

Suppose the segment sequence: A.1, B.1, C.1, ..., Z.1 has been sent, where the letter indicates the sequence number and the digit represents the timestamp. Suppose also that segment B.1 has been lost. The timestamp in `TS.Recent` is 1 (from A.1), so C.1, ..., Z.1 are considered acceptable and are queued. When B is retransmitted as segment B.2 (using the latest timestamp), it fills the hole and causes all the segments through Z to be acknowledged and passed to the user. The timestamps of the queued segments are *\*not\** inspected again at this time, since they have already been accepted. When B.2 is accepted, `TS.Recent` is set to 2.

This rule allows reasonable performance under loss. A full window of data is in transit at all times, and after a loss a full window less one segment will show up out-of-sequence to be queued at the receiver (e.g., up to  $2^{30}$  bytes of data); the Timestamps option must not result in discarding this data.

In certain unlikely circumstances, the algorithm of rules R1-R5 could lead to discarding some segments unnecessarily, as shown in the following example:



Suppose again that segments: A.1, B.1, C.1, ..., Z.1 have been sent in sequence and that segment B.1 has been lost. Furthermore, suppose delivery of some of C.1, ... Z.1 is delayed until *after* the retransmission B.2 arrives at the receiver. These delayed segments will be discarded unnecessarily when they do arrive, since their timestamps are now out of date.

This case is very unlikely to occur. If the retransmission was triggered by a timeout, some of the segments C.1, ... Z.1 must have been delayed longer than the RTO time. This is presumably an unlikely event, or there would be many spurious timeouts and retransmissions. If B's retransmission was triggered by the "fast retransmit" algorithm, i.e., by duplicate <ACK>s, then the queued segments that caused these <ACK>s must have been received already.

Even if a segment were delayed past the RTO, the Fast Retransmit mechanism [Jacobson90c] will cause the delayed segments to be retransmitted at the same time as B.2, avoiding an extra RTT and therefore causing a very small performance penalty.

We know of no case with a significant probability of occurrence in which timestamps will cause performance degradation by unnecessarily discarding segments.

#### 4.4. Timestamp Clock

It is important to understand that the PAWS algorithm does not require clock synchronization between sender and receiver. The sender's timestamp clock is used to stamp the segments, and the sender uses the echoed timestamp to measure RTTs. However, the receiver treats the timestamp as simply a monotonically increasing serial number, without any necessary connection to its clock. From the receiver's viewpoint, the timestamp is acting as a logical extension of the high-order bits of the sequence number.

The receiver algorithm does place some requirements on the frequency of the timestamp clock.

(a) The timestamp clock must not be "too slow".

It MUST tick at least once for each  $2^{31}$  bytes sent. In fact, in order to be useful to the sender for round trip timing, the clock SHOULD tick at least once per window's worth of data, and even with the window extension defined in Section 2.2,  $2^{31}$  bytes must be at least two windows.

To make this more quantitative, any clock faster than 1 tick/sec will reject old duplicate segments for link speeds of ~8 Gbps.

A 1 ms timestamp clock will work at link speeds up to 8 Tbps ( $8 \times 10^{12}$ ) bps!

- (b) The timestamp clock must not be "too fast".

The recycling time of the timestamp clock MUST be greater than MSL seconds. Since the clock (timestamp) is 32 bits and the worst-case MSL is 255 seconds, the maximum acceptable clock frequency is one tick every 59 ns.

However, it is desirable to establish a much longer recycle period, in order to handle outdated timestamps on idle connections (see Section 4.5), and to relax the MSL requirement for preventing sequence number wrap-around. With a 1 ms timestamp clock, the 32-bit timestamp will wrap its sign bit in 24.8 days. Thus, it will reject old duplicates on the same connection if MSL is 24.8 days or less. This appears to be a very safe figure; an MSL of 24.8 days or longer can probably be assumed in the internet without requiring precise MSL enforcement.

Based upon these considerations, we choose a timestamp clock frequency in the range 1 ms to 1 sec per tick. This range also matches the requirements of the RTTM mechanism, which does not need much more resolution than the granularity of the retransmit timer, e.g., tens or hundreds of milliseconds.

The PAWS mechanism also puts a strong monotonicity requirement on the sender's timestamp clock. The method of implementation of the timestamp clock to meet this requirement depends upon the system hardware and software.

- o Some hosts have a hardware clock that is guaranteed to be monotonic between hardware resets.
- o A clock interrupt may be used to simply increment a binary integer by 1 periodically.
- o The timestamp clock may be derived from a system clock that is subject to being abruptly changed, by adding a variable offset value. This offset is initialized to zero. When a new timestamp clock value is needed, the offset can be adjusted as necessary to make the new value equal to or larger than the previous value (which was saved for this purpose).

#### 4.5. Outdated Timestamps

If a connection remains idle long enough for the timestamp clock of the other TCP to wrap its sign bit, then the value saved in TS.Recent will become too old; as a result, the PAWS mechanism will cause all subsequent segments to be rejected, freezing the connection (until the timestamp clock wraps its sign bit again).

With the chosen range of timestamp clock frequencies (1 sec to 1 ms), the time to wrap the sign bit will be between 24.8 days and 24800 days. A TCP connection that is idle for more than 24 days and then comes to life is exceedingly unusual. However, it is undesirable in principle to place any limitation on TCP connection lifetimes.

We therefore require that an implementation of PAWS include a mechanism to "invalidate" the TS.Recent value when a connection is idle for more than 24 days. (An alternative solution to the problem of outdated timestamps would be to send keep-alive segments at a very low rate, but still more often than the wrap-around time for timestamps, e.g., once a day. This would impose negligible overhead. However, the TCP specification has never included keep-alives, so the solution based upon invalidation was chosen.)

Note that a TCP does not know the frequency, and therefore, the wraparound time, of the other TCP, so it must assume the worst. The validity of TS.Recent needs to be checked only if the basic PAWS timestamp check fails, i.e., only if  $SEG.TSval < TS.Recent$ . If TS.Recent is found to be invalid, then the segment is accepted, regardless of the failure of the timestamp check, and rule R3 updates TS.Recent with the TSval from the new segment.

To detect how long the connection has been idle, the TCP MAY update a clock or timestamp value associated with the connection whenever TS.Recent is updated, for example. The details will be implementation-dependent.

#### 4.6. Header Prediction

"Header prediction" [Jacobson90a] is a high-performance transport protocol implementation technique that is most important for high-speed links. This technique optimizes the code for the most common case, receiving a segment correctly and in order. Using header prediction, the receiver asks the question, "Is this segment the next in sequence?" This question can be answered in fewer machine instructions than the question, "Is this segment within the window?"

Adding header prediction to our timestamp procedure leads to the following recommended sequence for processing an arriving TCP

segment:

- H1) Check timestamp (same as step R1 above)
- H2) Do header prediction: if segment is next in sequence and if there are no special conditions requiring additional processing, accept the segment, record its timestamp, and skip H3.
- H3) Process the segment normally, as specified in RFC 793. This includes dropping segments that are outside the window and possibly sending acknowledgments, and queuing in-window, out-of-sequence segments.

Another possibility would be to interchange steps H1 and H2, i.e., to perform the header prediction step H2 \*first\*, and perform H1 and H3 only when header prediction fails. This could be a performance improvement, since the timestamp check in step H1 is very unlikely to fail, and it requires unsigned modulo arithmetic. To perform this check on every single segment is contrary to the philosophy of header prediction. We believe that this change might produce a measurable reduction in CPU time for TCP protocol processing on high-speed networks.

However, putting H2 first would create a hazard: a segment from  $2^{32}$  bytes in the past might arrive at exactly the wrong time and be accepted mistakenly by the header-prediction step. The following reasoning has been introduced in [RFC1185] to show that the probability of this failure is negligible.

If all segments are equally likely to show up as old duplicates, then the probability of an old duplicate exactly matching the left window edge is the maximum segment size (MSS) divided by the size of the sequence space. This ratio must be less than  $2^{-16}$ , since MSS must be  $< 2^{16}$ ; for example, it will be  $(2^{12})/(2^{32}) = 2^{-20}$  for a 100 Mbit/s link. However, the older a segment is, the less likely it is to be retained in the Internet, and under any reasonable model of segment lifetime the probability of an old duplicate exactly at the left window edge must be much smaller than  $2^{-16}$ .

The 16 bit TCP checksum also allows a basic unreliability of one part in  $2^{16}$ . A protocol mechanism whose reliability exceeds the reliability of the TCP checksum should be considered "good enough", i.e., it won't contribute significantly to the overall error rate. We therefore believe we can ignore the problem of an old duplicate being accepted by doing header prediction before checking the timestamp.

However, this probabilistic argument is not universally accepted, and the consensus at present is that the performance gain does not justify the hazard in the general case. It is therefore recommended that H2 follow H1.

#### 4.7. IP Fragmentation

At high data rates, the protection against old segments provided by PAWS can be circumvented by errors in IP fragment reassembly (see [RFC4963]). The only way to protect against incorrect IP fragment reassembly is to not allow the segments to be fragmented. This is done by setting the Don't Fragment (DF) bit in the IP header. Setting the DF bit implies the use of Path MTU Discovery as described in [RFC1191], [RFC1981], and [RFC4821], thus any TCP implementation that implements PAWS MUST also implement Path MTU Discovery.

#### 4.8. Duplicates from Earlier Incarnations of Connection

The PAWS mechanism protects against errors due to sequence number wrap-around on high-speed connections. Segments from an earlier incarnation of the same connection are also a potential cause of old duplicate errors. In both cases, the TCP mechanisms to prevent such errors depend upon the enforcement of a maximum segment lifetime (MSL) by the Internet (IP) layer (see Appendix of RFC 1185 for a detailed discussion). Unlike the case of sequence space wrap-around, the MSL required to prevent old duplicate errors from earlier incarnations does not depend upon the transfer rate. If the IP layer enforces the recommended 2 minute MSL of TCP, and if the TCP rules are followed, TCP connections will be safe from earlier incarnations, no matter how high the network speed. Thus, the PAWS mechanism is not required for this case.

We may still ask whether the PAWS mechanism can provide additional security against old duplicates from earlier connections, allowing us to relax the enforcement of MSL by the IP layer. Appendix B explores this question, showing that further assumptions and/or mechanisms are required, beyond those of PAWS. This is not part of the current extension.

### 5. Conclusions and Acknowledgements

This memo presented a set of extensions to TCP to provide efficient operation over large bandwidth \* delay product paths and reliable operation over very high-speed paths. These extensions are designed to provide compatible interworking with TCP stacks that do not implement the extensions.

These mechanisms are implemented using TCP options for scaled windows and timestamps. The timestamps are used for two distinct mechanisms: RTTM (Round Trip Time Measurement) and PAWS (Protection Against Wrapped Sequences).

The Window Scale option was originally suggested by Mike St. Johns of USAF/DCA. The present form of the option was suggested by Mike Karels of UC Berkeley in response to a more cumbersome scheme defined by Van Jacobson. Lixia Zhang helped formulate the PAWS mechanism description in [RFC1185].

Finally, much of this work originated as the result of discussions within the End-to-End Task Force on the theoretical limitations of transport protocols in general and TCP in particular. Task force members and other on the end2end-interest list have made valuable contributions by pointing out flaws in the algorithms and the documentation. Continued discussion and development since the publication of [RFC1323] originally occurred in the IETF TCP Large Windows Working Group, later on in the End-to-End Task Force, and most recently in the IETF TCP Maintenance Working Group. The authors are grateful for all these contributions.

## 6. Security Considerations

The TCP sequence space is a fixed size, and as the window becomes larger it becomes easier for an attacker to generate forged packets that can fall within the TCP window, and be accepted as valid segments. While use of timestamps and PAWS can help to mitigate this, when using PAWS, if an attacker is able to forge a packet that is acceptable to the TCP connection, a timestamp that is in the future would cause valid segments to be dropped due to PAWS checks. Hence, implementers should take care to not open the TCP window drastically beyond the requirements of the connection.

A naive implementation that derives the timestamp clock value directly from a system uptime clock may unintentionally leak this information to an attacker. This does not directly compromise any of the mechanisms described in this document. However, this may be valuable information to a potential attacker. An implementer should evaluate the potential impact and mitigate this accordingly (i.e. by using a random offset for the timestamp clock on each connection, or using an external, real-time derived timestamp clock source).

Expanding the TCP window beyond 64 KiB for IPv6 allows Jumbograms [RFC2675] to be used when the local network supports packets larger than 64 KiB. When larger TCP segments are used, the TCP checksum becomes weaker.

Mechanisms to protect the TCP header from modification should also protect the TCP options.

Middleboxes and TCP options:

Some middleboxes have been known to remove the TCP options described in this document from the <SYN> segment. Middleboxes that remove TCP options described in this document from the <SYN> segment interfere with the selection of parameters appropriate for the session. Removing any of these options in a <SYN,ACK> segment will leave the end hosts in a state that destroys the proper operation of the protocol.

- \* If a Window Scale option is removed from a <SYN,ACK> segment, the end hosts will not negotiate the window scaling factor correctly. Middleboxes must not remove or modify the Window Scale option from <SYN,ACK> segments.
- \* If a stateful firewall uses the window field to detect whether a received segment is inside the current window, and does not support the Window Scale option, it will not be able to correctly determine whether or not a packet is in the window. These middle boxes must also support the Window Scale option and apply the scale factor when processing segments. If the window scale factor cannot be determined, it must not do window based processing.
- \* If the Timestamps option is removed from the <SYN> or <SYN,ACK> segment, high speed connections that need PAWS would not have that protection. Middleboxes should not remove the Timestamps option.

Implementations that depend on PAWS could provide a mechanism for the application to determine whether or not PAWS is in use on the connection, and chose to terminate the connection if that protection doesn't exist. This is not just to protect the connection against middleboxes that might remove the Timestamps option, but also against remote hosts that do not have Timestamp support.

## 7. IANA Considerations

This document has no actions for IANA.

## 8. References

### 8.1. Normative References

- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
- [RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, November 1990.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

### 8.2. Informative References

- [Allman99] Allman, M. and V. Paxson, "On Estimating End-to-End Network Path Properties", Proc. ACM SIGCOMM Technical Symposium, Cambridge, MA, September 1999, <<http://aciri.org/mallman/papers/estimation-la.pdf>>.
- [Ekstroem04] Ekstroem, H. and R. Ludwig, "The Peak-Hopper: A New End-to-End Retransmission Timer for Reliable Unicast Transport", INFOCOM 2004 IEEE, March 2004, <<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.76.2748&rep=rep1&type=pdf>>.
- [Floyd05] Floyd, S., "[tcpm] How the RTO should be estimated with timestamps", Message from 26.Jan.2007 to the tcpm mailing list, August 2005, <<http://www.ietf.org/mail-archive/web/tcpmp/current/msg02508.html>>.
- [Garlick77] Garlick, L., Rom, R., and J. Postel, "Issues in Reliable Host-to-Host Protocols", Proc. Second Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977, <<http://www.rfc-editor.org/ien/ien12.txt>>.
- [Hamming77] Hamming, R., "Digital Filters", Prentice Hall, Englewood Cliffs, N.J. ISBN 0-13-212571-4, 1977.
- [Jacobson88a] Jacobson, V., "Congestion Avoidance and Control", SIGCOMM '88, Stanford, CA., August 1988, <<http://ee.lbl.gov/papers/congavoid.pdf>>.



- [Jacobson90a] Jacobson, V., "4BSD Header Prediction", ACM Computer Communication Review, April 1990.
- [Jacobson90c] Jacobson, V., "Modified TCP congestion avoidance algorithm", Message to the end2end-interest mailing list, April 1990, <<ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail>>.
- [Jain86] Jain, R., "Divergence of Timeout Algorithms for Packet Retransmissions", Proc. Fifth Phoenix Conf. on Comp. and Comm., Scottsdale, Arizona, March 1986, <<http://arxiv.org/ftp/cs/papers/9809/9809097.pdf>>.
- [Karn87] Karn, P. and C. Partridge, "Estimating Round-Trip Times in Reliable Transport Protocols", Proc. SIGCOMM '87, August 1987.
- [Ludwig00] Ludwig, R. and K. Sklower, "The Eifel Retransmission Timer", ACM SIGCOMM Computer Communication Review Volume 30 Issue 3, July 2000, <<http://ccr.sigcomm.org/archive/2000/july00/LudwigFinal.pdf>>.
- [Martin03] Martin, D., "[Tsvwg] RFC 1323.bis", Message to the tsvwg mailing list, September 2003, <<http://www.ietf.org/mail-archive/web/tsvwg/current/msg04435.html>>.
- [Mathis08] Mathis, M., "[tcpm] Example of 1323 window retraction problem", Message to the tcpm mailing list, March 2008, <<http://www.ietf.org/mail-archive/web/tcpm/current/msg03564.html>>.
- [Medina04] Medina, A., Allman, M., and S. Floyd, "Measuring Interactions Between Transport Protocols and Middleboxes", Proc. ACM SIGCOMM/USENIX Internet Measurement Conference, October 2004, August 2004, <<http://www.icir.net/tbit/tbit-Aug2004.pdf>>.
- [Medina05] Medina, A., Allman, M., and S. Floyd, "Measuring the Evolution of Transport Protocols in the Internet", ACM Computer Communication Review 35(2), April 2005, <<http://icir.net/floyd/papers/TCPEvolution-Mar2005.pdf>>.

- [RFC0896] Nagle, J., "Congestion control in IP/TCP internetworks", RFC 896, January 1984.
- [RFC1072] Jacobson, V. and R. Braden, "TCP extensions for long-delay paths", RFC 1072, October 1988.
- [RFC1110] McKenzie, A., "Problem with the TCP big window option", RFC 1110, August 1989.
- [RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.
- [RFC1185] Jacobson, V., Braden, B., and L. Zhang, "TCP Extension for High-Speed Paths", RFC 1185, October 1990.
- [RFC1323] Jacobson, V., Braden, B., and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.
- [RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, August 1996.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.
- [RFC2581] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.
- [RFC2675] Borman, D., Deering, S., and R. Hinden, "IPv6 Jumbograms", RFC 2675, August 1999.
- [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.
- [RFC4015] Ludwig, R. and A. Gurtov, "The Eifel Response Algorithm for TCP", RFC 4015, February 2005.
- [RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.
- [RFC4963] Heffner, J., Mathis, M., and B. Chandler, "IPv4 Reassembly Errors at High Data Rates", RFC 4963, July 2007.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.

- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, 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.
- [RFC6691] Borman, D., "TCP Options and Maximum Segment Size (MSS)", RFC 6691, July 2012.
- [Watson81] Watson, R., "Timer-based Mechanisms in Reliable Transport Protocol Connection Management", Computer Networks, Vol. 5, 1981.
- [Zhang86] Zhang, L., "Why TCP Timers Don't Work Well", Proc. SIGCOMM '86, Stowe, VT, August 1986.

## Appendix A. Implementation Suggestions

### TCP Option Layout

The following layout is recommended for sending options on non-<SYN> segments, to achieve maximum feasible alignment of 32-bit and 64-bit machines.

+-----+	+-----+	+-----+	+-----+
NOP	NOP	TSopt	10
+-----+	+-----+	+-----+	+-----+
	TSval timestamp		
+-----+	+-----+	+-----+	+-----+
	TSecr timestamp		
+-----+	+-----+	+-----+	+-----+

### Interaction with the TCP Urgent Pointer

The TCP Urgent pointer, like the TCP window, is a 16 bit value. Some of the original discussion for the TCP Window Scale option included proposals to increase the Urgent pointer to 32 bits. As it turns out, this is unnecessary. There are two observations that should be made:

- (1) With IP Version 4, the largest amount of TCP data that can be sent in a single packet is 65495 bytes (64 KiB - 1 -- size of fixed IP and TCP headers).
- (2) Updates to the urgent pointer while the user is in "urgent mode" are invisible to the user.

This means that if the Urgent Pointer points beyond the end of the TCP data in the current segment, then the user will remain in urgent mode until the next TCP segment arrives. That segment will update the urgent pointer to a new offset, and the user will never have left urgent mode.

Thus, to properly implement the Urgent Pointer, the sending TCP only has to check for overflow of the 16 bit Urgent Pointer field before filling it in. If it does overflow, than a value of 65535 should be inserted into the Urgent Pointer.

The same technique applies to IP Version 6, except in the case of IPv6 Jumbograms. When IPv6 Jumbograms are supported, [RFC2675] requires additional steps for dealing with the Urgent Pointer, these are described in section 5.2 of [RFC2675].

## Appendix B. Duplicates from Earlier Connection Incarnations

There are two cases to be considered: (1) a system crashing (and losing connection state) and restarting, and (2) the same connection being closed and reopened without a loss of host state. These will be described in the following two sections.

### B.1. System Crash with Loss of State

TCP's quiet time of one MSL upon system startup handles the loss of connection state in a system crash/restart. For an explanation, see for example "When to Keep Quiet" in the TCP protocol specification [RFC0793]. The MSL that is required here does not depend upon the transfer speed. The current TCP MSL of 2 minutes seemed acceptable as an operational compromise, when many host systems used to take this long to boot after a crash. Current host systems can boot considerably faster.

The Timestamps option may be used to ease the MSL requirements (or to provide additional security against data corruption). If timestamps are being used and if the timestamp clock can be guaranteed to be monotonic over a system crash/restart, i.e., if the first value of the sender's timestamp clock after a crash/restart can be guaranteed to be greater than the last value before the restart, then a quiet

time is unnecessary.

To dispense totally with the quiet time would require that the host clock be synchronized to a time source that is stable over the crash/restart period, with an accuracy of one timestamp clock tick or better. We can back off from this strict requirement to take advantage of approximate clock synchronization. Suppose that the clock is always re-synchronized to within  $N$  timestamp clock ticks and that booting (extended with a quiet time, if necessary) takes more than  $N$  ticks. This will guarantee monotonicity of the timestamps, which can then be used to reject old duplicates even without an enforced MSL.

## B.2. Closing and Reopening a Connection

When a TCP connection is closed, a delay of  $2 \times \text{MSL}$  in TIME-WAIT state ties up the socket pair for 4 minutes (see Section 3.5 of [RFC0793]). Applications built upon TCP that close one connection and open a new one (e.g., an FTP data transfer connection using Stream mode) must choose a new socket pair each time. The TIME-WAIT delay serves two different purposes:

- (a) Implement the full-duplex reliable close handshake of TCP.

The proper time to delay the final close step is not really related to the MSL; it depends instead upon the RTO for the FIN segments and therefore upon the RTT of the path. (It could be argued that the side that is sending a FIN knows what degree of reliability it needs, and therefore it should be able to determine the length of the TIME-WAIT delay for the FIN's recipient. This could be accomplished with an appropriate TCP option in FIN segments.)

Although there is no formal upper-bound on RTT, common network engineering practice makes an RTT greater than 1 minute very unlikely. Thus, the 4 minute delay in TIME-WAIT state works satisfactorily to provide a reliable full-duplex TCP close. Note again that this is independent of MSL enforcement and network speed.

The TIME-WAIT state could cause an indirect performance problem if an application needed to repeatedly close one connection and open another at a very high frequency, since the number of available TCP ports on a host is less than  $2^{16}$ . However, high network speeds are not the major contributor to this problem; the RTT is the limiting factor in how quickly connections can be opened and closed. Therefore, this problem will be no worse at high transfer speeds.

- (b) Allow old duplicate segments to expire.

To replace this function of TIME-WAIT state, a mechanism would have to operate across connections. PAWS is defined strictly within a single connection; the last timestamp (TS.Recent) is kept in the connection control block, and discarded when a connection is closed.

An additional mechanism could be added to the TCP, a per-host cache of the last timestamp received from any connection. This value could then be used in the PAWS mechanism to reject old duplicate segments from earlier incarnations of the connection, if the timestamp clock can be guaranteed to have ticked at least once since the old connection was open. This would require that the TIME-WAIT delay plus the RTT together must be at least one tick of the sender's timestamp clock. Such an extension is not part of the proposal of this RFC.

Note that this is a variant on the mechanism proposed by Garlick, Rom, and Postel [Garlick77], which required each host to maintain connection records containing the highest sequence numbers on every connection. Using timestamps instead, it is only necessary to keep one quantity per remote host, regardless of the number of simultaneous connections to that host.

## Appendix C. Summary of Notation

The following notation has been used in this document.

### Options

WSopt:	TCP Window Scale Option
TSopt:	TCP Timestamps option

### Option Fields

shift.cnt:	Window scale byte in WSopt
TSval:	32-bit Timestamp Value field in TSopt
TSocr:	32-bit Timestamp Reply field in TSopt

### Option Fields in Current Segment

SEG.TSval:	TSval field from TSopt in current segment
------------	---

SEG.TSecr: TSecr field from TSopt in current segment  
 SEG.WSopt: 8-bit value in WSopt

#### Clock Values

my.TSclock: System wide source of 32-bit timestamp values  
 my.TSclock.rate: Period of my.TSclock (1 ms to 1 sec)  
 Snd.TSoffset: A offset for randomizing Snd.TSclock  
 my.TSclock: my.TSclock + Snd.TSoffset

#### Per-Connection State Variables

TS.Recent: Latest received Timestamp  
 Last.ACK.sent: Last ACK field sent  
 Snd.TS.OK: 1-bit flag  
 Snd.WS.OK: 1-bit flag  
 Rcv.Wind.Shift: Receive window scale exponent  
 Snd.Wind.Shift: Send window scale exponent  
 Start.Time: Snd.TSclock value when segment being timed was  
               sent (used by pre-1323 code).

#### Procedure

Update\_SRTT(m) Procedure to update the smoothed RTT and RTT  
                   variance estimates, using the rules of  
                   [Jacobson88a], given m, a new RTT measurement

### Appendix D. Event Processing Summary

#### OPEN Call

...

An initial send sequence number (ISS) is selected. Send a <SYN>  
 segment of the form:

<SEQ=ISS><CTL=SYN><TSval=Snd.TSclock><WSopt=Rcv.Wind.Shift>

...

#### SEND Call

CLOSED STATE (i.e., TCB does not exist)

...

## LISTEN STATE

If the foreign socket is specified, then change the connection from passive to active, select an ISS. Send a <SYN> segment containing the options: <TSval=Snd.TSclock> and <WSopt=Rcv.Wind.Shift>. Set SND.UNA to ISS, SND.NXT to ISS+1. Enter SYN-SENT state. ...

## SYN-SENT STATE

## SYN-RECEIVED STATE

...

## ESTABLISHED STATE

## CLOSE-WAIT STATE

Segmentize the buffer and send it with a piggybacked acknowledgment (acknowledgment value = RCV.NXT). ...

If the urgent flag is set ...

If the Snd.TS.OK flag is set, then include the TCP Timestamps option <TSval=Snd.TSclock,TSecr=TS.Recent> in each data segment.

Scale the receive window for transmission in the segment header:

SEG.WND = (RCV.WND >> Rcv.Wind.Shift).

## SEGMENT ARRIVES

...

If the state is LISTEN then

first check for an RST

...

second check for an ACK

...

third check for a SYN

if the SYN bit is set, check the security. If the ...



...

if the SEG.PRC is less than the TCB.PRC then continue.

Check for a Window Scale option (WSopt); if one is found, save SEG.WSopt in Snd.Wind.Shift and set Snd.WS.OK flag on. Otherwise, set both Snd.Wind.Shift and Rcv.Wind.Shift to zero and clear Snd.WS.OK flag.

Check for a TSopt option; if one is found, save SEG.TSval in the variable TS.Recent and turn on the Snd.TS.OK bit.

Set RCV.NXT to SEG.SEQ+1, IRS is set to SEG.SEQ and any other control or text should be queued for processing later. ISS should be selected and a <SYN> segment sent of the form:

<SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

If the Snd.WS.OK bit is on, include a WSopt option <WSopt=Rcv.Wind.Shift> in this segment. If the Snd.TS.OK bit is on, include a TSopt <TSval=Snd.TSclock, TSecr=TS.Recent> in this segment. Last.ACK.sent is set to RCV.NXT.

SND.NXT is set to ISS+1 and SND.UNA to ISS. The connection state should be changed to SYN-RECEIVED. Note that any other incoming control or data (combined with SYN) will be processed in the SYN-RECEIVED state, but processing of SYN and ACK should not be repeated. If the listen was not fully specified (i.e., the foreign socket was not fully specified), then the unspecified fields should be filled in now.

fourth other text or control

...

If the state is SYN-SENT then

first check the ACK bit

...

...

fourth check the SYN bit

...

If the SYN bit is on and the security/compartments and precedence are acceptable then, RCV.NXT is set to SEG.SEQ+1, IRS is set to SEG.SEQ, and any acknowledgements on the retransmission queue which are thereby acknowledged should be removed.

Check for a Window Scale option (WSopt); if it is found, save SEG.WSopt in Snd.Wind.Shift; otherwise, set both Snd.Wind.Shift and Rcv.Wind.Shift to zero.

Check for a TSopt option; if one is found, save SEG.TSval in variable TS.Recent and turn on the Snd.TS.OK bit in the connection control block. If the ACK bit is set, use Snd.TSclock - SEG.TSecr as the initial RTT estimate.

If SND.UNA > ISS (our <SYN> has been ACKed), change the connection state to ESTABLISHED, form an <ACK> segment:

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

and send it. If the Snd.Echo.OK bit is on, include a TSopt option <TSval=Snd.TSclock,TSecr=TS.Recent> in this <ACK> segment. Last.ACK.sent is set to RCV.NXT.

Data or controls which were queued for transmission may be included. If there are other controls or text in the segment then continue processing at the sixth step below where the URG bit is checked, otherwise return.

Otherwise enter SYN-RECEIVED, form a <SYN,ACK> segment:

<SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

and send it. If the Snd.Echo.OK bit is on, include a TSopt option <TSval=Snd.TSclock,TSecr=TS.Recent> in this segment. If the Snd.WS.OK bit is on, include a WSopt option <WSopt=Rcv.Wind.Shift> in this segment. Last.ACK.sent is set to RCV.NXT.

If there are other controls or text in the segment, queue them for processing after the ESTABLISHED state has been reached, return.

fifth, if neither of the SYN or RST bits is set then drop the segment and return.

Otherwise,

First, check sequence number

SYN-RECEIVED STATE  
ESTABLISHED STATE  
FIN-WAIT-1 STATE  
FIN-WAIT-2 STATE  
CLOSE-WAIT STATE  
CLOSING STATE  
LAST-ACK STATE  
TIME-WAIT STATE

Segments are processed in sequence. Initial tests on arrival are used to discard old duplicates, but further processing is done in SEG.SEQ order. If a segment's contents straddle the boundary between old and new, only the new parts should be processed.

Rescale the received window field:

TrueWindow = SEG.WND << Snd.Wind.Shift,

and use "TrueWindow" in place of SEG.WND in the following steps.

Check whether the segment contains a Timestamp Option and bit Snd.TS.OK is on. If so:

If SEG.TSval < TS.Recent and the RST bit is off, then test whether connection has been idle less than 24 days; if all are true, then the segment is not acceptable; follow steps below for an unacceptable segment.

If SEG.SEQ is less than or equal to Last.ACK.sent, then save SEG.TSval in variable TS.Recent.

There are four cases for the acceptability test for an incoming segment:

...

If an incoming segment is not acceptable, an acknowledgment should be sent in reply (unless the RST bit is set, if so drop the segment and return):

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

Last.ACK.sent is set to SEG.ACK of the acknowledgment. If the Snd.Echo.OK bit is on, include the Timestamps option <TSval=Snd.TSclock,TSecr=TS.Recent> in this <ACK> segment. Set Last.ACK.sent to SEG.ACK and send the <ACK> segment. After sending the acknowledgment, drop the unacceptable segment and return.

...

fifth check the ACK field.

if the ACK bit is off drop the segment and return.

if the ACK bit is on

...

ESTABLISHED STATE

If SND.UNA < SEG.ACK <= SND.NXT then, set SND.UNA <- SEG.ACK. Also compute a new estimate of round-trip time. If Snd.TS.OK bit is on, use Snd.TSclock - SEG.TSecr; otherwise use the elapsed time since the first segment in the retransmission queue was sent. Any segments on the retransmission queue which are thereby entirely acknowledged...

...

Seventh, process the segment text.

ESTABLISHED STATE

FIN-WAIT-1 STATE

FIN-WAIT-2 STATE

...

Send an acknowledgment of the form:

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

If the Snd.TS.OK bit is on, include Timestamp Option <TSval=Snd.TSclock,TSecr=TS.Recent> in this <ACK> segment. Set Last.ACK.sent to SEG.ACK of the acknowledgment, and send it. This acknowledgment should be piggy-backed on a segment being transmitted if possible without incurring undue delay.

...

#### Appendix E. Timestamps Edge Cases

While the rules laid out for when to calculate RTTM produce the correct results most of the time, there are some edge cases where an incorrect RTTM can be calculated. All of these situations involve the loss of segments. It is felt that these scenarios are rare, and that if they should happen, they will cause a single RTTM measurement to be inflated, which mitigates its effects on RTO calculations.

[Martin03] cites two similar cases when the returning <ACK> is lost, and before the retransmission timer fires, another returning <ACK> segment arrives, which acknowledges the data. In this case, the RTTM calculated will be inflated:

```

clock
tc=1  <A, TSval=1> ----->

tc=2  (lost) <---- <ACK(A), TSecr=1, win=n>
      (RTTM would have been 1)

      (receive window opens, window update is sent)
tc=5  <---- <ACK(A), TSecr=1, win=m>
      (RTTM is calculated at 4)

```

One thing to note about this situation is that it is somewhat bounded by RTO + RTT, limiting how far off the RTTM calculation will be. While more complex scenarios can be constructed that produce larger inflations (e.g., retransmissions are lost), those scenarios involve multiple segment losses, and the connection will have other more serious operational problems than using an inflated RTTM in the RTO calculation.

#### Appendix F. Window Retraction Example

Consider an established TCP connection using a scale factor of 128, Snd.Wind.Shift=7 and Rcv.Wind.Shift=7, that is running with a very small window because the receiver is bottlenecked and both ends are doing small reads and writes.

Consider the ACKs coming back:

SEG.ACK	SEG.WIN	computed SND.WIN	receiver's actual window
1000	2	1256	1300

The sender writes 40 bytes and receiver ACKs:

```
1040      2      1296      1300
```

The sender writes 5 additional bytes and the receiver has a problem.  
Two choices:

```
1045      2      1301      1300  - BEYOND BUFFER
```

```
1045      1      1173      1300  - RETRACTED WINDOW
```

This is a general problem and can happen any time the sender does a write which is smaller than the window scale factor.

In most stacks it is at least partially obscured when the window size is larger than some small number of segments because the stacks prefer to announce windows that are an integral number of segments, rounded up to the next scale factor. This plus silly window suppression tends to cause less frequent, larger window updates. If the window was rounded down to a segment size there is more opportunity to advance the window, the BEYOND BUFFER case above, rather than retracting it.

#### Appendix G. RTO calculation modification

Taking multiple RTT samples per window would shorten the history calculated by the RTO mechanism in [RFC6298], and the below algorithm aims to maintain a similar history as originally intended by [RFC6298].

It is roughly known how many samples a congestion window worth of data will yield, not accounting for ACK compression, and ACK losses. Such events will result in more history of the path being reflected in the final value for RTO, and are uncritical. This modification will ensure that a similar amount of time is taken into account for the RTO estimation, regardless of how many samples are taken per window:

```
ExpectedSamples = ceiling(FlightSize / (SMSS * 2))
```

```
alpha' = alpha / ExpectedSamples
```

```
beta' = beta / ExpectedSamples
```

Note that the factor 2 in ExpectedSamples is due to "Delayed ACKs".

Instead of using alpha and beta in the algorithm of [RFC6298], use

alpha' and beta' instead:

```
RTTVAR <- (1 - beta') * RTTVAR + beta' * |SRTT - R'|  
SRTT <- (1 - alpha') * SRTT + alpha' * R'  
  
(for each sample R')
```

#### Appendix H. Changes from RFC 1323

Several important updates and clarifications to the specification in RFC 1323 are made in these document. The technical changes are summarized below:

- (a) A wrong reference to SND.WND was corrected to SEG.WND in Section 2.3
- (b) Section 2.4 was added describing the unavoidable window retraction issue, and explicitly describing the mitigation steps necessary.
- (c) In Section 3.2 the wording how the Timestamps option negotiation is to be performed was updated with RFC2119 wording. Further, a number of paragraphs were added to clarify the expected behavior with a compliant implementation using TSopt, as RFC1323 left room for interpretation - e.g. potential late enablement of TSopt.
- (d) The description of which TSecr values can be used to update the measured RTT has been clarified. Specifically, with timestamps, the Karn algorithm [Karn87] is disabled. The Karn algorithm disables all RTT measurements during retransmission, since it is ambiguous whether the <ACK> is for the original segment, or the retransmitted segment. With timestamps, that ambiguity is removed since the TSecr in the <ACK> will contain the TSval from whichever data segment made it to the destination.
- (e) RTTM update processing explicitly excludes segments not updating SND.UNA. The original text could be interpreted to allow taking RTT samples when SACK acknowledges some new, non-continuous data.
- (f) In RFC1323, section 3.4, step (2) of the algorithm to control which timestamp is echoed was incorrect in two regards:

(1) It failed to update TS.recent for a retransmitted segment that resulted from a lost <ACK>.

(2) It failed if SEG.LEN = 0.

In the new algorithm, the case of SEG.TSval >= TS.recent is included for consistency with the PAWS test.

- (g) It is now recommended that the Timestamps option is included in <RST> segments if the incoming segment contained a Timestamps option.
- (h) <RST> segments are explicitly excluded from PAWS processing.
- (i) Added text to clarify the precedence between regular TCP [RFC0793] and this document Timestamps option / PAWS processing. Discussion about combined acceptability checks are ongoing.
- (j) Snd.TSoffset and Snd.TSclock variables have been added. Snd.TSclock is the sum of my.TSclock and Snd.TSoffset. This allows the starting points for timestamp values to be randomized on a per-connection basis. Setting Snd.TSoffset to zero yields the same results as [RFC1323].
- (k) Appendix A has been expanded with information about the TCP Urgent Pointer. An earlier revision contained text around the TCP MSS option, which was split off into [RFC6691].
- (l) One correction was made to the Event Processing Summary in Appendix D. In SEND CALL/ESTABLISHED STATE, RCV.WND is used to fill in the SEG.WND value, not SND.WND.
- (m) Appendix G was added to exemplify how an RTO calculation might be updated to properly take the much higher RTT sampling frequency enabled by the Timestamps option into account.

Editorial changes of the document, that don't impact the implementation or function of the mechanisms described in this document include:

- (a) Removed much of the discussion in Section 1 to streamline the document. However, detailed examples and discussions in Section 2, Section 3 and Section 4 are kept as guideline for implementers.



- (b) Removed references to "new" options, as the options were introduced in [RFC1323] already. Changed the text in Section 1.3 to specifically address TS and WS options.
- (c) Section 1.4 was added for [RFC2119] wording. Normative text was updated with the appropriate phrases.
- (d) Added < > brackets to mark specific types of segments, and replaced most occurrences of "packet" with "segment", where TCP segments are referred to.
- (e) Updated the text in Section 3 to take into account what has been learned since [RFC1323].
- (f) Removed the list of changes between [RFC1323] and prior versions. These changes are mentioned in Appendix C of [RFC1323].
- (g) Moved Appendix Changes from RFC 1323 to the end of the appendices for easier lookup. In addition, the entries were split into a technical and an editorial part, and sorted to roughly correspond with the sections in the text where they apply.

#### Authors' Addresses

David Borman  
Quantum Corporation  
Mendota Heights MN 55120  
USA

Email: david.borman@quantum.com

Bob Braden  
University of Southern California  
4676 Admiralty Way  
Marina del Rey CA 90292  
USA

Email: braden@isi.edu

Van Jacobson  
Packet Design  
2465 Latham Street  
Mountain View CA 94040  
USA

Email: [van@packetdesign.com](mailto:van@packetdesign.com)

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

Email: [rs@netapp.com](mailto:rs@netapp.com)



Internet Draft  
draft-ietf-tcpm-fastopen-04.txt  
Intended status: Experimental  
Expiration date: February, 2014

Y. Cheng  
J. Chu  
S. Radhakrishnan  
A. Jain  
Google, Inc.  
July 15, 2013

## TCP Fast Open

### Status of this Memo

Distribution of this memo is unlimited.

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/lid-abstracts.html>

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

### Copyright Notice

Copyright (c) 2013 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.

### Abstract

TCP Fast Open (TFO) allows data to be carried in the SYN and SYN-ACK packets and consumed by the receiving end during the initial connection handshake, thus saving up to one full round trip time (RTT) compared to the standard TCP, which requires a three-way handshake (3WHS) to complete before data can be exchanged. However TFO deviates from the standard TCP semantics in that the data in the SYN could be replayed to an application in some rare circumstances. Applications should not use TFO unless they can tolerate this issue, which is detailed in the Applicability section.

## 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]. TFO refers to TCP Fast Open. Client refers to the TCP's active open side and server refers to the TCP's passive open side.

## Table of Contents

1. Introduction . . . . .	3
2. Data In SYN . . . . .	3
2.1 Relaxing TCP Semantics on Duplicated SYNs . . . . .	4
2.2. SYNs with Spoofed IP Addresses . . . . .	4
3. Protocol Overview . . . . .	5
4. Protocol Details . . . . .	7
4.1. Fast Open Cookie . . . . .	7
4.1.1. TCP Options . . . . .	7
4.1.2. Server Cookie Handling . . . . .	8
4.1.3. Client Cookie Handling . . . . .	9
4.2. Fast Open Protocol . . . . .	9
4.2.1. Fast Open Cookie Request . . . . .	10
4.2.2. TCP Fast Open . . . . .	11
5. Security Considerations . . . . .	13
5.1. Resource Exhaustion Attack by SYN Flood with Valid Cookies . . . . .	13
5.1.1 Attacks from behind Sharing Public IPs (NATs) . . . . .	14
5.2. Amplified Reflection Attack to Random Host . . . . .	15
6. TFO's Applicability . . . . .	16
6.1 Duplicate Data in SYNs . . . . .	16
6.2 Potential Performance Improvement . . . . .	16
6.3. Example: Web Clients and Servers . . . . .	16
6.3.1. HTTP Request Replay . . . . .	16
6.3.2. Comparison with HTTP Persistent Connections . . . . .	17
7. Open Areas for Experimentation . . . . .	17
7.1. Performance impact due to middle-boxes and NAT . . . . .	18
7.2. Cookie-less Fast Open . . . . .	18
8. Related Work . . . . .	18

8.1. T/TCP . . . . .	19
8.2. Common Defenses Against SYN Flood Attacks . . . . .	19
8.3. TCP Cookie Transaction (TCPCT) . . . . .	19
8.4. Speculative Connections by the Applications . . . . .	19
9. IANA Considerations . . . . .	19
10. Acknowledgement . . . . .	20
11. References . . . . .	20
11.1. Normative References . . . . .	20
11.2. Informative References . . . . .	20
Appendix A. Example Socket API Changes to support TFO . . . . .	22
Authors' Addresses . . . . .	22

## 1. Introduction

TCP Fast Open (TFO) enables data to be exchanged safely during TCP's connection handshake. This document describes a design that enables applications to save a round trip while avoiding severe security ramifications. At the core of TFO is a security cookie used by the server side to authenticate a client initiating a TFO connection. This document covers the details of exchanging data during TCP's initial handshake, the protocol for TFO cookies, potential new security vulnerabilities and their mitigation, and the new socket API.

TFO is motivated by the performance needs of today's Web applications. Current TCP only permits data exchange after 3WSH [RFC793], which adds one RTT to network latency. For short Web transfers this additional RTT is a significant portion of overall network latency [THK98], even when HTTP persistent connection is widely used. For example, the Chrome browser keeps TCP connections idle for up to 5 minutes but 35% of Chrome HTTP requests are made on new TCP connections [RCCJR11]. For such Web and Web-like applications placing data in the SYN can yield significant latency improvements. Next we describe how we resolve the challenges that arise upon doing so.

## 2. Data In SYN

Standard TCP already allows data to be carried in SYN packets ([RFC793], section 3.4) but forbids the receiver from delivering it to the application until 3WSH is completed. This is because TCP's initial handshake serves to capture old or duplicate SYNs.

Allowing data in SYN packets to be delivered raises two issues discussed in the following subsections. These issues make TFO unsuitable for certain applications. Therefore TCP implementations

MUST NOT use TFO by default, but only use TFO if requested explicitly by the application on a per service port basis. Applications need to evaluate TFO applicability described in Section 6 before using TFO.

## 2.1 Relaxing TCP Semantics on Duplicated SYNs

TFO allows data to be delivered to the application before 3WHS is completed, thus opening itself to a data integrity issue in either of the two cases below:

- a) the receiver host receives data in a duplicate SYN after it has forgotten it received the original SYN (e.g. due to a reboot);
- b) the duplicate is received after the connection created by the original SYN has been closed and the close was initiated by the sender (so the receiver will not be protected by the 2MSL TIMEWAIT state).

The now obsoleted T/TCP [RFC1644] attempted to address these issues. It is not successful and not deployed due to various vulnerabilities [PHRACK98]. Rather than trying to capture all dubious SYN packets to make TFO 100% compatible with TCP semantics, we made a design decision early on to accept old SYN packets with data, i.e., to restrict TFO use to a class of applications (Section 6) that are tolerant of duplicate SYN packets with data. We believe this is the right design trade-off balancing complexity with usefulness.

## 2.2. SYNs with Spoofed IP Addresses

Standard TCP suffers from the SYN flood attack [RFC4987] because bogus SYN packets, i.e., SYN packets with spoofed source IP addresses can easily fill up a listener's small queue, causing a service port to be blocked completely until timeouts. Secondary damage comes from these SYN requests taking up memory space. Though this is less of an issue today as servers typically have plenty of memory.

TFO goes one step further to allow server-side TCP to send up data to the application layer before 3WHS is completed. This opens up serious new vulnerabilities. Applications serving ports that have TFO enabled may waste lots of CPU and memory resources processing the requests and producing the responses. If the response is much larger than the request, the attacker can further mount an amplified reflection attack against victims of choice beyond the TFO server itself.

Numerous mitigation techniques against regular SYN flood attacks exist and have been well documented [RFC4987]. Unfortunately none are applicable to TFO. We propose a server-supplied cookie to mitigate these new vulnerabilities in Section 3 and evaluate the effectiveness

of the defense in Section 7.

### 3. Protocol Overview

The key component of TFO is the Fast Open Cookie (cookie), a message authentication code (MAC) tag generated by the server. The client requests a cookie in one regular TCP connection, then uses it for future TCP connections to exchange data during 3WHS:

Requesting a Fast Open Cookie:

1. The client sends a SYN with a Fast Open Cookie Request option.
2. The server generates a cookie and sends it through the Fast Open Cookie option of a SYN-ACK packet.
3. The client caches the cookie for future TCP Fast Open connections (see below).

Performing TCP Fast Open:

1. The client sends a SYN with Fast Open Cookie option and data.
2. The server validates the cookie:
  - a. If the cookie is valid, the server sends a SYN-ACK acknowledging both the SYN and the data. The server then delivers the data to the application.
  - b. Otherwise, the server drops the data and sends a SYN-ACK acknowledging only the SYN sequence number.
3. If the server accepts the data in the SYN packet, it may send the response data before the handshake finishes. The max amount is governed by the TCP's congestion control [RFC5681].
4. The client sends an ACK acknowledging the SYN and the server data. If the client's data is not acknowledged, the client retransmits the data in the ACK packet.
5. The rest of the connection proceeds like a normal TCP connection. The client can repeat many Fast Open operations once it acquires a cookie (until the cookie is expired by the server). Thus TFO is useful for applications that have temporal locality on client and server connections.

Requesting Fast Open Cookie in connection 1:



TCP A (Client)		TCP B(Server)
CLOSED		LISTEN
#1 SYN-SENT	----- <SYN, CookieOpt=NIL> ----->	SYN-RCVD
#2 ESTABLISHED (caches cookie C)	<----- <SYN, ACK, CookieOpt=C> -----	SYN-RCVD

Performing TCP Fast Open in connection 2:

TCP A (Client)		TCP B(Server)
CLOSED		LISTEN
#1 SYN-SENT	----- <SYN=x, CookieOpt=C, DATA_A> ----->	SYN-RCVD
#2 ESTABLISHED	<----- <SYN=y, ACK=x+len(DATA_A)+1> -----	SYN-RCVD
#3 ESTABLISHED	<----- <ACK=x+len(DATA_A)+1, DATA_B>-----	SYN-RCVD
#4 ESTABLISHED	----- <ACK=y+1>----->	ESTABLISHED
#5 ESTABLISHED	---- <ACK=y+len(DATA_B)+1>----->	ESTABLISHED

## 4. Protocol Details

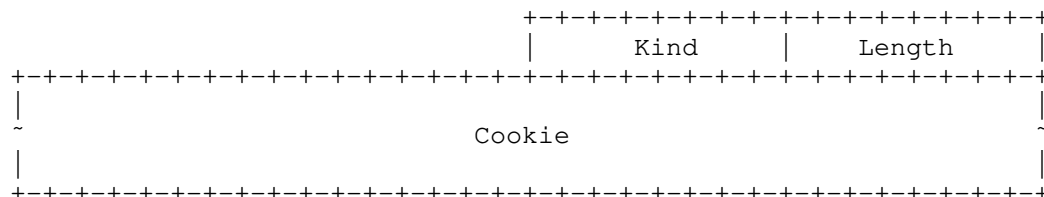
### 4.1. Fast Open Cookie

The Fast Open Cookie is designed to mitigate new security vulnerabilities in order to enable data exchange during handshake. The cookie is a message authentication code tag generated by the server and is opaque to the client; the client simply caches the cookie and passes it back on subsequent SYN packets to open new connections. The server can expire the cookie at any time to enhance security.

#### 4.1.1. TCP Options

##### Fast Open Cookie Option

The server uses this option to grant a cookie to the client in the SYN-ACK packet; the client uses it to pass the cookie back to the server in subsequent SYN packets.



Kind                    1 byte: constant TBD (assigned by IANA)

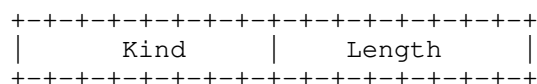
Length                 1 byte: range 6 to 18 (bytes); limited by  
                         remaining space in the options field.  
                         The number MUST be even.

Cookie                 4 to 16 bytes (Length - 2)

Options with invalid Length values or without SYN flag set MUST be ignored. The minimum Cookie size is 4 bytes. Although the diagram shows a cookie aligned on 32-bit boundaries, alignment is not required.

##### Fast Open Cookie Request Option

The client uses this option in the SYN packet to request a cookie from a TFO-enabled server



Kind                    1 byte: same as the Fast Open Cookie option

Length                 1 byte: constant 2. This distinguishes the option

from the Fast Open cookie option.  
Options with invalid Length values, without SYN flag set, or with ACK flag set MUST be ignored.

#### 4.1.2. Server Cookie Handling

The server is in charge of cookie generation and authentication. The cookie SHOULD be a message authentication code tag with the following properties:

1. The cookie authenticates the client's (source) IP address of the SYN packet. The IP address may be an IPv4 or IPv6 address.
2. The cookie can only be generated by the server and can not be fabricated by any other parties including the client.
3. The generation and verification are fast relative to the rest of SYN and SYN-ACK processing.
4. A server may encode other information in the cookie, and accept more than one valid cookie per client at any given time. But this is all server implementation dependent and transparent to the client.
5. The cookie expires after a certain amount of time. The reason for cookie expiration is detailed in the "Security Consideration" section. This can be done by either periodically changing the server key used to generate cookies or including a timestamp when generating the cookie.

To gradually invalidate cookies over time, the server can implement key rotation to generate and verify cookies using multiple keys. This approach is useful for large-scale servers to retain Fast Open rolling key updates. We do not specify a particular mechanism because the implementation is server specific.

The server supports the cookie generation and verification operations:

- GetCookie(IP\_Address): returns a (new) cookie
- IsCookieValid(IP\_Address, Cookie): checks if the cookie is valid, i.e., it has not expired and it authenticates the client IP address.

Example Implementation: a simple implementation is to use AES\_128 to encrypt the IPv4 (with padding) or IPv6 address and truncate to 64 bits. The server can periodically update the key to expire the cookies. AES encryption on recent processors is fast and takes only a

few hundred nanoseconds [RCCJR11].

If only one valid cookie is allowed per-IP and the server can regenerate the cookie independently, the best validation process is to simply regenerate a valid cookie and compare it against the incoming cookie. In that case if the incoming cookie fails the check, a valid cookie is readily available to be sent to the client.

#### 4.1.3. Client Cookie Handling

The client **MUST** cache cookies from servers for later Fast Open connections. For a multi-homed client, the cookies are both client and server IP dependent. Beside the cookie, we **RECOMMEND** that the client caches the MSS and RTT to the server to enhance performance.

The MSS advertised by the server is stored in the cache to determine the maximum amount of data that can be supported in the SYN packet. This information is needed because data is sent before the server announces its MSS in the SYN-ACK packet. Without this information, the data size in the SYN packet is limited to the default MSS of 536 bytes [RFC1122]. The client **SHOULD** update the cache MSS value whenever it discovers new MSS value, e.g., through path MTU discovery.

Caching RTT allows seeding a more accurate SYN timeout than the default value [RFC6298]. This lowers the performance penalty if the network or the server drops the SYN packets with data or the cookie options.

The cache replacement algorithm is not specified and is left to the implementations.

Note that before TFO sees wide deployment, clients **SHOULD** cache negative responses from servers in order to reduce the amount of futile TFO attempts. Since TFO is enabled on a per-service port basis but cookies are independent of service ports, clients' cache should include remote port numbers too.

#### 4.2. Fast Open Protocol

One predominant requirement of TFO is to be fully compatible with existing TCP implementations, both on the client and the server sides.

The server keeps two variables per listening port:

FastOpenEnabled: default is off. It **MUST** be turned on explicitly by the application. When this flag is off, the server does not perform

any TFO related operations and MUST ignore all cookie options.

**PendingFastOpenRequests:** tracks number of TFO connections in SYN-RCVD state. If this variable goes over a preset system limit, the server SHOULD disable TFO for all new connection requests until PendingFastOpenRequests drops below the system limit. This variable is used for defending some vulnerabilities discussed in the "Security Considerations" section.

The server keeps a FastOpened flag per TCB to mark if a connection has successfully performed a TFO.

#### 4.2.1. Fast Open Cookie Request

Any client attempting TFO MUST first request a cookie from the server with the following steps:

1. The client sends a SYN packet with a Fast Open Cookie Request option.
2. The server SHOULD respond with a SYN-ACK based on the procedures in the "Server Cookie Handling" section. This SYN-ACK SHOULD contain a Fast Open Cookie option if the server currently supports TFO for this listener port.
3. If the SYN-ACK contains a Fast Open Cookie option, the client replaces the cookie and other information as described in the "Client Cookie Handling" section. Otherwise, if the SYN-ACK is first seen, i.e., not a (spurious) retransmission, the client MAY remove the server information from the cookie cache. If the SYN-ACK is a spurious retransmission without valid Fast Open Cookie Option, the client does nothing to the cookie cache for the reasons below.

The network or servers may drop the SYN or SYN-ACK packets with the new cookie options, which will cause SYN or SYN-ACK timeouts. We RECOMMEND both the client and the server to retransmit SYN and SYN-ACK without the cookie options on timeouts. This ensures the connections of cookie requests will go through and lowers the latency penalty (of dropped SYN/SYN-ACK packets). The obvious downside for maximum compatibility is that any regular SYN drop will fail the cookie (although one can argue the delay in the data transmission till after 3WSH is justified if the SYN drop is due to network congestion). Next section describes a heuristic to detect such drops when the client receives the SYN-ACK.

We also RECOMMEND the client to record servers that failed to respond to cookie requests and only attempt another cookie request after

certain period. An alternate proposal is to request cookie in FIN instead since FIN-drop by incompatible middle-box does not affect latency. However such paths are likely to drop SYN packet with data later, and many applications close the connections with RST instead, so the actual benefit of this approach is not clear.

#### 4.2.2. TCP Fast Open

Once the client obtains the cookie from the target server, it can perform subsequent TFO connections until the cookie is expired by the server.

Client: Sending SYN

To open a TFO connection, the client **MUST** have obtained a cookie from the server:

1. Send a SYN packet.
  - a. If the SYN packet does not have enough option space for the Fast Open Cookie option, abort TFO and fall back to regular 3WHS.
  - b. Otherwise, include the Fast Open Cookie option with the cookie of the server. Include any data up to the cached server MSS or default 536 bytes.
2. Advance to SYN-SENT state and update SND.NXT to include the data accordingly.
3. If RTT is available from the cache, seed SYN timer according to [RFC6298].

To deal with network or servers dropping SYN packets with payload or unknown options, when the SYN timer fires, the client **SHOULD** retransmit a SYN packet without data and Fast Open Cookie options.

Server: Receiving SYN and responding with SYN-ACK

Upon receiving the SYN packet with Fast Open Cookie option:

1. Initialize and reset a local FastOpened flag. If FastOpenEnabled is false, go to step 5.
2. If PendingFastOpenRequests is over the system limit, go to step 5.
3. If IsCookieValid() in section 4.1.2 returns false, go to step 5.
4. Buffer the data and notify the application. Set FastOpened flag

and increment PendingFastOpenRequests.

5. Send the SYN-ACK packet. The packet MAY include a Fast Open Option. If FastOpened flag is set, the packet acknowledges the SYN and data sequence. Otherwise it acknowledges only the SYN sequence. The server MAY include data in the SYN-ACK packet if the response data is readily available. Some application may favor delaying the SYN-ACK, allowing the application to process the request in order to produce a response, but this is left up to the implementation.
6. Advance to the SYN-RCVD state. If the FastOpened flag is set, the server MUST follow the congestion control [RFC5681], in particular the initial congestion window [RFC3390], to send more data packets.

Note that if SYN-ACK is lost, regular TCP reduces the initial congestion window before sending any data. In this case TFO is slightly more aggressive in the first data round trip even though it does not change the congestion control.

If the SYN-ACK timer fires, the server SHOULD retransmit a SYN-ACK segment with neither data nor Fast Open Cookie options for compatibility reasons.

A special case is simultaneous open where the SYN receiver is a client in SYN-SENT state. The protocol remains the same because [RFC793] already supports both data in SYN and simultaneous open. But the client's socket may have data available to read before it's connected. This document does not cover the corresponding API change.

Client: Receiving SYN-ACK

The client SHOULD perform the following steps upon receiving the SYN-ACK:

1. Update the cookie cache if the SYN-ACK has a Fast Open Cookie Option or MSS option or both.
2. Send an ACK packet. Set acknowledgment number to RCV.NXT and include the data after SND.UNA if data is available.
3. Advance to the ESTABLISHED state.

Note there is no latency penalty if the server does not acknowledge the data in the original SYN packet. The client SHOULD retransmit any unacknowledged data in the first ACK packet in step 2. The data exchange will start after the handshake like a regular TCP

connection.

If the client has timed out and retransmitted only regular SYN packets, it can heuristically detect paths that intentionally drop SYN with Fast Open option or data. If the SYN-ACK acknowledges only the initial sequence and does not carry a Fast Open cookie option, presumably it is triggered by a retransmitted (regular) SYN and the original SYN or the corresponding SYN-ACK was lost.

Server: Receiving ACK

Upon receiving an ACK acknowledging the SYN sequence, the server decrements PendingFastOpenRequests and advances to the ESTABLISHED state. No special handling is required further.

## 5. Security Considerations

The Fast Open cookie stops an attacker from trivially flooding spoofed SYN packets with data to burn server resources or to mount an amplified reflection attack on random hosts. The server can defend against spoofed SYN floods with invalid cookies using existing techniques [RFC4987]. We note that although generating bogus cookies is cost-free, the cost of validating the cookies, inherent to any authentication scheme, may not be substantial compared to processing a regular SYN packet.

### 5.1. Resource Exhaustion Attack by SYN Flood with Valid Cookies

However, the attacker may still obtain cookies from some compromised hosts, then flood spoofed SYN with data and "valid" cookies (from these hosts or other vantage points). With DHCP, it's possible to obtain cookies of past IP addresses without compromising any host. Below we identify new vulnerabilities of TFO and describe the countermeasures.

Like regular TCP handshakes, TFO is vulnerable to such an attack. But the potential damage can be much more severe. Besides causing temporary disruption to service ports under attack, it may exhaust server CPU and memory resources. Such an attack will show up on application server logs as a application level DoS from Bot-nets, triggering other defenses and alerts.

For this reason it is crucial for the TFO server to limit the maximum number of total pending TFO connection requests, i.e., PendingFastOpenRequests. When the limit is exceeded, the server temporarily disables TFO entirely as described in "Server Cookie Handling". Then subsequent TFO requests will be downgraded to regular connection requests, i.e., with the data dropped and only SYN



acknowledged. This allows regular SYN flood defense techniques [RFC4987] like SYN-cookies to kick in and prevent further service disruption.

The main impact of SYN floods against the standard TCP stack is not directly from the floods themselves costing TCP processing overhead or host memory, but rather from the spoofed SYN packets filling up the often small listener's queue.

On the other hand, TFO SYN floods can cause damage directly if admitted without limit into the stack. The RST packets from the spoofed host will fuel rather than defeat the SYN floods as compared to the non-TFO case, because the attacker can flood more SYNs with data to cost more data processing resources. For this reason, a TFO server needs to monitor the connections in SYN-RCVD being reset in addition to imposing a reasonable max queue length. Implementations may combine the two, e.g., by continuing to account for those connection requests that have just been reset against the listener's PendingFastOpenRequests until a timeout period has passed.

Limiting the maximum number of pending TFO connection requests does make it easy for an attacker to overflow the queue, causing TFO to be disabled. We argue that causing TFO to be disabled is unlikely to be of interest to attackers because the service will remain intact without TFO hence there is hardly any real damage.

#### 5.1.1 Attacks from behind Sharing Public IPs (NATs)

An attacker behind NAT can easily obtain valid cookies to launch the above attack to hurt other clients that share the path. [BRISCOE12] suggested that the server can extend cookie generation to include the TCP timestamp---GetCookie(IP\_Address, Timestamp)---and implement it by encrypting the concatenation of the two values to generate the cookie. The client stores both the cookie and its corresponding timestamp, and echoes both in the SYN. The server then implements IsCookieValid(IP\_Address, Timestamp, Cookie) by encrypting the IP and timestamp data and comparing it with the cookie value.

This enables the server to issue different cookies to clients that share the same IP address, hence can selectively discard those misused cookies from the attacker. However the attacker can simply repeat the attack with new cookies. The server would eventually need to throttle all requests from the IP address just like the current approach. Moreover this approach requires modifying [RFC 1323] to send non-zero Timestamp Echo Reply in SYN, potentially cause firewall issues. Therefore we believe the benefit does not outweigh the drawbacks.

## 5.2. Amplified Reflection Attack to Random Host

Limiting PendingFastOpenRequests with a system limit can be done without Fast Open Cookies and would protect the server from resource exhaustion. It would also limit how much damage an attacker can cause through an amplified reflection attack from that server. However, it would still be vulnerable to an amplified reflection attack from a large number of servers. An attacker can easily cause damage by tricking many servers to respond with data packets at once to any spoofed victim IP address of choice.

With the use of Fast Open Cookies, the attacker would first have to steal a valid cookie from its target victim. This likely requires the attacker to compromise the victim host or network first.

The attacker here has little interest in mounting an attack on the victim host that has already been compromised. But it may be motivated to disrupt the victim's network. Since a stolen cookie is only valid for a single server, it has to steal valid cookies from a large number of servers and use them before they expire to cause sufficient damage without triggering the defense.

One can argue that if the attacker has compromised the target network or hosts, it could perform a similar but simpler attack by injecting bits directly. The degree of damage will be identical, but TFO-specific attack allows the attacker to remain anonymous and disguises the attack as from other servers.

The best defense is for the server not to respond with data until handshake finishes. In this case the risk of amplification reflection attack is completely eliminated. But the potential latency saving from TFO may diminish if the server application produces responses earlier before the handshake completes.

## 6. TFO's Applicability

This section is to help applications considering TFO to evaluate TFO's benefits and drawbacks using the Web client and server applications as an example throughout. A proposed socket API change is in the Appendix.

### 6.1 Duplicate Data in SYNs

It is possible, though uncommon, that using TFO results in the first data written to a socket to be delivered more than once to the application on the remote host (Section 2.1). This replay potential only applies to data in the SYN but not subsequent data exchanges. The client **MUST NOT** use TFO to send data in the SYN, and the server **MUST NOT** accept data in the SYN if it cannot handle receiving the same SYN data more than once, due to reasons described before.

### 6.2 Potential Performance Improvement

TFO is designed for latency-conscious applications that are sensitive to TCP's initial connection setup delay. To benefit from TFO, the first application data unit (e.g., an HTTP request) needs to be no more than TCP's maximum segment size (minus options used in SYN). Otherwise the remote server can only process the client's application data unit once the rest of it is delivered after the initial handshake, diminishing TFO's benefit.

To the extent possible, applications **SHOULD** reuse the connection to take advantage of TCP's built-in congestion control and reduce connection setup overhead. An application employs too many short-lived connections will negatively impact network stability, as these connections often exit before TCP's congestion control algorithm takes effect.

### 6.3. Example: Web Clients and Servers

#### 6.3.1. HTTP Request Replay

While TFO is motivated by Web applications, the browser should not use TFO to send requests in SYNs if those requests cannot tolerate replays. One example is POST requests without application-layer transaction protection (e.g., a unique identifier in the request header).

TFO is particularly useful for GET requests. Even though not all GET requests are idempotent, GETs are frequently replayed today across striped TCP connections. After a server receives an HTTP request but before the ACKs of the requests reach the browser, the browser may

timeout and retry the same request on another (possibly new) TCP connection. This differs from a TFO replay only in that the replay is initiated by the browser, not by the TCP stack.

Finally, TFO is safe and useful for HTTPS requests because it saves the first SSL handshake RTT and the HTTP request is sent after the connection establishes.

### 6.3.2. Comparison with HTTP Persistent Connections

Is TFO useful given the wide deployment of HTTP persistent connections? The short answer is yes. Studies [RCCJR11][AERG11] show that the average number of transactions per connection is between 2 and 4, based on large-scale measurements from both servers and clients. In these studies, the servers and clients both kept idle connections up to several minutes, well into "human think" time.

Keeping connections open and idle even longer risks a greater performance penalty. [HNESSK10][MQXMZ11] show that the majority of home routers and ISPs fail to meet the the 124-minute idle timeout mandated in [RFC5382]. In [MQXMZ11], 35% of mobile ISPs silently timeout idle connections within 30 minutes. End hosts, unaware of silent middle-box timeouts, suffer multi-minute TCP timeouts upon using those long-idle connections.

To circumvent this problem, some applications send frequent TCP keep-alive probes. However, this technique drains power on mobile devices [MQXMZ11]. In fact, power has become such a prominent issue in modern LTE devices that mobile browsers close HTTP connections within seconds or even immediately [SOUDERS11].

[RCCJR11] studied Chrome browser performance based on 28 days of global statistics. The Chrome browser keeps idle HTTP persistent connections for 5 to 10 minutes. However the average number of the transactions per connection is only 3.3 and TCP 3WHS accounts for up to 25% of the HTTP transaction network latency. The authors tested a Linux TFO implementation with TFO enabled Chrome browser on popular web sites in emulated environments such as residential broadband and mobile networks. They showed that TFO improves page load time by 10% to 40%.

## 7. Open Areas for Experimentation

We now outline some areas that need experimentation in the Internet and under different network scenarios. These experiments should help the community evaluate Fast Open benefits and risks towards further standardization and implementation of Fast Open and its related protocols.

### 7.1. Performance impact due to middle-boxes and NAT

[MAF04] found that some middle-boxes and end-hosts may drop packets with unknown TCP options. Studies [LANGLEY06, HNRGHT11] both found that 6% of the probed paths on the Internet drop SYN packets with data or with unknown TCP options. The TFO protocol deals with this problem by falling back to regular TCP handshake and re-transmitting SYN without data or cookie options after the initial SYN timeout. Moreover the implementation is recommended to negatively cache such incidents to avoid recurring timeouts. Further study is required to evaluate the performance impact of these malicious drop behaviors.

Another interesting study is the (loss of) TFO performance benefit behind certain carrier-grade NAT. Typically hosts behind a NAT sharing the same IP address will get the same cookie for the same server. This will not prevent TFO from working. But on some carrier-grade NAT configurations where every new TCP connection from the same physical host uses a different public IP address, TFO does not provide latency benefits. However, there is no performance penalty either, as described in Section "Client: Receiving SYN-ACK".

### 7.2. Cookie-less Fast Open

The cookie mechanism mitigates resource exhaustion and amplification attacks. However cookies are not necessary if the server has application-level protection or is immune to these attacks. For example a Web server that only replies with a simple HTTP redirect response that fits in the SYN-ACK packet may not care about resource exhaustion. Such an application can safely disable TFO cookie checks.

Disabling cookies simplifies both the client and the server, as the client no longer needs to request a cookie and the server no longer needs to check or generate cookies. Disabling cookies also potentially simplifies configuration, as the server no longer needs a key. It may be preferable to enable SYN cookies and disable TFO [RFC4987] when a server is overloaded by a large-scale Bot-net attack.

Careful experimentation is necessary to evaluate if cookie-less TFO is practical. The implementation can provide an experimental feature to allow zero length, or null, cookies as opposed to the minimum 4 bytes cookies. Thus the server may return a null cookie and the client will send data in SYN with it subsequently. If the server believes it's under a DoS attack through other defense mechanisms, it can switch to regular Fast Open for listener sockets.

## 8. Related Work

### 8.1. T/TCP

TCP Extensions for Transactions [RFC1644] attempted to bypass the three-way handshake, among other things, hence shared the same goal but also the same set of issues as TFO. It focused most of its effort battling old or duplicate SYNs, but paid no attention to security vulnerabilities it introduced when bypassing 3WS [PHRACK98].

As stated earlier, we take a practical approach to focus TFO on the security aspect, while allowing old, duplicate SYN packets with data after recognizing that 100% TCP semantics is likely infeasible. We believe this approach strikes the right tradeoff, and makes TFO much simpler and more appealing to TCP implementers and users.

### 8.2. Common Defenses Against SYN Flood Attacks

[RFC4987] studies on mitigating attacks from regular SYN flood, i.e., SYN without data. But from the stateless SYN-cookies to the stateful SYN Cache, none can preserve data sent with SYN safely while still providing an effective defense.

The best defense may be to simply disable TFO when a host is suspected to be under a SYN flood attack, e.g., the SYN backlog is filled. Once TFO is disabled, normal SYN flood defenses can be applied. The "Security Consideration" section contains a thorough discussion on this topic.

### 8.3. TCP Cookie Transaction (TCPCT)

TCPCT [RFC6013] eliminates server state during initial handshake and defends spoofing DoS attacks. Like TFO, TCPCT allows SYN and SYN-ACK packets to carry data. But the server can only send up to MSS bytes of data during the handshake instead of the initial congestion window unlike TFO. Therefore applications like Web may not receive the latency benefit as TFO.

### 8.4. Speculative Connections by the Applications

Some Web browsers maintain a history of the domains for frequently visited web pages. The browsers then speculatively pre-open TCP connections to these domains before the user initiates any requests for them [BELSHELL]. The downside of this approach is that it wastes server and network resources by initiating and maintaining idle connections; It is also subject to the NAT timeout issues described in Section 6.3.2. TFO offers similar performance improvement without the added overhead.

## 9. IANA Considerations

The Fast Open Cookie Option and Fast Open Cookie Request Option define no new namespace. The options require IANA to allocate one value from the TCP option Kind namespace. Early implementation before the IANA allocation SHOULD follow [EXPOPT] and use experimental option 254 and magic number 0xF989 (16 bits), then migrate to the new option after the allocation accordingly.

## 10. Acknowledgement

We thank Rick Jones, Bob Briscoe, Adam Langley, Matt Mathis, Neal Cardwell, Roberto Peon, William Chan, Eric Dumazet, and Tom Herbert for their feedbacks. We especially thank Barath Raghavan for his contribution on the security design of Fast Open and proofreading this draft numerous times.

## 11. References

### 11.1. Normative References

- [RFC793] Postel, J. "Transmission Control Protocol", RFC 793, September 1981.
- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.
- [RFC5382] S. Guha, Ed., Biswas, K., Ford B., Sivakumar S., Srisuresh, P., "NAT Behavioral Requirements for TCP", RFC 5382
- [RFC5681] Allman, M., Paxson, V. and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.
- [RFC6298] Paxson, V., Allman, M., Chu, J. and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.
- [RFC6928] Chu, J., Dukkkipati, N., Cheng, Y. and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, April 2013.

### 11.2. Informative References

- [AERG11] M. Al-Fares, K. Elmeleegy, B. Reed, and I. Gashinsky, "Overclocking the Yahoo! CDN for Faster Web Page Loads". In Proceedings of Internet Measurement Conference, November 2011.
- [EXPOPT] Touch, Joe, "Shared Use of Experimental TCP Options", Internet-Draft draft-ietf-tcpm-experimental-options (work in progress), June 2013.
- [HNESSK10] S. Haetonen, A. Nyrhinen, L. Eggert, S. Strowes, P.

Sarolahti, M. Kojo., "An Experimental Study of Home Gateway Characteristics". In Proceedings of Internet Measurement Conference. October 2010

[HNRGHT11] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, H. Tokuda, "Is it Still Possible to Extend TCP?". In Proceedings of Internet Measurement Conference. November 2011.

[LANGLEY06] Langley, A, "Probing the viability of TCP extensions", URL <http://www.imperialviolet.org/binary/ecntest.pdf>

[MAF04] Medina, A., Allman, M., and S. Floyd, "Measuring Interactions Between Transport Protocols and Middleboxes", In Proceedings of Internet Measurement Conference, October 2004.

[MQXMZ11] Z. Mao, Z. Qian, Q. Xu, Z. Mao, M. Zhang. "An Untold Story of Middleboxes in Cellular Networks", In Proceedings of SIGCOMM. August 2011.

[PHRACK98] "T/TCP vulnerabilities", Phrack Magazine, Volume 8, Issue 53 artical 6. July 8, 1998. URL <http://www.phrack.com/issues.html?issue=53&id=6>

[QWGMSS11] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, O. Spatscheck. "Profiling Resource Usage for Mobile Applications: A Cross-layer Approach", In Proceedings of International Conference on Mobile Systems. April 2011.

[RCCJR11] Radhakrishnan, S., Cheng, Y., Chu, J., Jain, A. and Raghavan, B., "TCP Fast Open". In Proceedings of 7th ACM CoNEXT Conference, December 2011.

[RFC1644] Braden, R., "T/TCP -- TCP Extensions for Transactions Functional Specification", RFC 1644, July 1994.

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

[RFC6013] Simpson, W., "TCP Cookie Transactions (TCPCT)", RFC6013, January 2011.

[SOUDERS11] S. Souders. "Making A Mobile Connection". <http://www.stevesouders.com/blog/2011/09/21/making-a-mobile-connection/>

[THK98] Touch, J., Heidemann, J., Obraczka, K., "Analysis of HTTP



Performance", USC/ISI Research Report 98-463. December 1998.

[BRISCOE12] Briscoe, B., "Some ideas building on draft-ietf-tcpm-fastopen-01", tcpm list,  
[http://www.ietf.org/mail-archive/web/tcpm/current/January 16, 2014msg07192.html](http://www.ietf.org/mail-archive/web/tcpm/current/January%2016,%2014msg07192.html)

[BELSHE12] Belshe, M., "The era of browser preconnect.",  
<http://www.belshe.com/2011/02/10/the-era-of-browser-preconnect/>

## Appendix A. Example Socket API Changes to support TFO

The design rationale is to minimize changes to the socket API and hence applications, in order to reduce the deployment hurdle. The following changes have been implemented in Linux 3.7 or later kernels.

### A.1 MSG\_FASTOPEN flag for sendto() or sendmsg()

MSG\_FASTOPEN marks the attempt to send data in SYN like a combination of connect() and sendto(), by performing an implicit connect() operation. It blocks until the handshake has completed and the data is buffered.

For non-blocking socket it returns the number of bytes buffered and sent in the SYN packet. If the cookie is not available locally, it returns -1 with errno EINPROGRESS, and sends a SYN with TFO cookie request automatically. The caller needs to write the data again when the socket is connected.

It returns the same errno as connect() if the handshake fails.

### A.2 TCP\_FASTOPEN setsockopt() Socket Option

The option enables Fast Open on the listener socket. The option value specifies the PendingFastOpenRequests threshold, i.e., the maximum length of pending SYNs with data payload. Once enabled, the TCP implementation will respond with TFO cookies per request.

Previously accept() returns only after a socket is connected. But for a Fast Open connection, accept() returns upon receiving a SYN with a valid Fast Open cookie and data, and the data is available to be read through, e.g., recvmsg(), read().

## Authors' Addresses

Yuchung Cheng Google, Inc. 1600 Amphitheatre Parkway Mountain View,  
CA 94043, USA EMail: ycheng@google.com

Jerry Chu Google, Inc. 1600 Amphitheatre Parkway Mountain View, CA  
94043, USA EMail: hkchu@google.com

Sivasankar Radhakrishnan Department of Computer Science and  
Engineering University of California, San Diego 9500 Gilman Dr La  
Jolla, CA 92093-0404 EMail: sivasankar@cs.ucsd.edu

Arvind Jain Google, Inc. 1600 Amphitheatre Parkway Mountain View, CA  
94043, USA EMail: arvind@google.com

TCPM Working Group  
Internet-Draft  
Obsoletes: 2861 (if approved)  
Updates: 5681 (if approved)  
Intended status: Standards Track  
Expires: January 2, 2014

G. Fairhurst  
A. Sathiaselalan  
R. Secchi  
University of Aberdeen  
July 01, 2013

Updating TCP to support Rate-Limited Traffic  
draft-ietf-tcpm-newcwv-02

Abstract

This document proposes an update to RFC 5681 to address issues that arise when TCP is used to support traffic that exhibits periods where the sending rate is limited by the application rather than the congestion window. It updates TCP to allow a TCP sender to restart quickly following either an idle or 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.

NOTE: The standards status of this WG document is under review for consideration as either Experimental (EXP) or Proposed Standard (PS). This decision will be made later as the document is finalised.

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 2, 2014.

Copyright Notice

Copyright (c) 2013 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 . . . . .	4
2. Reviewing experience with TCP-CWV . . . . .	5
3. Terminology . . . . .	5
4. An updated TCP response to idle and application-limited periods . . . . .	6
4.1. A method for preserving cwnd during the idle and application-limited periods. . . . .	7
4.2. Initialisation . . . . .	8
4.3. The nonvalidated phase . . . . .	8
4.4. TCP congestion control during the nonvalidated phase . . . . .	8
4.4.1. Response to congestion in the nonvalidated phase . . . . .	9
4.4.2. Adjustment at the end of the nonvalidated phase . . . . .	10
4.4.3. Examples of Implementation . . . . .	11
5. Determining a safe period to preserve cwnd . . . . .	12
6. Security Considerations . . . . .	13
7. IANA Considerations . . . . .	13
8. Acknowledgments . . . . .	13
9. Author Notes . . . . .	14
9.1. Other related work . . . . .	14
9.2. Revision notes . . . . .	16
10. References . . . . .	17
10.1. Normative References . . . . .	17
10.2. Informative References . . . . .	18
Authors' Addresses . . . . .	19

## 1. Introduction

TCP is used to support a range of application behaviours. The TCP congestion window (cwnd) controls the number of unacknowledged packets/bytes that a TCP flow may have in the network at any time, a value known as the FlightSize [RFC5681]. 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). 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. This latter case is called rate-limited. The focus of this document is on the operation of TCP in such an idle or rate-limited case.

Standard TCP [RFC5681] requires the cwnd to be reset to the restart window (RW) when an application becomes idle. [RFC2861] noted that this TCP behaviour was not always observed in current implementations. Recent experiments [Bis08] confirm this to still be the case.

Standard TCP does not impose additional restrictions on the growth of the cwnd when a TCP sender is rate-limited. A rate-limited sender may therefore 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 Congestion Window Validation (CWV). CWV was intended to help reduce cases where TCP accumulated an invalid 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 this in a way that is expected to mitigate the impact on an application that varies its sending rate. The method described applies to both a rate-limited and an idle condition. Section 5 describes the rationale for selecting the safe period to preserve the cwnd.

## 2. Reviewing experience with TCP-CWV

RFC 2861 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 for a rate-limited session where a sender transmitted at a rate less than allowed by cwnd.

RFC 2861 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 some 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 RTCP Restart Window (RW) after this period.

Experience with RFC 2861 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 RFC2681, 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 new terminology is introduced:

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.

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

#### 4. An updated TCP response to idle and application-limited periods

This section proposes an update to the TCP congestion control behaviour during an idle or rate-limited period. The new method permits a TCP sender to preserve the cwnd when an application becomes idle for a period of time (the non-validated period, NVP, see section 5). The period where actual usage is less than allowed by cwnd, is named as the non-validated phase. This method allows an application 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 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. It also reduces the incentive for an application to send data simply to keep transport congestion state. (This is sometimes known as "padding").

The new method does not differentiate between times when the sender has become idle or rate-limited. This is partly a response to recognition that some applications wish to transmit at a rate less than allowed by the sender cwnd, and that it can be hard to make a



distinction between rate-limited and idle behaviour. This 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).

The method is specified in following subsections.

4.1. A method for preserving cwnd during the idle and application-limited periods.

[RFC5681] defines a variable, FlightSize, that indicates the amount of outstanding data in the network. This is assumed to be equal to the value of Pipe calculated based on the pipe algorithm [RFC3517]. In RFC5681 this value is used during loss recovery, whereas in this method a new variable "pipeACK" is introduced to measure the acknowledged size of the pipe, which is used to determine if the sender has validated the cwnd.

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 [RFC3517], 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. 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 sample 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, the pipeACK variable is set to the maximum (undefined) value. This value is used to inhibit entering the nonvalidated phase until the first new measurement of a pipeACK sample.

The method RECOMMENDS that the TCP SACK option [RFC3517] is enabled. 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 higher cwnd following loss.

#### 4.2. Initialisation

A sender starts a TCP connection in the Validated phase and initialises the pipeACK variable to the maximum (undefined) value.

#### 4.3. The nonvalidated phase

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:  $\text{pipeACK} \geq (1/2) * \text{cwnd}$ . 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]). The rule for transitioning to the non-validated phase is specified in section 4.3.
- 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. These mechanisms are specified in section 4.3.

The value  $1/2$  was selected to reduce the effects of variations in the pipeACK variable, and to allow the sender some flexibility in when it sends data.

#### 4.4. TCP congestion control during the nonvalidated phase

A TCP sender MUST enter the non-validated phase when the pipeACK is less than  $(1/2) * \text{cwnd}$ .

A TCP sender that enters the non-validated phase will preserve the cwnd (i.e., this neither grows nor reduces while the sender remains in this phase). If the sender receives an indication of congestion (loss or Explicit Congestion Notification, ECN, mark [RFC3168]) it uses the method described below. The phase is concluded after a fixed period of time (the NVP, as explained in section 4.3.2) or when the sender transmits sufficient data so that  $\text{pipeACK} > (1/2) * \text{cwnd}$  (i.e. it is no longer rate-limited).

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

- o The cwnd is not increased when ACK packets are received in this phase.
- o If the sender receives an indication of congestion while in the non-validated phase (i.e. detects loss, or an ECN mark), the sender MUST exit the non-validated phase (reducing the cwnd as defined in section 4.3.1).
- 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]. (The resulting reduction of cwnd described in section 4.3.2 is appropriate, since any accumulated path history is considered unreliable).
- 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 consume the entire cwnd. However a change to the validated phase will release the sender from constraints on the growth of cwnd, and restore the use of the standard congestion response.)

#### 4.4.1. Response to congestion in the nonvalidated 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 must be selected based on the utilised rate.

A sender that detects a packet-drop or receives an ECN marked packet MUST record the current FlightSize in the variable LossFlightSize and calculate a safe cwnd, by setting it to the value specified in Section 3.2 of [RFC5681].

A TCP sender MUST calculate a safe cwnd to use for loss recovery using the method below:

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

This new cwnd is set to reflect that a nonvalidated cwnd may be much 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:

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

Where, R is the volume of data that was retransmitted during the recovery phase. This follows the method proposed for Jump Start [Liu07]. The inclusion of the term R makes this adjustment more conservative than standard TCP. (This is required, since the sender may have sent more segments than a Standard TCP sender would have done. The additional reduction is beneficial when the LossFlightSize significantly overshoots the available path capacity incurring significant loss, for instance an intense traffic burst following a non-validated period.)

If the sender implements a method that allows it to identify the number of ECN-marked segments within a window that were observed by the receiver, the sender SHOULD use the method above, further reducing R by the number of marked segments.

The sender MUST also re-initialise the pipeACK variable to the maximum (undefined) value. This ensures that standard TCP methods are used immediately after completing loss recovery.

#### 4.4.2. Adjustment at the end of the nonvalidated phase

During the non-validated phase, a sender can produce bursts of data of up to the cwnd in size. While this is no different to standard TCP, it is desirable to control the maximum burst size, e.g. by setting a burst size limit, using a pacing algorithm, or some other method [Hug01].

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]).

(This adjustment ensures that sender responds conservatively at the end of the non-validated phase by reducing the cwnd to better reflect the current rate of the sender. The cwnd update does not take into account FlightSize or pipeACK value because these values only reflect historical data and do not reflect the current sending rate.)

#### 4.4.3. Examples of Implementation

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

XXX This section is work in progress - discussion is welcome to help complete this section XXX

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.

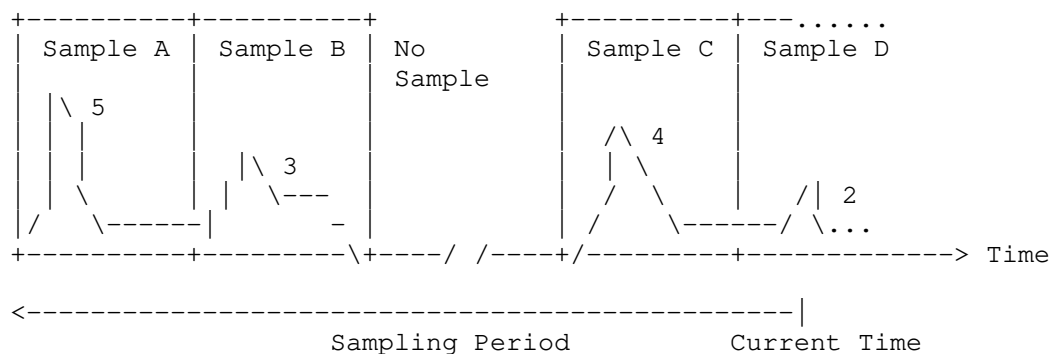


Figure XX: Example of measuring pipeACK samples

Figure XX 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. The current value of the pipeACK variable will be 5, the maximum across all samples.

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.

The NVP period does not necessarily require a new timer to be implemented. An alternative is to record a timestamp when the sender enters the NVP. Each time a sender transmits a new segment, this timestamp may be used to determine if the NVP period has expired. If the period expires, the sender may take into account how many units of the NVP period have passed and make one reduction (as defined in section 4.3.2) for each NVP period.

## 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 non-validated period, 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 modify the network path, disrupting the RTT measurement and changing the capacity available to a TCP connection, however such changes do not often occur within 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

The authors acknowledge the contributions of Dr I Biswas, Mr 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, and Joe Touch. 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

### 9.1. Other related work

There are several issues to be discussed more widely:

- o Should the method explicitly state a procedure for limiting burstiness or pacing?

This is often regarded as good practice, but is not presently a formal part of TCP. draft-hughes-restart-00.txt provides some discussion of this topic.

- o There are potential interactions with the Experimental update in [RFC6928] 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 attended 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 segments are all ACK'ed 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 results in an unusually low cwnd ( $1/2$  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 RTO), 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 Nonvalidated phase. This value better reflects the capacity that the flow has utilised in the network path.

## 9.2. 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 [RFC3517] to be enabled - 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.

## 10. References

### 10.1. Normative References

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

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2861] Handley, M., Padhye, J., and S. Floyd, "TCP Congestion Window Validation", RFC 2861, June 2000.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.
- [RFC3517] Blanton, E., Allman, M., Fall, K., and L. Wang, "A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP", RFC 3517, April 2003.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.
- [RFC6928] Chu, J., Dukkkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, April 2013.

## 10.2. Informative References

- [Bis08] Biswas and 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, Sathiaselvan, Secchi, and Fairhurst, "Analysing TCP for Bursty Traffic, Int'l J. of Communications, Network and System Sciences, 7(3)", June 2010.
- [Bis11] Biswas, "PhD Thesis, Internet congestion control for variable rate TCP traffic, School of Engineering, University of Aberdeen", June 2011.
- [Fair12] Fairhurst, Biswas, Biswas, and Biswas, "Enhancing TCP Performance to support Variable-Rate Traffic, 2nd Capacity Sharing Workshop, ACM CoNEXT, Nice, France, 10th December 2012.", June 2008.
- [Hug01] Hughes, Touch, and Heidemann, "&#8730;&#8730;Issues in TCP Slow-Start Restart After Idle (Work-in-Progress)", December 2001.

[Liu07] Liu, Allman, Jiny, and Wang, "Congestion Control without a Startup Phase, 5th International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet), Los Angeles, California, USA", February 2007.

#### 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 Sathiaselan  
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: August 20, 2013

P. Hurtig

A. Brunstrom

Karlstad University

A. Petlund

Simula Research Laboratory AS

M. Welzl

University of Oslo

February 16, 2013

TCP and SCTP RTO Restart  
draft-ietf-tcpm-rtorestart-00

Abstract

This document describes a modified algorithm for managing the TCP and SCTP retransmission timers that provides faster loss recovery when a connection's amount of outstanding data is small. The modification allows the transport to restart its retransmission timer more aggressively 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 August 20, 2013.

Copyright Notice

Copyright (c) 2013 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 uses two mechanisms to detect segment loss. First, if a segment is not acknowledged within a certain amount of time, a retransmission timeout (RTO) occurs, and the segment is retransmitted [RFC6298]. 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 segments are not mistaken as lost. Second, when a sender receives duplicate acknowledgments, the fast retransmit algorithm infers segment loss and triggers a retransmission [RFC5681]. Duplicate acknowledgments are generated by a receiver when out-of-order segments arrive. As both segment loss and segment reordering cause out-of-order arrival, fast retransmit waits for three duplicate acknowledgments before considering the segment as lost. In some situations, however, the number of outstanding segments is not enough to trigger three duplicate acknowledgments, and the sender must rely on lengthy RTOs for loss recovery.

The amount of outstanding segments 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 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, which is frequent if the total amount of data is small (e.g. web traffic).

The first two situations can occur for any flow, as external factors at the network and/or host level cause them. The third situation primarily affects flows that are short or have a low transmission rate. Typical examples of applications that produce short flows are



web servers. [RJ10] shows that 70% of all web objects, found at the top 500 sites, are too small for fast retransmit to work. [BPS98] shows that about 56% of all retransmissions sent by a busy web server are sent after RTO expiry. While the experiments were not conducted using SACK [RFC2018], only 4% of the RTO-based retransmissions could have been avoided. Applications have a low transmission rate when data is sent in response to actions, or as a reaction to real life events. Typical examples of such applications are stock trading systems, remote computer operations and online games. What is special about this class of applications is that they are time-dependant, and extra latency can reduce the application service level [P09]. Although such applications may represent a small amount of data sent on the network, a considerable number of flows have such properties and the importance of low latency is high.

The RTO restart approach outlined in this document makes the RTO slightly more aggressive when the number of outstanding segments is small, in an attempt to enable faster loss recovery for all segments while being robust to reordering. While it still conforms to the requirement in [RFC6298] that segments must not be retransmitted earlier than RTO seconds after their original transmission, it could increase the chance for a spurious timeout, which could degrade performance when the congestion window (cwnd) is large -- for example, when an application sends enough data to reach a cwnd covering 100 segments and then stops. The likelihood and potential impact of this problem as well as possible mitigation strategies are currently under investigation.

While this document focuses on TCP, the described changes are also valid for the Stream Control Transmission Protocol (SCTP) [RFC4960] which has similar loss recovery and congestion control algorithms.

### 1.1. Requirements Language

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

## 2. RTO Restart Overview

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. However, by restarting the timer on each incoming acknowledgment, retransmissions are not typically triggered RTO seconds after their

previous transmission but rather RTO seconds after the last ACK arrived. The duration of this extra delay depends on several factors but is in most cases approximately one RTT. Hence, in most situations the time before a retransmission is triggered is equal to "RTO + RTT".

The extra delay can be significant, especially for applications that use a lower RTT<sub>min</sub> than the standard of 1 second and/or in environments with high RTTs, e.g. mobile networks. The restart approach 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 [RFC1122], which restarts the RTO timer at the sender. The RTO restart is performed 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 is "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 delayed ACK timer will also be included in the effective loss detection time.

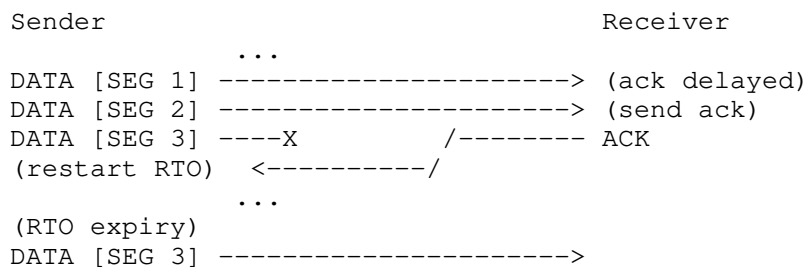


Figure 1: RTO restart example

During normal TCP bulk transfer the current RTO restart approach is not a problem. Actually, 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, and the current approach can therefore be 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."

There are only a few 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. Furthermore, for connections that do not utilize enough capacity to enable fast retransmit, RTO is the only choice. The time needed for loss detection in such scenarios can become a serious performance bottleneck.

### 3. RTO Restart Algorithm

To enable faster loss recovery for connections that are unable to use fast retransmit, an alternative RTO restart can be used. 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 approach makes the RTO more aggressive than the standardized approach in [RFC6298] but still conforms to the requirement in [RFC6298] that segments must not be retransmitted earlier than RTO seconds after their original transmission.

This document specifies the following update of step 5.3 in Section 5 of [RFC6298] (and a similar update in Section 6.3.2 of [RFC4960] for SCTP):

When an ACK is received that acknowledges new data:

- (1) Set T\_earliest = 0.
- (2) If the following two conditions hold:
  - (a) The number of outstanding segments is less than four.
  - (b) There is no unsent data ready for transmission or the receiver's advertised window does not permit transmission.

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 "RTO - T\_earliest" seconds (for the current value of RTO).

The update requires TCP implementations to track the time elapsed since the transmission of the earliest outstanding segment (T\_earliest). As the alternative restart is used only when the number of outstanding segments is less than four only four segments need to be tracked. Furthermore, some implementations of TCP (e.g.

Linux TCP) already track the transmission times of all segments.

#### 4. Discussion

The currently standardized algorithm has been shown to add at least one RTT to the loss recovery process in TCP [LS00] and SCTP [HB08][PBP09]. Applications that have strict timing requirements (e.g. telephony signaling and gaming) rather than throughput requirements may want to use a lower RTT<sub>min</sub> than the standard of 1 second [RFC4166]. For such applications the modified restart approach could be important as the RTT and also the delayed ACK timer of receivers will be large components of the effective loss recovery time. Measurements in [HB08] have shown that the total transfer time of a lost segment (including the original transmission time and the loss recovery time) can be reduced with up to 35% using the suggested approach. These results match those presented in [PGH06][PBP09], where the modified restart approach is shown to significantly reduce retransmission latency.

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 duplicate acknowledgments. By transmitting new segments, the sender attempts to generate additional duplicate acknowledgments to enable fast retransmit. However, limited transmit does not help if no previously unsent data is ready for transmission or if the receiver is out of buffer space. [RFC5827] specifies an early retransmit algorithm to enable fast loss recovery in such situations. By dynamically lowering the amount of duplicate acknowledgments needed for fast retransmit (dupthresh), based on the number of outstanding segments, a smaller number of duplicate acknowledgments are 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, the early retransmit cannot help. Third, if the network path reorders segments, the algorithm might cause more unnecessary retransmissions than fast retransmit.

Following the fast retransmit mechanism standardized in [RFC5681] this draft assumes a value of 3 for dupthresh. However, by considering a dynamic value for dupthresh a tighter integration with early retransmit (or other experimental algorithms) could also be possible.

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. The Probe timeout (PTO) is at least 2 RTTs, and only scheduled in case the RTO is farther than the PTO. A spurious PTO is less risky than a spurious RTO, as it would not have the same negative effects (clearing the scoreboard and restarting with slow-start). In contrast, RTO restart is trying to make the RTO more appropriate in cases where there is no need to be overly cautious.

TLP could kick in in situations where RTO restart does not apply, and it could overrule (yielding a similar general behavior, but with a lower timeout) RTO restart in cases where the number of outstanding segments is smaller than 4 and no new segments are available for transmission. The shorter RTO from RTO restart also reduces the probability that TLP is activated because PTO might be farther than RTO.

## 5. IANA Considerations

This memo includes no request to IANA.

## 6. Security Considerations

This document discusses a change in how to set the retransmission timer's value when restarted. This change does not raise any new security issues with TCP or SCTP.

## 7. References

### 7.1. Normative References

- [RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.
- [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.

- [RFC4166] Coene, L. and J. Pastor-Balbas, "Telephony Signalling Transport over Stream Control Transmission Protocol (SCTP) Applicability Statement", RFC 4166, February 2006.
- [RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, 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.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.

## 7.2. Informative References

- [BPS98] Balakrishnan, H., Padmanabhan, V., Seshan, S., Stemm, M., and R. Katz, "TCP Behavior of a Busy Web Server: Analysis and Improvements", Proc. IEEE INFOCOM Conf., March 1998.
- [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.
- [HB08] Hurtig, P. and A. Brunstrom, "SCTP: designed for timely message delivery?", Springer Telecommunication Systems, May 2010.
- [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.
- [RJ10] Ramachandran, S., "Web metrics: Size and number of  
resources", Google [http://code.google.com/speed/articles/  
web-metrics.html](http://code.google.com/speed/articles/web-metrics.html), May 2010.
- [TLP] Dukkkipati, N., Cardwell, N., Cheng, Y., and M. Mathis,  
"TCP Loss Probe (TLP): An Algorithm for Fast Recovery of  
Tail Losses", draft-dukkkipati-tcpm-tcp-loss-probe-00.txt  
(work in progress), July 2012.

## 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](mailto: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](mailto: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](mailto:michawe@ifi.uio.no)





TCP Maintenance and Minor Extensions (tcpm)  
Internet-Draft  
Intended status: Experimental  
Expires: December 22, 2013

M. Kuehlewind, Ed.  
University of Stuttgart  
R. Scheffenegger  
NetApp, Inc.  
June 20, 2013

More Accurate ECN Feedback in TCP  
draft-kuehlewind-tcpm-accurate-ecn-02

Abstract

Explicit Congestion Notification (ECN) is an IP/TCP mechanism where network nodes can mark IP packets instead of dropping them to indicate congestion to the end-points. ECN-capable receivers will feedback this information to the sender. ECN is specified for TCP in such a way that only one feedback signal can be transmitted per Round-Trip Time (RTT). Recently, new TCP mechanisms like ConEx or DCTCP need more accurate ECN feedback information in the case where more than one marking is received in one RTT. This document specifies a different scheme for the ECN feedback in the TCP header to provide more than one feedback signal per RTT. Furthermore this document specifies a re-use of the Urgent Pointer in the TCP header if the URG flag is not set to increase the robustness of the proposed ECN feedback scheme.

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 22, 2013.

## Copyright Notice

Copyright (c) 2013 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
1.1. Overview ECN and ECN Nonce in IP/TCP . . . . .	3
1.2. Re-Use of the Urgent field in TCP . . . . .	4
1.3. Requirements Language . . . . .	4
2. More Accurate ECN Feedback . . . . .	5
2.1. Negotiation during the TCP handshake . . . . .	5
2.2. Feedback Coding . . . . .	7
2.2.1. Codepoint Coding of the more Accurate ECN (ACE) field . . . . .	7
2.2.2. Use with ECN Nonce . . . . .	8
2.2.3. Auxiliary data in the Urgent Pointer field . . . . .	8
2.3. More Accurate ECN TCP Receiver . . . . .	9
2.4. More Accurate ECN TCP Sender . . . . .	11
3. Acknowledgements . . . . .	12
4. IANA Considerations . . . . .	12
5. Security Considerations . . . . .	12
6. References . . . . .	12
6.1. Normative References . . . . .	13
6.2. Informative References . . . . .	13
Authors' Addresses . . . . .	14

## 1. Introduction

Explicit Congestion Notification (ECN) [RFC3168] is an IP/TCP mechanism where network nodes can mark IP packets instead of dropping them to indicate congestion to the end-points. ECN-capable receivers will feedback this information to the sender. ECN is specified for TCP in such a way that only one feedback signal can be transmitted per Round-Trip Time (RTT). Recently, proposed mechanisms like Congestion Exposure (ConEx) or DCTCP [Ali10] need more accurate ECN feedback information in case when more than one marking is received in one RTT.

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

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

#### 1.1. Overview ECN and ECN Nonce in IP/TCP

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

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

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

byte 13 in the TCP header as the Nonce Sum (NS) bit. With ECN-Nonce a nonce sum is maintain that counts the occurrence of ECT(1) packets.

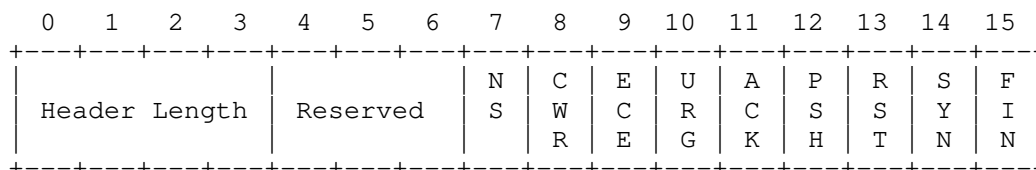


Figure 1: The (post-ECN Nonce) definition of the TCP header flags

### 1.2. Re-Use of the Urgent field in TCP

RFC0793 specified a mechanism to indicate "urgent data" to a receiver. However, this mechanism is rarely used, and RFC6093 argues to deprecate the use of the mechanism. Furthermore, the content of the Urgent Pointer was always defined to be valid only, when the URG TCP header flag is set. The position of the Urgent Pointer field as well as the URG flag are displayed in Figure 2.

In this document the Urgent Pointer field is defined to be (re)usable for auxiliary data if the URG flag is not set. Note that as the contents of this field were previously undefined when the URG bit is not set, a new mechanism using these bits SHOULD not rely on the correct delivery. Further below in this document a new usage for four bits of the Urgent Pointer counter is defined.

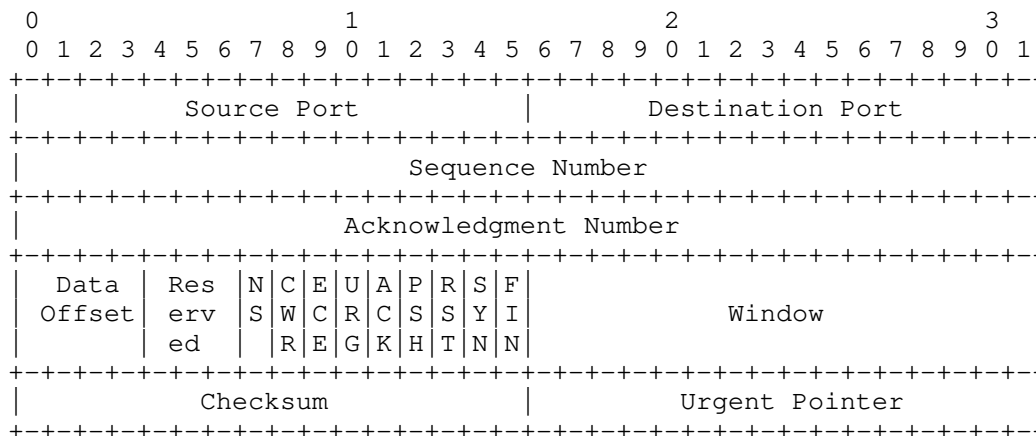


Figure 2: TCP Header Format showing the 16 bit Urgent pointer

### 1.3. Requirements Language

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

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

The ECN field in the IP header:

CE:       the Congestion Experienced codepoint, and  
ECT(0):   the first ECN-Capable Transport codepoint, and  
ECT(1):   the second ECN-Capable Transport codepoint.

The ECN flags in the TCP header:

CWR:       the Congestion Window Reduced flag,  
ECE:       the ECN-Echo flag, and  
NS:        ECN Nonce Sum.

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

## 2. More Accurate ECN Feedback

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

### 2.1. Negotiation during the TCP handshake

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

classic ECN implicit if the receiver does not support the more accurate ECN feedback scheme.

A responding host (host B) MUST return a <SYN,ACK> with flags CWR=1 and ECE=0. The NS flag may be either 0 or 1, as described below. The responding host MUST NOT set this combination of flags unless the preceding <SYN> has already requested support for accurate ECN feedback as above.

These handshakes including the fallback when the receiver only support the classic ECN or ECN-Nonce are summarized in Table 1 below. X indicates that NS can be either 0 or 1 depending on whether congestion had been experienced (see below). The handshake indicating any of the other flavors of ECN are also shown for comparison. To compress the width of the table, the headings of the first four columns have been severely abbreviated, as following:

Ac: \*Ac\*curate ECN Feedback

N: ECN-\*N\*once (RFC3540)

E: \*E\*CN (RFC3168)

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

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

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

The responding host (B) MAY set the NS bit to 1 to indicate a congestion feedback for the <SYN> packet. Otherwise the receiver (B) MUST reply to the sender with NS=0. The addition of ECN to TCP <SYN,ACK> packets is discussed and specified as experimental in [RFC5562] where the addition of ECN to the SYN packet is optionally described. The security implications when using this option are not further discussed here. Only if the initial <SYN> from client A is marked CE, the server B SHOULD set the NS flag to 1 to indicate the congestion immediately, instead of delaying the signal to the first

acknowledgment when the actual data transmission has started. So, server B MAY set the alternative TCP header flags in its <SYN,ACK>: NS=1, CWR=1 and ECE=0.

Recall that, if the <SYN,ACK> reflects the same flag settings as the preceding <SYN> (because there may exist broken TCP implementations that behave this way), [RFC3168] specifies that the whole connection MUST revert to Not-ECT.

## 2.2. Feedback Coding

This section proposes the new coding to provide a more accurate ECN feedback by use of the two ECN TCP bits (ECE/CWR) as well as the TCP NS bit and the optional use of the Urgent Pointer if the URG flag is not set. This coding MUST only be used if the more accurate ECN Feedback has been negotiated successfully in the TCP handshake.

### 2.2.1. Codepoint Coding of the more Accurate ECN (ACE) field

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

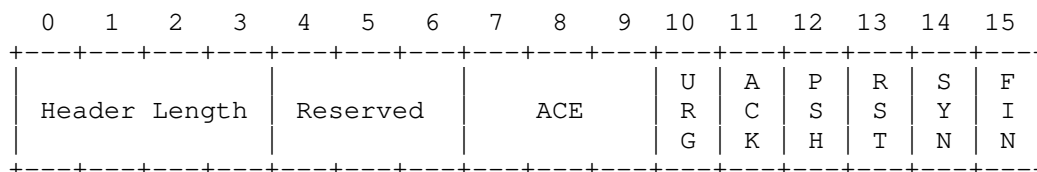
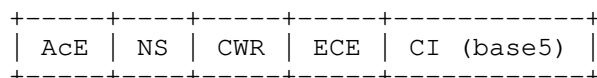


Figure 3: Definition of the ACE field within bytes 13 and 14 of the TCP Header (when SYN=0).

The 8 possible codepoints are shown below. Five of them are used to encode a "congestion indication" (CI) counter. The other three codepoints are defined in the next section to be used for an integrity check based on ECN-Nonce. The CI counter maintains the number of CE marks observed at the receiver (see Section 2.3).





0	0	0	0	0
1	0	0	1	1
2	0	1	0	2
3	0	1	1	3
4	1	0	0	4
5	1	0	1	–
6	1	1	0	–
7	1	1	1	–

Table 2: Codepoint assignment for accurate ECN feedback

Also note that, whenever the SYN flag of a TCP segment is set (including when the ACK flag is also set), the NS, CWR and ECE flags (i.e. the ACE field of the <SYN,ACK>) MUST NOT be interpreted as the 3-bit codepoint, which is only used in non-SYN packets.

#### 2.2.2. Use with ECN Nonce

In ECN Nonce, by comparing the number of incoming ECT(1) notifications with the actual number of packets that were transmitted with an ECT(1) mark as well as the sum of the sender's two internal counters, the sender can probabilistically detect a receiver that sends false marks or suppresses accurate ECN feedback, or a path that does not properly support ECN.

If an ECT(1) mark is received, an ETC(1) counter (E1) is incremented. The receiver has to convey that updated information to the sender with the next possible ACK using the three remaining codepoints as shown in Table 3.

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

Table 3: Codepoint assignment for accurate ECN feedback and ECN Nonce

#### 2.2.3. Auxiliary data in the Urgent Pointer field

In order to provide improved resiliency against loss or ACK thinning, the limited number of bits in the existing TCP flags field is insufficient. At the same time is it not necessary to deliver higher order bits with every returned segment, or even reliably at all. Therefore four bits of the reused Urgent Pointer field are defined as the "Top ACE" field of the more accurate ECN feedback, as indicated in Figure 4. This field carries the top (binary) counter value, if the according codepoint does signal the feedback of a counter. Therefore, we call this field "Top ACE".

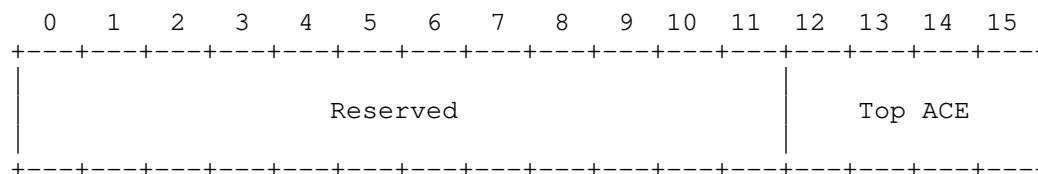


Figure 4: The (post-ECN Nonce) definition of the TCP header flags

As 5 codepoints are set aside to provide reasonable resiliency under typical marking and loss regimes, the combination between the 4 bits in the Top ACE field and the 5 codepoints in the ACE field allow for up to  $16 \times 5 = 80$  congestion indications to be unambiguously signaled back to the sender, even with more extreme levels of CE marking, or return ACK loss.

A combination with the 3 remaining codepoints (e.g. to signal a counter for the number of observed ECT1 packets) and this field allows for up to  $16 \times 3 = 48$  distinct indications.

The reserved bits SHOULD be set to zero, and MUST NOT be interpreted when evaluating the combination of the "Top ACE": "ACE" fields. Also, when the URG flag is set, the entire Urgent Pointer MUST NOT be interpreted to carry significance for the Accurate ECN feedback.

### 2.3. More Accurate ECN TCP Receiver

This section describes the receiver-side action to signal the accurate ECN feedback back to the sender. To select the correct codepoint for each ACK, the receiver will need to maintain a congestion indication (CI) counter of how many CE marking have been seen during a connection and an ECT(1) counter (E1) that is incremented on the reception of a ECT(1) marked packet.

Thus for each incoming segment with a CE marking, the receiver will increase CI by 1. With each ACK the receiver will calculate CI modulo 5 and set the respective codepoint in the ACE field (see Table 2). In addition, the receiver calculates CI divided by 5 and may set

the "Top ACE" field to this value, provided the URG flag is not set in the segment. To avoid counter wrap around in a high congestion situation, the receiver MAY switch from a delayed ACK behavior to send ACKs immediately after the data packet reception if needed.

By default an accurate ECN receiver SHOULD echo the current value of the CI counter, using one of the codepoints encoding the CI counter. Whenever a CE marked segment is received and thus the value of the CI is changed, the receiver MUST echo the then current CI value in the next ACK sent. The receiver MAY use the "Top ACE" field in addition if the URG flag is not set.

The requirement to signal an updated CI value immediately with the next ACK may conflict with a delayed ACK ratios larger than two, when using the available number of codepoints only when "Top ACE" can not be used. A receiver MAY change the ACK'ing rate such that a sufficient rate of feedback signals can be sent. However, in the combination with the redefined Urgent Pointer field, no change in the ACK rate should be required.

Whenever a ECT(1) marked packet arrives, the receiver SHOULD signal the current value of the E1 counter (modulo 3) in the next ACK using the respective codepoint. If a CE mark was received before sending the next ACK (e.g. delayed ACKs) sending the current CI value update MUST take precedence. Further resilience against lost ACKs MAY be provided by inserting the high order bits of the E1 counter (E1 divided by 3) into the Top ACE field.

For the implementation it is suggested to maintain two counters so to avoid costly division operations while processing the header information for the ACK. The first counter can be mapped directly into the ACE field. A wrap by the count of 5 is implemented as a single conditional check, and when that happens, a secondary, high-order counter is increased once. This secondary counter can then be mapped directly into the Top ACE field.

```
if (CE) {
    if (CIcnt == 5) {
        CIcnt = 0
        CIovf += 1
    } else
        CIcnt += 1
}

ACE      = CIcnt;
TopACE   = CIovf;
```

Figure 5: Implementation example

#### 2.4. More Accurate ECN TCP Sender

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

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

On the arrival of every ACK, the sender updates the local CI.r value to the signaled CI value in the ACK as conveyed by the combination of the ACE and "Top ACE" fields in the Urgent Pointer if the URG flag is not set.

If the URG flag is set and thus the "Top ACE" field in the Urgent Pointer field is not available, the sender calculates a value D as the difference between value of the ACE field and the current CI.r value modulo 5. D is assumed to be the number of CE marked packets that arrived at the receiver since it sent the previously received ACK. Thus the local counter CI.r must be increased by D.

As only a limited number of E1 codepoints exist and the receiver might not acknowledge every single data packet immediately (e.g. delayed ACKs), a sender SHOULD NOT mark more than  $1/m$  of the packets with ECT(1), where m is the ACK ratio (e.g. 50% when every second data packet triggers an ACK). This constraint can be lifted when a sender determines, that the auxiliary data is available (the Top ACE field of an ACK with an E1 codepoint is increasing with the number of sent ECT(1) segments). A sender SHOULD send no more than 3 consecutive packets marked with ECT(1), as long as the validity of the auxiliary data in the Top ACE field has not been confirmed.

### 3. Acknowledgements

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

### 4. IANA Considerations

This memo includes a request to IANA, to set up a new registry. This registry redefines the use of the 16 bit "Urgent Pointer" while the URG flag is not set. 4 of those bits ("Top ACE") are defined within this document to be interpreted in conjunction with another field ("ACE"), overwriting three of the existing TCP flags into a single field.

### 5. Security Considerations

TBD

ACK loss

This scheme sends each codepoint only once. In the worst case at least one, and often two or more consecutive ACKs can be dropped without losing congestion information, even when the auxiliary data field in the former Urgent Pointer field is unavailable (i.e. the URG flag is set, or a middlebox clears its contents).

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

ECN Nonce

In the proposed scheme there are three more codepoints available that could be used for an integrity check like ECN Nonce. If ECN nonce would be implemented as proposed in Section 2.2.2, even more information would be provided for ECN Nonce than in the original specification.

A delayed ACK ratio of two can be sustained indefinitely without reverting to auxiliary information, even during heavy congestion, but not during excessive ECT(1) marking, which is under the control of the sender. A higher ACK ratio can be sustained when congestion is low, and the auxiliary data is available.

### 6. References

## 6.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.
- [RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, June 2003.

## 6.2. Informative References

- [Ali10] Alizadeh, M., Greenberg, A., Maltz, D., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and M. Sridharan, "DCTCP: Efficient Packet Transport for the Commoditized Data Center", Jan 2010.
- [I-D.briscoe-tsvwg-re-ecn-tcp] Briscoe, B., Jacquet, A., Moncaster, T., and A. Smith, "Re-ECN: Adding Accountability for Causing Congestion to TCP/IP", draft-briscoe-tsvwg-re-ecn-tcp-09 (work in progress), October 2010.
- [RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", RFC 5562, June 2009.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.
- [RFC5690] Floyd, S., Arcia, A., Ros, D., and J. Iyengar, "Adding Acknowledgement Congestion Control to TCP", RFC 5690, February 2010.
- [draft-kuehlewind-tcpm-accurate-ecn-option] Kuehlewind, M. and R. Scheffenegger, "Accurate ECN Feedback Option in TCP", draft-kuehlewind-tcpm-accurate-ecn-option-01 (work in progress), Jul 2012.

Authors' Addresses

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

Email: [mirja.kuehlewind@ikr.uni-stuttgart.de](mailto:mirja.kuehlewind@ikr.uni-stuttgart.de)

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

Phone: +43 1 3676811 3146  
Email: [rs@netapp.com](mailto:rs@netapp.com)

TCPM Working Group  
Internet-Draft  
Intended status: Experimental  
Expires: January 03, 2014

M. Kuehlewind  
University of Stuttgart  
B. Trammell  
ETH Zurich  
July 02, 2013

A Mechanism for ECN Path Probing and Fallback  
draft-kuehlewind-tcpm-ecn-fallback-00.txt

Abstract

Explicit Congestion Notification (ECN) is a TCP/IP extension that is widely implemented but hardly used due to the perceived unusability of ECN on many paths through the Internet caused by ECN-ignorant routers and middleboxes. This document specifies an ECN probing and fall-back mechanism in case ECN has been successfully negotiated between two connection endpoints, but might not be usable on the path.

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 03, 2014.

Copyright Notice

Copyright (c) 2013 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

The deployment of Explicit Congestion Notification (ECN) [RFC3168] and AQM would arguably improve end-to-end performance in the Internet, by providing a congestion signal to the transport layer without relying on queue tail drop and packet loss. However, though ECN has been standardized since 2000, implementation and deployment have lagged significantly, in part due to the perceived unusability of ECN on many paths through the Internet caused by ECN-ignorant routers and middleboxes.

Recent research by the authors [KuNeTr13] has shown accelerating deployment of ECN-capable servers in the Internet, due to the deployment of TCP stacks for which ECN is enabled by default. In addition, ECN is usable end-to-end on the vast majority of paths measured in this study: that is, a Congestion Experienced mark will cause a ECN Echo on the associated ACK. However, there still exist a non-negligible number of paths on which a successfully negotiated usage of ECN will not result in a connection on which congestion will be correctly echoed, or worse, leads to the loss of packets with CE or ECE set.

This document presents an experimental, in-band, runtime method for determining the usability of ECN by a given traffic flow, based on the active measurement method described in [KuNeTr13]. If ECN is successfully negotiated but found by this method to be unusable, it can be disabled on subsequent packets in the flow in order to avoid connectivity problems caused by ECN-unusability on the path.

## 2. ECN Path Capability Probing

A TCP sender can determine whether or not its path to the receiver is usable for ECN using the procedure detailed below.

1. The sender attempts to negotiate ECN usage as per Section 6.1.1 of [RFC3168]. If ECN is not successfully negotiated, the procedure ends, and ECN is not used for the duration of the connection.
2. The sender disables the normal usage of ECN for the duration of the procedure, as the ECN codepoints are used for path probing. This means all segments are sent with the Non-ECN-Capable codepoint during this procedure unless otherwise stated. Moreover, the sender will only take loss as a congestion signal

and will not react with window reductions to the ECN-Echo (ECE) feedback signal from the receiver during this procedure.

3. The sender sets the Non-ECN-Capable codepoint in the IP header until it has completed sending the first N segments, where N is the size of the initial congestion window. Loss is used to discover congestion for these segments.
4. The next three segments sent consist of the "CE probe": three segments are sent with the Congestion Experienced codepoint set.
5. If all three of the CE probe segments are lost and must be retransmitted, the path is deemed not ECN-usable and the sender falls back as in Section 3.
6. If the ECE flag is not set on the ACK segment(s) sent by the receiver acknowledging the CE probe segments, the path may or may not be usable, as that there might be middleboxes/gateways that (arguably correctly) clear CE on segments from end hosts, because they assume that congestion can not have occurred up to this point on the path. In this case, the sender may continue using ECN, because while it may not work for detecting congestion, the use of ECN does not negatively affect connectivity. Note that this behavior can be more precisely detected using ECN Nonce [RFC3540].
7. While the sender does not reduce the congestion window for the ECE segment(s) sent for the CE probe segments, it does set CWR on the subsequent segment sent.
8. If no fallback has occurred by the time the ACK of the final CE probe segment is received, the path is deemed ECN usable, and the sender ends the probing procedure and proceeds to use ECN normally as in [RFC3168].

As the probing begins after all the segments in the initial congestion window have been sent, it requires more than an initial congestion window plus 6 segments (3 CE probe + 3 duplicated ACKs) of available data to send. As this information is only available at the higher layer, a configuration option per connection should be provided to dis/enable ECN as well as ECN probing. Otherwise, ECN should not be enabled for such short flows while using this procedure.

### 3. ECN Fallback

If ECN is found to be unusable on a given flow by path capability probing as in Section 2 above, the sender simply stops setting any

ECN-Capable-Transport codepoint on subsequent packets in the flow. The receiver MUST, however, still set ECE on any ACK for a packet with CE set. Note that this behavior is consistent with section 6.1.1 of [RFC3168].

A sender may keep a cache of paths found to be unusable for ECN and disable ECN for subsequent connections on a per-destination basis. In this case, the receiver should periodically (i.e., on the order of hours or days) expire these cache entries to cause re-probing to occur in order to account for routing changes in the network.

[EDITOR'S NOTE: what to do on RTO?]

#### 4. Discussion

[EDITOR'S NOTE: need to think about how this would interact with conex; an analysis comparing the delay caused by path probing as opposed to the delay caused by ECN failure would be interesting.]

[EDITOR'S NOTE: initial implementation results go here?..]

#### 5. Security Considerations

[FIXME: we'll have to explore attacks against this mechanism which could affect network or connection stability, so the following is wrong...]

This document has no security considerations.

#### 6. IANA Considerations

This document has no IANA considerations.

#### 7. References

##### 7.1. Normative References

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

##### 7.2. Informative References

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

[KuNeTr13]

Kuehlewind, M., Neuner, S., and B. Trammell, "On the state of ECN and TCP Options on the Internet", Mar 2013.

(In LNCS 7799, Proceedings of PAM 2013, Hong Kong)

Authors' Addresses

Mirja Kuehlewind  
University of Stuttgart  
Pfaffenwaldring 47  
70569 Stuttgart  
Germany

Email: [mirja.kuehlewind@ikr.uni-stuttgart.de](mailto:mirja.kuehlewind@ikr.uni-stuttgart.de)

Brian Trammell  
Swiss Federal Institute of Technology Zurich  
Gloriastrasse 35  
8092 Zurich  
Switzerland

Phone: +41 44 632 70 13  
Email: [trammell@tik.ee.ethz.ch](mailto:trammell@tik.ee.ethz.ch)

TCP Maintenance and Minor Extensions  
(TCPM) WG  
Internet-Draft  
Obsoletes: 4614 (if approved)  
Intended status: Informational  
Expires: November 22, 2013

M. Duke  
Boeing Phantom Works  
R. Braden  
ISI  
W. Eddy  
MTI Systems  
E. Blanton  
Purdue University  
A. Zimmermann  
NetApp, Inc.  
May 21, 2013

A Roadmap for Transmission Control Protocol (TCP) Specification  
Documents  
draft-zimmermann-tcpm-tcp-rfc4614bis-02

Abstract

This document contains a "roadmap" to the Requests for Comments (RFC) documents relating to the Internet's Transmission Control Protocol (TCP). This roadmap provides a brief summary of the documents defining TCP and various TCP extensions that have accumulated in the RFC series. This serves as a guide and quick reference for both TCP implementers and other parties who desire information contained in the TCP-related RFCs.

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 November 22, 2013.

Copyright Notice

Copyright (c) 2013 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. Basic Functionality . . . . .	4
3. Recommended Enhancements . . . . .	7
3.1. Fundamental Changes . . . . .	7
3.2. Congestion Control and Loss Recovery Extensions . . . . .	9
3.3. SACK-Based Loss Recovery and Congestion Control . . . . .	10
3.4. Detection and Prevention of Spurious Retransmissions . . . . .	11
3.5. Path MTU Discovery . . . . .	12
3.6. Header Compression . . . . .	13
3.7. Defending Spoofing and Flooding Attacks . . . . .	14
4. Experimental Extensions . . . . .	15
4.1. Architectural Guidelines . . . . .	16
4.2. Congestion Control and Loss Recovery Extensions . . . . .	16
4.3. Detection and Prevention of Spurious Retransmissions . . . . .	18
4.4. Multipath TCP . . . . .	19
5. Historic Extensions . . . . .	20
6. Support Documents . . . . .	22
6.1. Foundational Works . . . . .	23
6.2. Architectural Guidelines . . . . .	24
6.3. Difficult Network Environments . . . . .	25
6.4. Guidance for Developing, Analyzing, and Evaluating TCP . . . . .	28
6.5. Implementation Advice . . . . .	29
6.6. Management Information Bases . . . . .	31
6.7. Tools and Tutorials . . . . .	32
6.8. Case Studies . . . . .	33
7. Undocumented TCP Features . . . . .	34
8. Security Considerations . . . . .	36
9. IANA Considerations . . . . .	36
10. Acknowledgments . . . . .	36
11. References . . . . .	37
11.1. Normative References . . . . .	37
11.2. Informative References . . . . .	46
Authors' Addresses . . . . .	47

## 1. Introduction

A correct and efficient implementation of the Transmission Control Protocol (TCP) is a critical part of the software of most Internet hosts. As TCP has evolved over the years, many distinct documents have become part of the accepted standard for TCP. At the same time, a large number of more experimental modifications to TCP have also been published in the RFC series, along with informational notes, case studies, and other advice.

As an introduction to newcomers and an attempt to organize the plethora of information for old hands, this document contains a "roadmap" to the TCP-related RFCs. It provides a brief summary of the RFC documents that define TCP. This should provide guidance to implementers on the relevance and significance of the standards-track extensions, informational notes, and best current practices that relate to TCP.

This document is not an update of RFC 1122 and is not a rigorous standard for what needs to be implemented in TCP. This document is merely an informational roadmap that captures, organizes, and summarizes most of the RFC documents that a TCP implementer, experimenter, or student should be aware of. Particular comments or broad categorizations that this document makes about individual mechanisms and behaviors are not to be taken as definitive, nor should the content of this document alone influence implementation decisions.

This roadmap includes a brief description of the contents of each TCP-related RFC. In some cases, we simply supply the abstract or a key summary sentence from the text as a terse description. In addition, a letter code after an RFC number indicates its category in the RFC series (see BCP 9 [RFC2026] for explanation of these categories):

S - Standards Track (Proposed Standard, Draft Standard, or Internet Standard)

E - Experimental

I - Informational

H - Historic

B - Best Current Practice

U - Unknown (not formally defined)

Note that the category of an RFC does not necessarily reflect its current relevance. For instance, RFC 5681 is nearly universally deployed although it is only a Draft Standard. Similarly, some Informational RFCs contain significant technical proposals for changing TCP.

Finally, if an error in the technical content has been found after publication of an RFC, this fact is indicated by the term "(Errata)" in the headline of the RFC's description. The contents of the errata can be found at the RFC editor home page [Errata].

This roadmap is divided into four main sections. Section 2 lists the RFCs that describe absolutely required TCP behaviors for proper functioning and interoperability. Further RFCs that describe strongly encouraged, but non-essential, behaviors are listed in Section 3. Experimental extensions that are not yet standard practices, but that potentially could be in the future, are described in Section 4.

The reader will probably notice that these three sections are broadly equivalent to MUST/SHOULD/MAY specifications (per RFC 2119), and although the authors support this intuition, this document is merely descriptive; it does not represent a binding standards-track position. Individual implementers still need to examine the standards documents themselves to evaluate specific requirement levels.

A small number of older experimental extensions that have not been widely implemented, deployed, and used are noted in Section 5. Many other supporting documents that are relevant to the development, implementation, and deployment of TCP are described in Section 6.

A small number of fairly ubiquitous important implementation practices that is not currently documented in the RFC series is listed in Section 7.

Within each section, RFCs are listed in the chronological order of their publication dates.

## 2. Basic Functionality

A small number of documents compose the core specification of TCP. These define the required basic functionalities of TCP's header parsing, state machine, congestion control, and retransmission timeout computation. These base specifications must be correctly followed for interoperability.



RFC 793 S: "Transmission Control Protocol", STD 7 (September 1981)  
(Errata)

This is the fundamental TCP specification document [RFC0793]. Written by Jon Postel as part of the Internet protocol suite's core, it describes the TCP packet format, the TCP state machine and event processing, and TCP's semantics for data transmission, reliability, flow control, multiplexing, and acknowledgment.

Section 3.6 of RFC 793, describing TCP's handling of the IP precedence and security compartment, is mostly irrelevant today. RFC 2873 changed the IP precedence handling, and the security compartment portion of the API is no longer implemented or used. In addition, RFC 793 did not describe any congestion control mechanism. Otherwise, however, the majority of this document still accurately describes modern TCPs. RFC 793 is the last of a series of developmental TCP specifications, starting in the Internet Experimental Notes (IENs) and continuing in the RFC series.

RFC 1122 S: "Requirements for Internet Hosts - Communication Layers" (October 1989)

This document [RFC1122] updates and clarifies RFC 793, fixing some specification bugs and oversights. It also explains some features such as keep-alives and Karn's and Jacobson's RTO estimation algorithms [KP87][Jac88][JK92]. ICMP interactions are mentioned, and some tips are given for efficient implementation. RFC 1122 is an Applicability Statement, listing the various features that MUST, SHOULD, MAY, SHOULD NOT, and MUST NOT be present in standards-conforming TCP implementations. Unlike a purely informational "roadmap", this Applicability Statement is a standards document and gives formal rules for implementation.

RFC 2460 S: "Internet Protocol, Version 6 (IPv6) Specification" (December 1998) (Errata)

This document [RFC2460] is of relevance to TCP because it defines how the pseudo-header for TCP's checksum computation is derived when 128-bit IPv6 addresses are used instead of 32-bit IPv4 addresses. Additionally, RFC 2675 describes TCP changes required to support IPv6 jumbograms.

RFC 2873 S: "TCP Processing of the IPv4 Precedence Field" (June 2000) (Errata)

This document [RFC2873] removes from the TCP specification all processing of the precedence bits of the TOS byte of the IP

header. This resolves a conflict over the use of these bits between RFC 793 and Differentiated Services [RFC2474].

RFC 3390 S: "Increasing TCP's Initial Window" (October 2002)

This document [RFC3390] specifies an increase in the permitted initial window for TCP from one segment to three or four segments during the slow start phase, depending on the segment size.

RFC 5681 S: "TCP Congestion Control" (August 2009)

Although RFC 793 did not contain any congestion control mechanisms, today congestion control is a required component of TCP implementations. This document [RFC5681] defines the current versions of Van Jacobson's congestion avoidance and control mechanisms for TCP, based on his 1988 SIGCOMM paper [Jac88].

A number of behaviors that together constitute what the community refers to as "Reno TCP" are described in RFC 5681. The name "Reno" comes from the Net/2 release of the 4.3 BSD operating system. This is generally regarded as the least common denominator among TCP flavors currently found running on Internet hosts. Reno TCP includes the congestion control features of slow start, congestion avoidance, fast retransmit, and fast recovery.

RFC 1122 [RFC1122] mandates the implementation of a congestion control mechanism, and RFC 5681 [RFC5681] details the currently accepted mechanism. RFC 5681 differs slightly from the other documents listed in this section, as it does not affect the ability of two TCP endpoints to communicate; however, congestion control remains a critical component of any widely deployed TCP implementation and is required for the avoidance of congestion collapse and to ensure fairness among competing flows.

RFC 2001 and RFC 2581 are the conceptual precursors of RFC 5681. The most important changes relative to RFC 2581 are:

- (a) The initial window requirements were changed to allow larger Initial Windows as standardized in [RFC3390].
- (b) During slow start and congestion avoidance, the usage of Appropriate Byte Counting [RFC3465] is explicitly recommended.
- (c) The use of Limited Transmit [RFC3042] is now recommended.

RFC 6093 S: "On the Implementation of the TCP Urgent Mechanism" (January 2011)

This document [RFC6093] analyzes how current TCP stacks process TCP urgent indications, and how the behavior of widely deployed

middleboxes affects the urgent indications processing. Based on their investigation, the document updates the relevant specifications such that they accommodate current practice in processing TCP urgent indications. Finally, the document raises awareness about the reliability of TCP urgent indications in the Internet, and recommends against the use of urgent mechanism.

RFC 6298 S: "Computing TCP's Retransmission Timer" (June 2011)

Abstract: "This document defines the standard algorithm that Transmission Control Protocol (TCP) senders are required to use to compute and manage their retransmission timer. It expands on the discussion in section 4.2.3.1 of RFC 1122 and upgrades the requirement of supporting the algorithm from a SHOULD to a MUST." [RFC6298]. RFC 6298 is the successor of RFC 2988, which changes the initial RTO from 3s to 1s.

RFC 6691 I: "TCP Options and Maximum Segment Size (MSS)" (July 2012)

This document [RFC6691] clarifies what value to use with the TCP Maximum Segment Size (MSS) option when IP and TCP options are in use.

### 3. Recommended Enhancements

This section describes recommended TCP modifications that improve performance and security. Section 3.1 represents fundamental changes to the protocol. Section 3.2 lists improvements in the congestion control and loss recovery mechanisms specified in RFC 5681. Section 3.3 describes further refinements that make use of selective acknowledgments. Section 3.4 describes algorithms that allows a TCP sender to detect whether it has entered loss recovery unnecessarily. Section 3.5 compromises Path MTU Discovery mechanisms. Header compression schemes for TCP/IP header compression are listed in Section 3.6. Finally, Section 3.7 deals with the problem of preventing forged segments and flooding attacks.

#### 3.1. Fundamental Changes

RFC 1323 allows better utilization of high bandwidth-delay product paths by providing some needed mechanisms for high-rate transfers. RFC 2675 describes changes to TCP's semantic for using IPv6 Jumbograms. RFC 5482 specifies the TCP User Timeout Option.

## RFC 1323 S: "TCP Extensions for High Performance" (May 1992)

This document [RFC1323] defines TCP extensions for window scaling, timestamps, and protection against wrapped sequence numbers, for efficient and safe operation over paths with large bandwidth-delay products. These extensions are commonly found in currently used systems; however, they may require manual tuning and configuration. One issue in this specification that is still under discussion concerns a modification to the algorithm for estimating the mean RTT when timestamps are used. RFC 1072 and RFC 1185 are the conceptual precursors of RFC 1323.

## RFC 2675 S: "IPv6 Jumbograms" (August 1999) (Errata)

IPv6 supports longer datagrams than were allowed in IPv4. These are known as Jumbograms, and use with TCP has necessitated changes to the handling of TCP's MSS and Urgent fields (both 16 bits). This document [RFC2675] explains those changes. Although it describes changes to basic header semantics, these changes should only affect the use of very large segments, such as IPv6 jumbograms, which are currently rarely used in the general Internet.

Supporting the behavior described in this document does not affect interoperability with other TCP implementations when IPv4 or non-jumbogram IPv6 is used. This document states that jumbograms are to only be used when it can be guaranteed that all receiving nodes, including each router in the end-to-end path, will support jumbograms. If even a single node that does not support jumbograms is attached to a local network, then no host on that network may use jumbograms. This explains why jumbogram use has been rare, and why this document is considered a performance optimization and not part of TCP over IPv6's basic functionality.

## RFC 5482 S: "TCP User Timeout Option" (June 2009)

As a local per-connection parameter the TCP user timeout controls how long transmitted data may remain unacknowledged before a connection is forcefully closed. This document [RFC5482] specifies the TCP User Timeout Option that allows one end of a TCP connection to advertise its current user timeout value. This information provides advice to the other end of the TCP connection to adapt its user timeout accordingly.

### 3.2. Congestion Control and Loss Recovery Extensions

Two of the most important aspects of TCP are its congestion control and loss recovery features. TCP traditionally treats lost packets as indicating congestion-related loss, and cannot distinguish between congestion-related loss and loss due to transmission errors. Even when ECN is in use, there is a rather intimate coupling between congestion control and loss recovery mechanisms. There are several extensions to both features, and more often than not, a particular extension applies to both. In this sub-section, we group enhancements to either congestion control, loss recovery, or both, which can be performed unilaterally; that is, without negotiating support between endpoints. In the next sub-section, we group the extensions that specify or rely on the SACK option, which must be negotiated bilaterally. TCP implementations should include the enhancements from both sub-sections so that TCP senders can perform well without regard to the feature sets of other hosts they connect to. For example, if SACK use is not successfully negotiated, a host should use the NewReno behavior as a fall back.

RFC 3042 S: "Enhancing TCP's Loss Recovery Using Limited Transmit" (January 2001)

Abstract: "This document proposes Limited Transmit, a new Transmission Control Protocol (TCP) mechanism that can be used to more effectively recover lost segments when a connection's congestion window is small, or when a large number of segments are lost in a single transmission window." [RFC3042] Tests from 2004 showed that Limited Transmit was deployed in roughly one third of the web servers tested [MAF04].

RFC 3168 S: "The Addition of Explicit Congestion Notification (ECN) to IP" (September 2001)

This document [RFC3168] defines a means for end hosts to detect congestion before congested routers are forced to discard packets. Although congestion notification takes place at the IP level, ECN requires support at the transport level (e.g., in TCP) to echo the bits and adapt the sending rate. This document updates RFC 793 to define two previously unused flag bits in the TCP header for ECN support. RFC 3540 provides a supplementary (experimental) means for more secure use of ECN, and RFC 2884 provides some sample results from using ECN.

RFC 3465 E: "TCP Congestion Control with Appropriate Byte Counting (ABC)" (February 2003)

This document [RFC3465] suggests that congestion control use the

number of bytes acknowledged instead of the number of acknowledgments received. The ABC mechanism behaves differently than the standard method when there is not a one-to-one relationship between data segments and acknowledgements. ABC still operates within the accepted guidelines, but is more robust to delayed ACKs and ACK-division [SCWA99][RFC3449].

RFC 6633 S: "Deprecation of ICMP Source Quench Messages" (May 2012)

This document [RFC6633] formally deprecates the use of ICMP Source Quench messages by transport protocols and provides a recommendation against the implementation of [RFC1016].

RFC 6582 S: "The NewReno Modification to TCP's Fast Recovery Algorithm" (April 2012)

This document [RFC6582] specifies a modification to the standard Reno fast recovery algorithm, whereby a TCP sender can use partial acknowledgments to make inferences determining the next segment to send in situations where SACK would be helpful but isn't available. Although it is only a slight modification, the NewReno behavior can make a significant difference in performance when multiple segments are lost from a single window of data.

RFC 2582 and RFC 3782 are the conceptual precursors of RFC 6582. The main change in RFC 3782 relative to RFC 2582 was to specify the Careful variant of NewReno's Fast Retransmit and Fast Recovery algorithms and advance those two algorithms from Experimental to Standards Track status. The main change in RFC 6582 relative to RFC 3782 was to solve a performance degradation that could occur if FlightSize on Full ACK reception is zero.

### 3.3. SACK-Based Loss Recovery and Congestion Control

The base TCP specification in RFC 793 provided only a simple cumulative acknowledgment mechanism. However, a selective acknowledgment (SACK) mechanism provides performance improvement in the presence of multiple packet losses from the same flight, more than outweighing the modest increase in complexity. A TCP should be expected to implement SACK; however, SACK is a negotiated option and is only used if support is advertised by both sides of a connection.

RFC 2018 S: "TCP Selective Acknowledgment Options" (October 1996) (Errata)

When more than one packet is lost during one round trip time TCP may experience poor performance since a TCP sender can only learn about a single lost packet per round trip time from cumulative

acknowledgments. This document [RFC2018] defines the basic selective acknowledgment (SACK) mechanism for TCP, which can help to overcome these limitations. The receiving TCP returns SACK blocks to inform the sender which data has been received. The sender can then retransmit only the missing data segments.

RFC 2883 S: "An Extension to the Selective Acknowledgement (SACK) Option for TCP" (July 2000)

This document [RFC2883] extends RFC 2018. It enables use of the SACK option to acknowledge duplicate packets. With this extension, called DSACK, the sender is able to infer the order of packets received at the receiver, and therefore to infer when it has unnecessarily retransmitted a packet.

RFC 6675 S: "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP" (August 2012)

This document [RFC6675] describes a conservative loss recovery algorithm for TCP that is based on the use of the selective acknowledgment (SACK) TCP option [RFC2018]. The algorithm conforms to the spirit of the congestion control specification in RFC 5681, but allows TCP senders to recover more effectively when multiple segments are lost from a single flight of data.

RFC 6675 is a revision of RFC 3517 to address several situations that are not handled explicitly before. In particular

- (a) it improves the loss detection in the event that the sender has outstanding segments that are smaller than SMSS.
- (b) it modifies the definition of a "duplicate acknowledgment" to utilize the SACK information in detecting loss.
- (c) it maintains the ACK clock under certain circumstances involving loss at the end of the window.

### 3.4. Detection and Prevention of Spurious Retransmissions

Spurious retransmission timeouts are harmful to TCP performance and multiple algorithms have been defined for detecting when spurious retransmissions have occurred, and then responding differently in order to recover performance. The IETF defined multiple algorithms because there are tradeoffs in whether or not certain TCP options need to be implemented, and IPR status. The Standards Track documents in this section are closely related to the Experimental documents in Section 4.3 also addressing this topic.

## RFC 4015 S: "The Eifel Response Algorithm for TCP" (February 2005)

This document [RFC4015] describes the response portion of the Eifel algorithm, which can be used in conjunction with one of several methods of detecting when loss recovery has been spuriously entered, such as the Eifel detection algorithm in RFC 3522, the algorithm in RFC 3708, or F-RTO in RFC 5682.

Abstract: "Based on an appropriate detection algorithm, the Eifel response algorithm provides a way for a TCP sender to respond to a detected spurious timeout. It adapts the retransmission timer to avoid further spurious timeouts, and can avoid - depending on the detection algorithm - the often unnecessary go-back-N retransmits that would otherwise be sent. In addition, the Eifel response algorithm restores the congestion control state in such a way that packet bursts are avoided."

## RFC 5682 S: "Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP" (September 2009)

The F-RTO detection algorithm [RFC5682], originally describes in RFC 4138, provides an option for inferring spurious retransmission timeouts. Unlike some similar detection methods (e.g. RFC 3522 and RFC 3708), F-RTO does not rely on the use of any TCP options. The basic idea is to send previously unsent data after the first retransmission after a RTO. If the ACKs advance the window, the RTO may be declared spurious.

## 3.5. Path MTU Discovery

The MTUs supported by different links and tunnels within the Internet can vary widely. Fragmentation of packets larger than the supported MTU on a hop is undesirable. As TCP is the segmentation layer for dividing an application's bytestream into IP packet payloads, TCP implementations generally include Path MTU Discovery (PMTUD) mechanisms in order to maximize the size of segments they send, without causing fragmentation within the network. Some algorithms may utilize signalling from routers on the path that the MTU has been exceeded.

## RFC 1191 S: "Path MTU Discovery" (November 1990)

Abstract: "This memo describes a technique for dynamically discovering the MTU of an arbitrary Internet path. It specifies a small change to the way routers generate one type of ICMP message. For a path that passes through a router that has not been so changed, this technique might not discover the correct path MTU, but it will always choose a path MTU as accurate as, and in many



cases more accurate than, the path MTU that would be chosen by current practice." [RFC1191]

RFC 1981 S: "Path MTU Discovery for IP version 6" (August 1996)

Abstract: "This document describes Path MTU Discovery for IP version 6. It is largely derived from RFC 1191, which describes Path MTU Discovery for IP version 4." [RFC1981]

RFC 4821 S: "Packetization Layer Path MTU Discovery" (March 2007)

Abstract: "This document describes a robust method for Path MTU Discovery (PMTUD) that relies on TCP or some other Packetization Layer to probe an Internet path with progressively larger packets. This method is described as an extension to RFC 1191 and RFC 1981, which specify ICMP-based Path MTU Discovery for IP versions 4 and 6, respectively." [RFC4821]

### 3.6. Header Compression

Especially in streaming applications, the overhead of TCP/IP headers could correspond to more than 50% of the total amount of data sent. Such large overheads may be tolerable in wired LANs where capacity is often not an issue, but are excessive for WANs and wireless systems where bandwidth is scarce. Header compressions schemes for TCP/IP like the RObust Header Compression (ROHC) can significantly compresses these overhead. It performs well over links with significant error rates and long round-trip times.

RFC 1144 S: "Compressing TCP/IP Headers for Low-Speed Serial Links" (February 1990)

This document [RFC1144] describes a method for compressing the headers of TCP/IP datagrams to improve performance over low speed serial links. The method described in this document is limited in its handling of TCP options and cannot compress the headers of SYNs and FINs.

RFC 6846 S: "RObust Header Compression (ROHC): A Profile for TCP/IP (ROHC-TCP)" January 2013)

From abstract: "This document specifies a RObust Header Compression (ROHC) profile for compression of TCP/IP packets. The profile, called ROHC-TCP, provides efficient and robust compression of TCP headers, including frequently used TCP options such as selective acknowledgments (SACKs) and Timestamps." [RFC6846] RFC 6846 is the successor of RFC 4996. It fixes a technical issue with the SACK compression and clarifies other

compression methods used.

### 3.7. Defending Spoofing and Flooding Attacks

By default, TCP lacks any cryptographic structures to differentiate legitimate segments and those spoofed from malicious hosts. Spoofing valid segments requires correctly guessing a number of fields. The documents in this sub-section describe ways to make that guessing harder, or to prevent it from being able to affect a connection negatively.

RFC 4953 I: "Defending TCP Against Spoofing Attacks" (July 2007)

This document [RFC4953] discusses the recently increased vulnerability of long-lived TCP connections, such as BGP connections, to resets (RSTs) spoofing attacks. The document analyses the vulnerability, discussing proposed solutions at the transport level and their inherent challenges, as well as existing network level solutions and the feasibility of their deployment.

RFC 5461 I: "TCP's Reaction to Soft Errors" (February 2009)

This document [RFC5461] describes a non-standard but widely implemented modification to TCP's handling of ICMP soft error messages that rejects pending connection-requests when such error messages are received. This behavior reduces the likelihood of long delays between connection-establishment attempts that may arise in some scenarios.

RFC 4987 I: "TCP SYN Flooding Attacks and Common Mitigations" (August 2007)

This document [RFC4987] describes the well-known TCP SYN flooding attack. It analyses and discusses various countermeasures against these attacks, including their use and trade-offs.

RFC 5925 S: "The TCP Authentication Option" (May 2010)

This document [RFC5925] describes the TCP Authentication Option (TCP-AO), which is used to authenticate TCP segments. TCP-AO obsoletes the TCP MD5 Signature option of RFC 2385. It supports the use of stronger hash functions, protects against replays for long-lived TCP connections (as used, e.g., in BGP and LDP), coordinates key exchanges between endpoints, and provides a more explicit recommendation for external key management. Cryptographic algorithms for TCP-AO are defined in [RFC5926].

RFC 5926 S: "Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)" (May 2010)

This document [RFC5926] specifies the algorithms and attributes that can be used in TCP Authentication Option's (TCP-AO) current manual keying mechanism and provides the interface for future message authentication codes (MACs).

RFC 5927 I: "ICMP attacks against TCP" (July 2010)

Abstract: "This document discusses the use of the Internet Control Message Protocol (ICMP) to perform a variety of attacks against the Transmission Control Protocol (TCP). Additionally, this document describes a number of widely implemented modifications to TCP's handling of ICMP error messages that help to mitigate these issues." [RFC5927]

RFC 5961 S: "Improving TCP's Robustness to Blind In-Window Attacks" (August 2010)

This document [RFC5961] describes minor modifications to how TCP handles inbound segments. This renders TCP connections, especially long-lived connections such as H-323 or BGP, are less vulnerable to spoofed packet injection attacks where the 4-tuple (the source and destination IP addresses and the source and destination ports) has been guessed.

RFC 6528 S: "Defending Against Sequence Number Attacks" (February 2012)

Abstract: "This document [RFC6528] specifies an algorithm for the generation of TCP Initial Sequence Numbers (ISNs), such that the chances of an off-path attacker guessing the sequence numbers in use by a target connection are reduced. This document revises (and formally obsoletes) RFC 1948, and takes the ISN generation algorithm originally proposed in that document to Standards Track, formally updating RFC 793.

#### 4. Experimental Extensions

The RFCs in this section are still experimental, but they may become proposed standards in the future. At least part of the reason that they are still experimental is to gain more wide-scale experience with them before a standards track decision is made. By their publication as experimental RFCs, it is hoped that the community of TCP researchers will analyze and test the contents of these RFCs. Although experimentation is encouraged, there is not yet formal

consensus that these are fully logical and safe behaviors. Wide-scale deployment of implementations that use these features should be well thought-out in terms of consequences.

#### 4.1. Architectural Guidelines

As multiple flows may share the same paths, sections of paths, or other resources, the TCP implementation may benefit from sharing information across TCP connections or other flows. Some Experimental proposals have been documented and some implementations have included the concepts.

RFC 2140 I: "TCP Control Block Interdependence" (April 1997)

This document [RFC2140] suggests how TCP connections between the same endpoints might share information, such as their congestion control state. To some degree, this is done in practice by a few operating systems; for example, Linux currently has a destination cache. Although this RFC is technically informational, the concepts it describes are in experimental use, so we include it in this section.

RFC 3124 S: "The Congestion Manager" (June 2001)

This document [RFC3124], the Congestion Manager, is a related proposal to RFC 2140. The idea behind the Congestion Manager, moving congestion control outside of individual TCP connections, represents a modification to the core of TCP, which supports sharing information among TCP connections as well. Although a Proposed Standard, some pieces of the Congestion Manager support architecture have not been specified yet, and it has not achieved use or implementation beyond experimental stacks, so it is not listed among the standard TCP enhancements in this roadmap.

#### 4.2. Congestion Control and Loss Recovery Extensions

TCP congestion control has been an extremely active research area for many years, as it determines the performance of many applications that use TCP. A number of experimental RFCs address issues with flow startup, overshoot, and steady-state behavior in the basic RFC 5681 algorithms.

RFC 2861 E: "TCP Congestion Window Validation" (June 2000)

This document [RFC2861] suggests reducing the congestion window over time when no packets are flowing. This behavior is more aggressive than that specified in RFC 5681, which says that a TCP sender SHOULD set its congestion window to the initial window

after an idle period of an RTO or greater.

RFC 3540 E: "Robust Explicit Congestion Notification (ECN) signaling with Nonces" (June 2003)

This document [RFC3540] describes an optional addition to ECN that protects against accidental or malicious concealment of marked packets from the TCP sender.

RFC 3649 E: "HighSpeed TCP for Large Congestion Windows" (December 2003)

This document [RFC3649] proposes a modification to TCP's congestion control mechanism for use with TCP connections with large congestion windows, to allow TCP to achieve a higher throughput in high-bandwidth environments.

RFC 3742 E: "Limited Slow-Start for TCP with Large Congestion Windows" (March 2004)

This document [RFC3742] describes a more conservative slow-start behavior to prevent massive packet losses when a connection uses a very large congestion window.

RFC 4782 E: "Quick-Start for TCP and IP" (January 2007) (Errata)

This document [RFC4782] specifies the optional Quick-Start mechanism for TCP. This mechanism allows connections to use higher sending rates at the beginning of the data transfer or after an idle period, provided that there is significant unused bandwidth along the path, and the sender and all of the routers along the path approve this higher rate.

RFC 5562 E: "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets" (June 2009)

This document [RFC5562] describes an experimental modification to ECN [RFC3168] for the use of ECN in TCP SYN/ACK packets. This would allow to ECN-mark rather than drop the TCP SYN/ACK packet at an ECN-capable router, and to avoid the severe penalty of a retransmission timeout for a connection when the SYN/ACK packet is dropped.

RFC 5690 I: "Adding Acknowledgement Congestion Control to TCP" (February 2010)

This document [RFC5690] describes a congestion control mechanism for acknowledgment (ACKs) traffic in TCP. The mechanism is based

on the acknowledgment congestion control of the Datagram Congestion Control Protocol's (DCCP's) [RFC4340] Congestion Control Identifier (CCID) 2 [RFC4341].

RFC 5827 E: "Early Retransmit for TCP and SCTP" (April 2010)

This document [RFC5827] proposes the "Early Retransmit" mechanism for TCP (and SCTP) that can be used to recover lost segments when a connection's congestion window is small. In certain special circumstances, Early Retransmit reduces the number of duplicate acknowledgments required to trigger fast retransmit to recover segment losses without waiting for a lengthy retransmission timeout.

RFC 6069 E: "Making TCP more Robust to Long Connectivity Disruptions (TCP-LCD)" (December 2010)

This document [RFC6069] describes how standard ICMP messages can be used to disambiguate true congestion loss from non-congestion loss caused by connectivity disruptions. It proposes a reversion strategy of TCP's retransmission timer that enables a more prompt detection of whether or not the connectivity has been restored.

RFC 6928 E: "Increasing TCP's Initial Window" (April 2013)

This document [RFC6928] proposes to increase the TCP initial window from between 2 and 4 segments, as specified in RFC 3390, to 10 segments with a fallback to the existing recommendation when performance issues are detected.

RFC 6937 E: "Proportional Rate Reduction for TCP" (May 2013)

This document [RFC6937] describes an experimental Proportional Rate Reduction (PRR) algorithm as an alternative to the widely deployed Fast Recovery algorithm, to improve the accuracy of the amount of data sent by TCP during loss recovery.

#### 4.3. Detection and Prevention of Spurious Retransmissions

In addition to the Standards Track extensions to deal with spurious retransmissions in Section 3.4, Experimental proposals have also been documented.

RFC 3522 E: "The Eifel Detection Algorithm for TCP" (April 2003)

The Eifel detection algorithm [RFC3522] allows a TCP sender to detect a posteriori whether it has entered loss recovery unnecessarily by using the TCP timestamp option to solve the ACK

ambiguity.

RFC 3708 E: "Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions" (February 2004)

Abstract: "TCP and Stream Control Transmission Protocol (SCTP) provide notification of duplicate segment receipt through Duplicate Selective Acknowledgement (DSACKs) and Duplicate Transmission Sequence Number (TSN) notification, respectively. This document presents conservative methods of using this information to identify unnecessary retransmissions for various applications." [RFC3708]

RFC 4653 E: "Improving the Robustness of TCP to Non-Congestion Events" (August 2008)

In the presence of non-congestion events, such as reordering an out-of-order segment does not necessarily indicates a lost segment and congestion. This document [RFC4653] proposes to increase the threshold used to trigger a fast retransmission from the fixed value of three duplicate ACKs to about one congestion window of data in order to disambiguate true segment loss from segment reordering.

#### 4.4. Multipath TCP

MultiPath TCP (MPTCP) is an ongoing effort within the IETF that allows a TCP connection to simultaneously use multiple IP-addresses/interfaces to spread their data across several subflows, while presenting a regular TCP interface to applications. Benefits of this include better resource utilization, better throughput and smoother reaction to failures. The documents listed in this section specify the Multipath TCP scheme, while the documents in Sections 6.4, 6.2 and 6.5 provide some additional background information.

RFC 6356 E: "Coupled Congestion Control for Multipath Transport Protocols" (August 2011)

This document [RFC6356] presents a congestion control algorithm for multipath transport protocols such as Multipath TCP. It couples the congestion control algorithms running on different subflows by linking their increase functions, and dynamically controls the overall aggressiveness of the multipath flow. The result is an algorithm that is fair to TCP at bottlenecks while moving traffic away from congested links.

RFC 6824 E: "TCP Extensions for Multipath Operation with Multiple Addresses" (January 2013) (Errata)

This document [RFC6824] presents protocol changes required to add multipath capability to TCP; specifically, those for signaling and setting up multiple paths ("subflows"), managing these subflows, reassembly of data, and termination of sessions.

## 5. Historic Extensions

The RFCs listed here define extensions that have thus far failed to arouse substantial interest from implementers and have never seen widespread, or were found to be defective for general use. Most of them are reclassified by RFC 6247 [RFC6247] to Historic status.

RFC 721 U: "Out-of-Band Control Signals in a Host-to-Host Protocol" (September 1976): lack of interest

RFC 721 [RFC0721] addresses the problem of implementing a reliable out-of-band signal (interrupts) for use in a host-to-host protocol. The proposal has not been included in the final TCP specification.

RFC 1078 U: "TCP Port Service Multiplexer (TCPMUX)" (November 1988): lack of interest

This document [RFC1078] propose a protocol to contact multiple services on a single well-known TCP port using a service name instead of a well-known number.

RFC 1106 H: "TCP Big Window and NAK Options" (June 1989): found defective

This RFC [RFC1106] defined an alternative to the Window Scale option for using large windows and described the "negative acknowledgement" or NAK option. There is a comparison of NAK and SACK methods, and early discussion of TCP over satellite issues. RFC 1110 explains some problems with the approaches described in RFC 1106. The options described in this document have not been adopted by the larger community, although NAKs are used in the SCPS-TP adaptation of TCP for satellite and spacecraft use, developed by the Consultative Committee for Space Data Systems (CCSDS).



RFC 1110 H: "A Problem with the TCP Big Window Option" (August 1989): deprecates RFC 1106

Abstract: "The TCP Big Window option discussed in RFC 1106 will not work properly in an Internet environment which has both a high bandwidth \* delay product and the possibility of disordering and duplicating packets. In such networks, the window size must not be increased without a similar increase in the sequence number space. Therefore, a different approach to big windows should be taken in the Internet." [RFC1110]

RFC 1146 H: "TCP Alternate Checksum Options" (March 1990): lack of interest

This document [RFC1146] defined more robust TCP checksums than the 16-bit ones-complement in use today. A typographical error in RFC 1145 is fixed in RFC 1146; otherwise, the documents are the same.

RFC 1263 I: "TCP Extensions Considered Harmful" (October 1991): lack of interest

This document [RFC1263] argues against "backwards compatible" TCP extensions. Specifically mentioned are several TCP enhancements that have been successful, including timestamps, window scaling, PAWS, and SACK. RFC 1263 presents an alternative approach called "protocol evolution", whereby several evolutionary versions of TCP would exist on hosts. These distinct TCP versions would represent upgrades to each other and could be header-incompatible. Interoperability would be provided by having a virtualization layer select the right TCP version for a particular connection. This idea did not catch on with the community, while the type of extensions RFC 1263 specifically targeted as harmful did become popular.

RFC 1379 H: "Extending TCP for Transactions -- Concepts" (November 1992): found defective

See RFC 1644.

RFC 1644 H: "T/TCP -- TCP Extensions for Transactions Functional Specification" (July 1994): found defective

The inventors of TCP believed that cached connection state could have been used to eliminate TCP's 3-way handshake, to support two-packet request/response exchanges. RFCs 1379 [RFC1379] and 1644 [RFC1644] show that this is far from simple. Furthermore, T/TCP floundered on the ease of denial-of-service attacks that can result. One idea pioneered by T/TCP lives on in RFC 2140, in the

sharing of state across connections.

RFC 1693 H: "An Extension to TCP: Partial Order Service" (November 1994): lack of interest

This document [RFC1693] defines a TCP extension for applications that do not care about the order in which application-layer objects are received. Examples are multimedia and database applications. In practice, these applications either accept the possible performance loss because of TCP's strict ordering or they use more specialized transport protocols.

RFC 1705 I: "Six Virtual Inches to the Left: The Problem with IPng" (October 1994): lack of interest

To overcome the exhaustion of the IP class B address space, suggest this document [RFC1705] that a new version of TCP (TCPng) needs to be developed and deployed. It proposes that a globally unique address be assigned to Transport layer to uniquely identify an internet host without specifying any routing information.

RFC 6013 I: "TCP Cookie Transactions (TCPCT)" (January 2011): lack of interest

This document [RFC6013] describes a method to exchange a cookie (nonce) during the connection establishment to negotiates elimination of receiver state. These cookies are later used to inhibit premature closing of connections, and reduce retention of state after the connection has terminated.

Since the cookie pair is too large to fit with the other TCP options in the 40 bytes of TCP option space, the document further describes method to extent the option space after the connection establishment.

Although the RFC 6013 is publish in 2011, the author of this document places it in this section of the roadmap document due to two factors.

- (a) The author is not aware of any wide deployment and use of RFC 6013.
- (b) RFC 6013 uses experimental TCP option codepoints, which prohibits a large scale deployment.

## 6. Support Documents

This section contains several classes of documents that do not necessarily define current protocol behaviors, but that are

nevertheless of interest to TCP implementers. Section 6.1 describes several foundational RFCs that give modern readers a better understanding of the principles underlying TCP's behaviors and development over the years. Section 6.2 contains architectural guidelines and principles for TCP architects and designers. The documents listed in Section 6.3 provide advice on using TCP in various types of network situations that pose challenges above those of typical wired links. Guidance for developing, analyzing, and evaluating TCP is given in Section 6.4. Some implementation notes and implementation advices can be found in Section 6.5. The TCP Management Information Bases are described in Section 6.6. RFCs that describe tools for testing and debugging TCP implementations or that contain high-level tutorials on the protocol are listed Section 6.7, and Section 6.8 lists a number of case studies that have explored TCP performance.

#### 6.1. Foundational Works

The documents listed in this section contain information that is largely duplicated by the standards documents previously discussed. However, some of them contain a greater depth of problem statement explanation or other context. Particularly, RFCs 813 - 817 (known as the "Dave Clark Five") describe some early problems and solutions (RFC 815 only describes the reassembly of IP fragments and is not included in this TCP roadmap).

RFC 675 U: "Specification of Internet Transmission Control Program" (December 1974)

This document [RFC0675] is a very early precursor of the infamous RFC 793 which already contained the three-way handshake in its final form and the concept of sliding windows for reliable data transmission. Apart from that the segment layout is totally different and the specified API differs from the latter RFC 793.

RFC 761 H: "DoD standard Transmission Control Protocol" (January 1980)

This document [RFC0761] is the immediate predecessor of RFC 793. The header format, the connection establishment including the different connection states, and the overall API correspond mostly to the final Standard RFC 793.

RFC 813 U: "Window and Acknowledgement Strategy in TCP" (July 1982)

This document [RFC0813] contains an early discussion of Silly Window Syndrome and its avoidance and motivates and describes the use of delayed acknowledgments.

RFC 814 U: "Name, Addresses, Ports, and Routes" (July 1982)

Suggestions and guidance for the design of tables and algorithms to keep track of various identifiers within a TCP/IP implementation are provided by this document [RFC0814].

RFC 816 U: "Fault Isolation and Recovery" (July 1982)

In this document [RFC0816], TCP's response to indications of network error conditions such as timeouts or received ICMP messages is discussed.

RFC 817 U: "Modularity and Efficiency in Protocol Implementation" (July 1982)

This document [RFC0817] contains implementation suggestions that are general and not TCP specific. However, they have been used to develop TCP implementations and describe some performance implications of the interactions between various layers in the Internet stack.

RFC 872 U: "TCP-ON-A-LAN" (September 1982)

Conclusion: "The sometimes-expressed fear that using TCP on a local net is a bad idea is unfounded." [RFC0872]

RFC 896 U: "Congestion Control in IP/TCP Internetworks" (January 1984)

This document [RFC0896] contains some early experiences with congestion collapse and some initial thoughts on how to avoid it using congestion control in TCP.

RFC 964 U: "Some Problems with the Specification of the Military Standard Transmission Control Protocol" (November 1985)

This document [RFC0964] points out several specification bugs in the US Military's MIL-STD-1778 document, which was intended as a successor to RFC 793. This serves to remind us of the difficulty in specification writing (even when we work from existing documents!).

## 6.2. Architectural Guidelines

Some documents in this section contain architectural guidance and concerns, while others specify TCP- and congestion-control-related mechanisms that are broadly applicable and have impacts on TCP's congestion control techniques. Some of these documents are direct

products of the Internet Architecture Board (IAB), giving their guidance on specific aspects of congestion control in the Internet.

RFC 1958 I: "Architectural Principles of the Internet" (June 1996)

This document [RFC1958] describes the underlying principles of the Internet architecture. It provides guidelines for network systems design that have proven useful in the evolution of the Internet.

RFC 2914 B: "Congestion Control Principles" (September 2000)

This document [RFC2914] motivates the use of end-to-end congestion control for preventing congestion collapse and providing fairness to TCP.

RFC 3439 I: "Some Internet Architectural Guidelines and Philosophy" (December 2002)

This document [RFC3439] extends RFC 1958 by outlining some philosophical guidelines for architects and designers of Internet backbone networks. The document describes the Simplicity Principle, which states that complexity is the primary mechanism that impedes efficient scaling.

RFC 6182 I: "Architectural Guidelines for Multipath TCP Development" (March 2011)

Abstract: "This document outlines architectural guidelines for the development of a Multipath Transport Protocol, with references to how these architectural components come together in the development of a Multipath TCP (MPTCP). This document lists certain high-level design decisions that provide foundations for the design of the MPTCP protocol, based upon these architectural requirements" [RFC6182]

### 6.3. Difficult Network Environments

As the internetworking field has explored wireless, satellite, cellular telephone, and other kinds of link-layer technologies, a large body of work has built up on enhancing TCP performance for such links. The RFCs listed in this section describe some of these more challenging network environments and how TCP interacts with them.

RFC 2488 B: "Enhancing TCP Over Satellite Channels using Standard Mechanisms" (January 1999)

From abstract: "While TCP works over satellite channels there are several IETF standardized mechanisms that enable TCP to more

effectively utilize the available capacity of the network path. This document outlines some of these TCP mitigations. At this time, all mitigations discussed in this document are IETF standards track mechanisms (or are compliant with IETF standards)." [RFC2488]

RFC 2757 I: "Long Thin Networks" (January 2000)

Several methods of improving TCP performance over long thin networks, such as geosynchronous satellite links, are discussed in this document [RFC2757]. A particular set of TCP options is developed that should work well in such environments and be safe to use in the global Internet. The implications of such environments have been further discussed in RFC 3150 and RFC 3155, and these documents should be preferred where there is overlap between them and RFC 2757.

RFC 2760 I: "Ongoing TCP Research Related to Satellites" (February 2000)

This document [RFC2760] discusses the advantages and disadvantages of several different experimental means of improving TCP performance over long-delay or error-prone paths. These include T/TCP, larger initial windows, byte counting, delayed acknowledgments, slow start thresholds, NewReno and SACK-based loss recovery, FACK [MM96], ECN, various corruption-detection mechanisms, congestion avoidance changes for fairness, use of multiple parallel flows, pacing, header compression, state sharing, and ACK congestion control, filtering, and reconstruction. Although RFC 2488 looks at standard extensions, this document focuses on more experimental means of performance enhancement.

RFC 3135 I: "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations" (June 2001)

From abstract: "This document is a survey of Performance Enhancing Proxies (PEPs) often employed to improve degraded TCP performance caused by characteristics of specific link environments, for example, in satellite, wireless WAN, and wireless LAN environments. Different types of Performance Enhancing Proxies are described as well as the mechanisms used to improve performance." [RFC3135]

RFC 3150 B: "End-to-end Performance Implications of Slow Links" (July 2001)

From abstract: "This document makes performance-related

recommendations for users of network paths that traverse "very low bit-rate" links....This recommendation may be useful in any network where hosts can saturate available bandwidth, but the design space for this recommendation explicitly includes connections that traverse 56 Kb/second modem links or 4.8 Kb/second wireless access links - both of which are widely deployed." [RFC3150]

RFC 3155 B: "End-to-end Performance Implications of Links with Errors" (August 2001)

From abstract: "This document discusses the specific TCP mechanisms that are problematic in environments with high uncorrected error rates, and discusses what can be done to mitigate the problems without introducing intermediate devices into the connection." [RFC3155]

RFC 3366 B: "Advice to link designers on link Automatic Repeat reQuest (ARQ)" (August 2002)

From abstract: "This document provides advice to the designers of digital communication equipment and link-layer protocols employing link-layer Automatic Repeat reQuest (ARQ) techniques. This document presumes that the designers wish to support Internet protocols, but may be unfamiliar with the architecture of the Internet and with the implications of their design choices for the performance and efficiency of Internet traffic carried over their links." [RFC3366]

RFC 3449 B: "TCP Performance Implications of Network Path Asymmetry" (December 2002)

From abstract: "This document describes TCP performance problems that arise because of asymmetric effects. These problems arise in several access networks, including bandwidth-asymmetric networks and packet radio subnetworks, for different underlying reasons. However, the end result on TCP performance is the same in both cases: performance often degrades significantly because of imperfection and variability in the ACK feedback from the receiver to the sender.

The document details several mitigations to these effects, which have either been proposed or evaluated in the literature, or are currently deployed in networks." [RFC3449]

RFC 3481 B: "TCP over Second (2.5G) and Third (3G) Generation Wireless Networks" (February 2003)

From abstract: "This document describes a profile for optimizing TCP to adapt so that it handles paths including second (2.5G) and third (3G) generation wireless networks." [RFC3481]

RFC 3819 B: "Advice for Internet Subnetwork Designers" (July 2004)

This document [RFC3819] describes how TCP performance can be negatively affected by some particular lower-layer behaviors and provides guidance in designing lower-layer networks and protocols to be amicable to TCP.

#### 6.4. Guidance for Developing, Analyzing, and Evaluating TCP

Documents in this section give general guidance for developing, analyzing, and evaluating TCP. Some of the documents discuss for example the properties of congestion control protocols that are "safe" for Internet deployment, as well as how to measure the properties of congestion control mechanisms and transport protocols.

RFC 4774 B: "Specifying Alternate Semantics for the Explicit Congestion Notification (ECN) Field" (November 2006)

This document [RFC4774] discusses some of the issues in defining alternate semantics for the ECN field, and specifies requirements for a safe co-existence in an Internet that may include routers that do not understand the defined alternate semantics.

RFC 5033 B: "Specifying New Congestion Control Algorithms" (August 2007)

This document [RFC5033] considers the evaluation of suggested congestion control algorithms that differ from the principles outlined in RFC 2914. It is useful for authors of such algorithms as well as for IETF members reviewing the associated documents.

RFC 5166 I: "Metrics for the Evaluation of Congestion Control Mechanisms" (March 2008)

This document [RFC5166] discusses metrics that needs to be considered when evaluating new or modified congestion control mechanisms for the Internet. Among others, the document discusses throughput, delay, loss rates, response times, fairness and robustness for challenging environments.



RFC 6181 I: "Threat Analysis for TCP Extensions for Multipath Operation with Multiple Addresses" (March 2011)

This document [RFC6181] describes a threat analysis for Multipath TCP (MPTCP). The document discusses several types of attacks and provides recommendations for MPTCP designers how to create an MPTCP specification that is as secure as the current (single-path) TCP.

RFC 6349 I: "Framework for TCP Throughput Testing" (August 2011)

From abstract: "This document describes a practical methodology for measuring end-to-end TCP throughput in a managed IP network. The goal is to provide a better indication in regard to user experience. In this framework, TCP and IP parameters are specified to optimize TCP throughput." [RFC6349]

#### 6.5. Implementation Advice

RFC 794 U: "PRE-EMPTION" (September 1981)

This document [RFC0794] discusses on a high-level the realization of pre-emption in TCP.

RFC 879 U: "The TCP Maximum Segment Size and Related Topics" (November 1983)

Abstract: "This memo discusses the TCP Maximum Segment Size Option and related topics. The purposes is to clarify some aspects of TCP and its interaction with IP. This memo is a clarification to the TCP specification, and contains information that may be considered as 'advice to implementers'." [RFC0879]

RFC 1071 U: "Computing the Internet Checksum" (September 1988) (Errata)

This document [RFC1071] lists a number of implementation techniques for efficiently computing the Internet checksum (used by TCP).

RFC 1624 I: "Computation of the Internet Checksum via Incremental Update" (May 1994)

Incrementally updating the Internet checksum is useful to routers in updating IP checksums. Some middleboxes that alter TCP headers may also be able to update the TCP checksum incrementally. This document [RFC1624] expands upon the explanation of the incremental update procedure in RFC 1071.

RFC 1936 I: "Implementing the Internet Checksum in Hardware" (April 1996)

This document [RFC1936] describes the motivation for implementing the Internet checksum in hardware, rather than in software, and provides an implementation example.

RFC 2525 I: "Known TCP Implementation Problems" (March 1999)

From abstract: "This memo catalogs a number of known TCP implementation problems. The goal in doing so is to improve conditions in the existing Internet by enhancing the quality of current TCP/IP implementations." [RFC2525]

RFC 2923 I: "TCP Problems with Path MTU Discovery" (September 2000)

From abstract: "This memo catalogs several known Transmission Control Protocol (TCP) implementation problems dealing with Path Maximum Transmission Unit Discovery (PMTUD), including the long-standing black hole problem, stretch acknowledgements (ACKs) due to confusion between Maximum Segment Size (MSS) and segment size, and MSS advertisement based on PMTU." [RFC2923]

RFC 3360 B: "Inappropriate TCP Resets Considered Harmful" (August 2002)

This document [RFC3360] is a plea that firewall vendors not send gratuitous TCP RST (Reset) packets when unassigned TCP header bits are used. This practice prevents desirable extension and evolution of the protocol and thus is potentially harmful to the future of the Internet.

RFC 3493 I: "Basic Socket Interface Extensions for IPv6" (February 2003)

This document [RFC3493] describes the de facto standard sockets API for programming with TCP. This API is implemented nearly ubiquitously in modern operating systems and programming languages.

RFC 6056 B: "Recommendations for Transport-Protocol Port Randomization" (December 2010)

This document [RFC6056] describes a number of simple and efficient methods for the selection of the client port number. It reduces the possibility of an attacker guessing the correct five-tuple (Protocol, Source/Destination Address, Source/Destination Port).

RFC 6191 B: "Reducing the TIME-WAIT State Using TCP timestamps"  
(April 2011)

This document [RFC6191] describes the usage of the TCP Timestamps option [JBB92] to perform heuristics to determine whether or not to allow the creation of a new incarnation of a connection that is in the TIME-WAIT state.

RFC 6429 I: "TCP Sender Clarification for Persist Condition"  
(December 2011)

This document [RFC6429] clarifies the actions that a TCP can be taken on connections that are experiencing the Zero Window Probe (ZWP) condition.

RFC 6897 I: "Multipath TCP (MPTCP) Application Interface  
Considerations (March 2013)

This document [RFC6897] characterizes the impact that Multipath TCP (MPTCP) may have on applications. It further discusses compatibility issues of MPTCP in combination with non-MPTCP-aware applications. Finally, it describes a basic API that is a simple extension of TCP's interface for MPTCP-aware applications.

## 6.6. Management Information Bases

The first MIB module defined for use with Simple Network Management Protocol (SNMP) (in RFC 1066 and its update, RFC 1156) was a single monolithic MIB module, called MIB-I. This evolved over time to be MIB-II (RFC 1213). It then became apparent that having a single monolithic MIB module was not scalable, given the number and breadth of MIB data definitions that needed to be included. Thus, additional MIB modules were defined, and those parts of MIB-II that needed to evolve were split off. Eventually, the remaining parts of MIB-II were also split off, the TCP-specific part being documented in RFC 2012.

RFC 2012 was obsoleted by RFC 4022, which is the primary TCP MIB document today. MIB-I, defined in RFC 1156, has been obsoleted by the MIB-II specification in RFC 1213. For current TCP implementers, RFC 4022 should be supported.

RFC 1066 H: "Management Information Base for Network Management of TCP/IP-based Internets" (August 1988)

This document [RFC1066] was the description of the TCP MIB. It was obsoleted by RFC 1156.

RFC 1156 S: "Management Information Base for Network Management of TCP/IP-based Internets" (May 1990)

This document [RFC1156] describes the required MIB fields for TCP implementations, with minor corrections and no technical changes from RFC 1066, which it obsoletes. This is the standards track document for MIB-I.

RFC 1213 S: "Management Information Base for Network Management of TCP/IP-based Internets: MIB-II" (March 1991)

This document [RFC1213] describes the second version of the MIB in a monolithic form. RFC 2012 updates this document by splitting out the TCP-specific portions.

RFC 2012 S: "SNMPv2 Management Information Base for the Transmission Control Protocol using SMIV2" (November 1996)

This document [RFC2012] defined the TCP MIB, in an update to RFC 1213. It is now obsoleted by RFC 4022.

RFC 2452 S: "IP Version 6 Management Information Base for the Transmission Control Protocol" (December 1998)

This document [RFC2452] augments RFC 2012 by adding an IPv6-specific connection table. The rest of 2012 holds for any IP version. RFC 2012 is now obsoleted by RFC 4022.

Although it is a standards track document, RFC 2452 is considered a historic mistake by the MIB community, as it is based on the idea of parallel IPv4 and IPv6 structures. Although IPv6 requires new structures, the community has decided to define a single generic structure for both IPv4 and IPv6. This will aid in definition, implementation, and transition between IPv4 and IPv6.

RFC 4022 S: "Management Information Base for the Transmission Control Protocol (TCP)" (March 2005)

This document [RFC4022] obsoletes RFC 2012 and RFC 2452 and specifies the current standard for the TCP MIB that should be deployed.

## 6.7. Tools and Tutorials

RFC 1180 I: "TCP/IP Tutorial" (January 1991) (Errata)

This document [RFC1180] is an extremely brief overview of the TCP/IP protocol suite as a whole. It gives some explanation as to how and where TCP fits in.

RFC 1470 I: "FYI on a Network Management Tool Catalog: Tools for Monitoring and Debugging TCP/IP Internets and Interconnected Devices" (June 1993)

A few of the tools that this document [RFC1470] describes are still maintained and in use today; for example, `ttcp` and `tcpdump`. However, many of the tools described do not relate specifically to TCP and are no longer used or easily available.

RFC 2398 I: "Some Testing Tools for TCP Implementors" (August 1998)

This document [RFC2398] describes a number of TCP packet generation and analysis tools. Although some of these tools are no longer readily available or widely used, for the most part they are still relevant and useable.

RFC 4614 I: "A Roadmap for Transmission Control Protocol (TCP) Specification Documents" (September 2006)

RFC 4614 [RFC4614] is the precursor of this document.

RFC 5783 I: "Congestion Control in the RFC Series" (February 2010)

This document [RFC5783] provides an overview of RFCs related to congestion control that have been published so far. The focus of the document are on end-host-based congestion control.

RFC 6077 I: "Open Research Issues in Internet Congestion Control" (January 2011)

This RFC [RFC6077] summarizes the main open problems in the domain of Internet congestion control. As a good starting point for newcomers, the document describes several new challenges that are becoming important as the network grows, as well as some issues that have been known for many years.

## 6.8. Case Studies

## RFC 700 U: "A Protocol Experiment" (August 1974)

This document [RFC0700] presents a field report about the deployment of a very early version of TCP, the so-called INWN #39 protocol, which is originally described by Cerf and Kahn in INWG Note #39 [CK73] to use a PDP-11 line printer via the ARPANET.

## RFC 889 U: "Internet Delay Experiments" (December 1983)

This document [RFC0889] is a status report about experiments concerning the TCP retransmission timeout calculation and also provides advices for implementers.

## RFC 1337 I: "TIME-WAIT Assassination Hazards in TCP" (May 1992)

This document [RFC1337] points out a problem with acting on received reset segments while one is in the TIME-WAIT state. The main recommendation is that hosts in TIME-WAIT ignore resets. This recommendation might not currently be widely implemented.

## RFC 2415 I: "Simulation Studies of Increased Initial TCP Window Size" (September 1998)

This document [RFC2415] presents results of some simulations using TCP initial windows greater than 1 segment. The analysis indicates that user-perceived performance can be improved by increasing the initial window to 3 segments.

## RFC 2416 I: "When TCP Starts Up With Four Packets Into Only Three Buffers" (September 1998)

This document [RFC2416] uses simulation results to clear up some concerns about using an initial window of 4 segments when the network path has less provisioning.

## RFC 2884 I: "Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks" (July 2000)

This document [RFC2884] describes experimental results that show some improvements to the performance of both short- and long-lived connections due to ECN.

## 7. Undocumented TCP Features

There are a few important implementation tactics for the TCP that have not yet been described in any RFC. Although this roadmap is

primarily concerned with mapping the TCP RFCs, this section is included because an implementer needs to be aware of these important issues.

#### Header Prediction

Header prediction is a trick to speed up the processing of segments. Van Jacobson and Mike Karels developed the technique in the late 1980s. The basic idea is that some processing time can be saved when most of a segment's fields can be predicted from previous segments. A good description of this was sent to the TCP-IP mailing list by Van Jacobson on March 9, 1988:

"Quite a bit of the speedup comes from an algorithm that we ('we' refers to collaborator Mike Karels and myself) are calling "header prediction". The idea is that if you're in the middle of a bulk data transfer and have just seen a packet, you know what the next packet is going to look like: It will look just like the current packet with either the sequence number or ack number updated (depending on whether you're the sender or receiver). Combining this with the "Use hints" epigram from Butler Lampson's classic "Epigrams for System Designers", you start to think of the tcp state (rcv.nxt, snd.una, etc.) as "hints" about what the next packet should look like.

If you arrange those "hints" so they match the layout of a tcp packet header, it takes a single 14-byte compare to see if your prediction is correct (3 longword compares to pick up the send & ack sequence numbers, header length, flags and window, plus a short compare on the length). If the prediction is correct, there's a single test on the length to see if you're the sender or receiver followed by the appropriate processing. E.g., if the length is non-zero (you're the receiver), checksum and append the data to the socket buffer then wake any process that's sleeping on the buffer. Update rcv.nxt by the length of this packet (this updates your "prediction" of the next packet). Check if you can handle another packet the same size as the current one. If not, set one of the unused flag bits in your header prediction to guarantee that the prediction will fail on the next packet and force you to go through full protocol processing. Otherwise, you're done with this packet. So, the \*total\* tcp protocol processing, exclusive of checksumming, is on the order of 6 compares and an add."

#### Forward Acknowledgement (FACK)

FACK [MM96] describes an alternate algorithm for triggering fast retransmit, based on the extent of the SACK scoreboard. Its goal

is to trigger fast retransmit as soon as the receiver's reassembly queue is larger than the DUPACK threshold, as indicated by the difference between the forward most SACK block edge and SND.UNA. This algorithm quickly and reliably triggers fast retransmit in the presence of burst losses -- often on the first SACK following such a loss. Such a threshold based algorithm also triggers fast retransmit immediately in the presence of any reordering with extent greater than the DUPACK threshold. FACK is implemented in Linux and turned on per default.

#### Highspeed Congestion Control

In the last decade significant research effort has been put into experimental TCP congestion control modifications for obtaining high throughput with reduced startup and recovery times. Only few RFCs have been published on some of these modifications, including HighSpeed TCP [RFC3649], Limited Slow-Start [RFC3742], and Quick-Start [RFC4782] (see Section 4.2), but high-rate congestion control mechanisms are still considered an open issue in congestion control research. Some other schemes have been published as Internet-Drafts, e.g. CUBIC [I-D.rhee-tcpm-cubic] (the standard TCP congestion control algorithm in Linux), Compound TCP [I-D.sridharan-tcpm-ctcp], and H-TCP [I-D.leith-tcp-htcp] or have been discussed a little by the IETF, but much of the work in this area has not been adopted within the IETF yet, so the majority of this work is outside the RFC series and may be discussed in other products of the IRTF Internet Congestion Control Research Group (ICCRG).

#### 8. 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.

#### 9. IANA Considerations

This document contains no IANA considerations.

#### 10. Acknowledgments

This document grew out of a discussion on the end2end-interest mailing list, the public list of the End-to-End Research Group of the IRTF, and continued development under the IETF's TCP Maintenance and Minor Extensions (TCPM) working group. We thank Joe Touch, Reiner



Ludwig, Pekka Savola, Gorrry Fairhurst, and Sally Floyd for their contributions, in particular. The chairs of the TCPM working group, Mark Allman and Ted Faber, have been instrumental in the development of this document. Keith McCloghrie provided some useful notes and clarification on the various MIB-related RFCs.

## 11. References

### 11.1. Normative References

- [RFC0675] Cerf, V., Dalal, Y., and C. Sunshine, "Specification of Internet Transmission Control Program", RFC 675, December 1974.
- [RFC0700] Mader, E., Plummer, W., and R. Tomlinson, "Protocol experiment", RFC 700, August 1974.
- [RFC0721] Garlick, L., "Out-of-Band Control Signals in a Host-to-Host Protocol", RFC 721, September 1976.
- [RFC0761] Postel, J., "DoD standard Transmission Control Protocol", RFC 761, January 1980.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
- [RFC0794] Cerf, V., "Pre-emption", RFC 794, September 1981.
- [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.
- [RFC0889] Mills, D., "Internet delay experiments", RFC 889, December 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.
- [RFC1066] McCloghrie, K. and M. Rose, "Management Information Base for network management of TCP/IP-based internets", RFC 1066, August 1988.
- [RFC1071] Braden, R., Borman, D., Partridge, C., and W. Plummer, "Computing the Internet checksum", RFC 1071, September 1988.
- [RFC1078] Lottor, M., "TCP port service Multiplexer (TCPMUX)", RFC 1078, November 1988.
- [RFC1106] Fox, R., "TCP big window and NAK options", RFC 1106, June 1989.
- [RFC1110] McKenzie, A., "Problem with the TCP big window option", RFC 1110, August 1989.
- [RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.
- [RFC1144] Jacobson, V., "Compressing TCP/IP headers for low-speed serial links", RFC 1144, February 1990.
- [RFC1146] Zweig, J. and C. Partridge, "TCP alternate checksum options", RFC 1146, March 1990.
- [RFC1156] McCloghrie, K. and M. Rose, "Management Information Base for network management of TCP/IP-based internets", RFC 1156, May 1990.
- [RFC1180] Socolofsky, T. and C. Kale, "TCP/IP tutorial", RFC 1180, January 1991.
- [RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, November 1990.
- [RFC1213] McCloghrie, K. and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets:MIB-II", STD 17, RFC 1213, March 1991.
- [RFC1263] O'Malley, S. and L. Peterson, "TCP Extensions Considered

Harmful", RFC 1263, October 1991.

- [RFC1323] Jacobson, V., Braden, B., and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.
- [RFC1337] Braden, B., "TIME-WAIT Assassination Hazards in TCP", RFC 1337, May 1992.
- [RFC1379] Braden, B., "Extending TCP for Transactions -- Concepts", RFC 1379, November 1992.
- [RFC1470] Enger, R. and J. Reynolds, "FYI on a Network Management Tool Catalog: Tools for Monitoring and Debugging TCP/IP Internets and Interconnected Devices", RFC 1470, June 1993.
- [RFC1624] Rijssinghani, A., "Computation of the Internet Checksum via Incremental Update", RFC 1624, May 1994.
- [RFC1644] Braden, B., "T/TCP -- TCP Extensions for Transactions Functional Specification", RFC 1644, July 1994.
- [RFC1693] Connolly, T., Amer, P., and P. Conrad, "An Extension to TCP : Partial Order Service", RFC 1693, November 1994.
- [RFC1705] Carlson, R. and D. Ficarella, "Six Virtual Inches to the Left: The Problem with IPng", RFC 1705, October 1994.
- [RFC1936] Touch, J. and B. Parham, "Implementing the Internet Checksum in Hardware", RFC 1936, April 1996.
- [RFC1958] Carpenter, B., "Architectural Principles of the Internet", RFC 1958, June 1996.
- [RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, August 1996.
- [RFC2012] McCloaghrie, K., "SNMPv2 Management Information Base for the Transmission Control Protocol using SMIV2", RFC 2012, November 1996.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.
- [RFC2140] Touch, J., "TCP Control Block Interdependence", RFC 2140, April 1997.
- [RFC2398] Parker, S. and C. Schmechel, "Some Testing Tools for TCP

Implementors", RFC 2398, August 1998.

- [RFC2415] Poduri, K., "Simulation Studies of Increased Initial TCP Window Size", RFC 2415, September 1998.
- [RFC2416] Shepard, T. and C. Partridge, "When TCP Starts Up With Four Packets Into Only Three Buffers", RFC 2416, September 1998.
- [RFC2452] Daniele, M., "IP Version 6 Management Information Base for the Transmission Control Protocol", RFC 2452, December 1998.
- [RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.
- [RFC2488] Allman, M., Glover, D., and L. Sanchez, "Enhancing TCP Over Satellite Channels using Standard Mechanisms", BCP 28, RFC 2488, January 1999.
- [RFC2525] Paxson, V., Dawson, S., Fenner, W., Griner, J., Heavens, I., Lahey, K., Semke, J., and B. Volz, "Known TCP Implementation Problems", RFC 2525, March 1999.
- [RFC2675] Borman, D., Deering, S., and R. Hinden, "IPv6 Jumbograms", RFC 2675, August 1999.
- [RFC2757] Montenegro, G., Dawkins, S., Kojo, M., Magret, V., and N. Vaidya, "Long Thin Networks", RFC 2757, January 2000.
- [RFC2760] Allman, M., Dawkins, S., Glover, D., Griner, J., Tran, D., Henderson, T., Heidemann, J., Touch, J., Kruse, H., Ostermann, S., Scott, K., and J. Semke, "Ongoing TCP Research Related to Satellites", RFC 2760, February 2000.
- [RFC2861] Handley, M., Padhye, J., and S. Floyd, "TCP Congestion Window Validation", RFC 2861, June 2000.
- [RFC2873] Xiao, X., Hannan, A., Paxson, V., and E. Crabbe, "TCP Processing of the IPv4 Precedence Field", RFC 2873, June 2000.
- [RFC2883] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, July 2000.
- [RFC2884] Hadi Salim, J. and U. Ahmed, "Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks",

RFC 2884, July 2000.

- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, September 2000.
- [RFC2923] Lahey, K., "TCP Problems with Path MTU Discovery", RFC 2923, September 2000.
- [RFC3042] Allman, M., Balakrishnan, H., and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, January 2001.
- [RFC3124] Balakrishnan, H. and S. Seshan, "The Congestion Manager", RFC 3124, June 2001.
- [RFC3135] Border, J., Kojo, M., Griner, J., Montenegro, G., and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations", RFC 3135, June 2001.
- [RFC3150] Dawkins, S., Montenegro, G., Kojo, M., and V. Magret, "End-to-end Performance Implications of Slow Links", BCP 48, RFC 3150, July 2001.
- [RFC3155] Dawkins, S., Montenegro, G., Kojo, M., Magret, V., and N. Vaidya, "End-to-end Performance Implications of Links with Errors", BCP 50, RFC 3155, August 2001.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.
- [RFC3360] Floyd, S., "Inappropriate TCP Resets Considered Harmful", BCP 60, RFC 3360, August 2002.
- [RFC3366] Fairhurst, G. and L. Wood, "Advice to link designers on link Automatic Repeat reQuest (ARQ)", BCP 62, RFC 3366, August 2002.
- [RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, October 2002.
- [RFC3439] Bush, R. and D. Meyer, "Some Internet Architectural Guidelines and Philosophy", RFC 3439, December 2002.
- [RFC3449] Balakrishnan, H., Padmanabhan, V., Fairhurst, G., and M. Sooriyabandara, "TCP Performance Implications of Network Path Asymmetry", BCP 69, RFC 3449, December 2002.

- [RFC3465] Allman, M., "TCP Congestion Control with Appropriate Byte Counting (ABC)", RFC 3465, February 2003.
- [RFC3481] Inamura, H., Montenegro, G., Ludwig, R., Gurtov, A., and F. Khafizov, "TCP over Second (2.5G) and Third (3G) Generation Wireless Networks", BCP 71, RFC 3481, February 2003.
- [RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, February 2003.
- [RFC3522] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC 3522, April 2003.
- [RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, June 2003.
- [RFC3649] Floyd, S., "HighSpeed TCP for Large Congestion Windows", RFC 3649, December 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.
- [RFC3742] Floyd, S., "Limited Slow-Start for TCP with Large Congestion Windows", RFC 3742, March 2004.
- [RFC3819] Karn, P., Bormann, C., Fairhurst, G., Grossman, D., Ludwig, R., Mahdavi, J., Montenegro, G., Touch, J., and L. Wood, "Advice for Internet Subnetwork Designers", BCP 89, RFC 3819, July 2004.
- [RFC4015] Ludwig, R. and A. Gurtov, "The Eifel Response Algorithm for TCP", RFC 4015, February 2005.
- [RFC4022] Raghunarayan, R., "Management Information Base for the Transmission Control Protocol (TCP)", RFC 4022, March 2005.
- [RFC4614] Duke, M., Braden, R., Eddy, W., and E. Blanton, "A Roadmap for Transmission Control Protocol (TCP) Specification Documents", RFC 4614, September 2006.
- [RFC4653] Bhandarkar, S., Reddy, A., Allman, M., and E. Blanton,

- "Improving the Robustness of TCP to Non-Congestion Events", RFC 4653, August 2006.
- [RFC4774] Floyd, S., "Specifying Alternate Semantics for the Explicit Congestion Notification (ECN) Field", BCP 124, RFC 4774, November 2006.
- [RFC4782] Floyd, S., Allman, M., Jain, A., and P. Sarolahti, "Quick-Start for TCP and IP", RFC 4782, January 2007.
- [RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.
- [RFC4953] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, July 2007.
- [RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, August 2007.
- [RFC5033] Floyd, S. and M. Allman, "Specifying New Congestion Control Algorithms", BCP 133, RFC 5033, August 2007.
- [RFC5166] Floyd, S., "Metrics for the Evaluation of Congestion Control Mechanisms", RFC 5166, March 2008.
- [RFC5461] Gont, F., "TCP's Reaction to Soft Errors", RFC 5461, February 2009.
- [RFC5482] Eggert, L. and F. Gont, "TCP User Timeout Option", RFC 5482, March 2009.
- [RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", RFC 5562, June 2009.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.
- [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.
- [RFC5690] Floyd, S., Arcia, A., Ros, D., and J. Iyengar, "Adding Acknowledgement Congestion Control to TCP", RFC 5690, February 2010.

- [RFC5783] Welzl, M. and W. Eddy, "Congestion Control in the RFC Series", RFC 5783, February 2010.
- [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.
- [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, June 2010.
- [RFC5926] Lebovitz, G. and E. Rescorla, "Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)", RFC 5926, June 2010.
- [RFC5927] Gont, F., "ICMP Attacks against TCP", RFC 5927, July 2010.
- [RFC5961] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", RFC 5961, August 2010.
- [RFC6013] Simpson, W., "TCP Cookie Transactions (TCPCT)", RFC 6013, January 2011.
- [RFC6056] Larsen, M. and F. Gont, "Recommendations for Transport-Protocol Port Randomization", BCP 156, RFC 6056, January 2011.
- [RFC6069] Zimmermann, A. and A. Hannemann, "Making TCP More Robust to Long Connectivity Disruptions (TCP-LCD)", RFC 6069, December 2010.
- [RFC6077] Papadimitriou, D., Welzl, M., Scharf, M., and B. Briscoe, "Open Research Issues in Internet Congestion Control", RFC 6077, February 2011.
- [RFC6093] Gont, F. and A. Yourtchenko, "On the Implementation of the TCP Urgent Mechanism", RFC 6093, January 2011.
- [RFC6181] Bagnulo, M., "Threat Analysis for TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6181, March 2011.
- [RFC6182] Ford, A., Raiciu, C., Handley, M., Barre, S., and J. Iyengar, "Architectural Guidelines for Multipath TCP Development", RFC 6182, March 2011.
- [RFC6191] Gont, F., "Reducing the TIME-WAIT State Using TCP Timestamps", BCP 159, RFC 6191, April 2011.



- [RFC6247] Eggert, L., "Moving the Undeployed TCP Extensions RFC 1072, RFC 1106, RFC 1110, RFC 1145, RFC 1146, RFC 1379, RFC 1644, and RFC 1693 to Historic Status", RFC 6247, May 2011.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.
- [RFC6349] Constantine, B., Forget, G., Geib, R., and R. Schrage, "Framework for TCP Throughput Testing", RFC 6349, August 2011.
- [RFC6356] Raiciu, C., Handley, M., and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols", RFC 6356, October 2011.
- [RFC6429] Bashyam, M., Jethanandani, M., and A. Ramaiah, "TCP Sender Clarification for Persist Condition", RFC 6429, December 2011.
- [RFC6528] Gont, F. and S. Bellovin, "Defending against Sequence Number Attacks", RFC 6528, February 2012.
- [RFC6582] Henderson, T., Floyd, S., Gurtov, A., and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 6582, April 2012.
- [RFC6633] Gont, F., "Deprecation of ICMP Source Quench Messages", RFC 6633, May 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.
- [RFC6691] Borman, D., "TCP Options and Maximum Segment Size (MSS)", RFC 6691, July 2012.
- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, January 2013.
- [RFC6846] Pelletier, G., Sandlund, K., Jonsson, L-E., and M. West, "RObust Header Compression (ROHC): A Profile for TCP/IP (ROHC-TCP)", RFC 6846, January 2013.
- [RFC6897] Scharf, M. and A. Ford, "Multipath TCP (MPTCP) Application

Interface Considerations", RFC 6897, March 2013.

[RFC6928] Chu, J., Dukkkipati, N., Cheng, Y., and M. Mathis,  
"Increasing TCP's Initial Window", RFC 6928, April 2013.

[RFC6937] Mathis, M., Dukkkipati, N., and Y. Cheng, "Proportional  
Rate Reduction for TCP", RFC 6937, May 2013.

## 11.2. Informative References

[CK73] Cerf, V. and R. Kahn, "Towards Protocols for Internetwork  
Communication", IFIP/TC6.1, NIC 18764, INWG 39,  
September 1973.

[Errata] "RFC Editor - RFC Errata",  
<<http://www.rfc-editor.org/errata.php>>.

[I-D.leith-tcp-htcp]  
Leith, D., "H-TCP: TCP Congestion Control for High  
Bandwidth-Delay Product Paths", draft-leith-tcp-htcp-06  
(work in progress), April 2008.

[I-D.rhee-tcpm-cubic]  
Rhee, I., Xu, L., and S. Ha, "CUBIC for Fast Long-Distance  
Networks", draft-rhee-tcpm-cubic-02 (work in progress),  
August 2008.

[I-D.sridharan-tcpm-ctcp]  
Sridharan, M., Tan, K., Bansal, D., and D. Thaler,  
"Compound TCP: A New TCP Congestion Control for High-Speed  
and Long Distance Networks", draft-sridharan-tcpm-ctcp-02  
(work in progress), November 2008.

[JK92] Jacobson, V. and M. Karels, "Congestion Avoidance and  
Control", This paper is a revised version of [Jac88], that  
includes an additional appendix. This paper has not been  
traditionally published, but is currently available at  
<ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z>. 1992.

[Jac88] Jacobson, V., "Congestion Avoidance and Control", ACM  
SIGCOMM 1988 Proceedings, in ACM Computer Communication  
Review, 18 (4), pp. 314-329, August 1988.

[KP87] Karn, P. and C. Partridge, "Round Trip Time Estimation",  
ACM SIGCOMM 1987 Proceedings, in ACM Computer  
Communication Review, 17 (5), pp. 2-7, August 1987.

[MAF04] Medina, A., Allman, M., and S. Floyd, "Measuring the

Evolution of Transport Protocols in the Internet", ACM Computer Communication Review, 35 (2), April 2005.

- [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.
- [RFC1016] Prue, W. and J. Postel, "Something a host could do with source quench: The Source Quench Introduced Delay (SQuID)", RFC 1016, July 1987.
- [RFC2026] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.
- [RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998.
- [RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, March 2006.
- [RFC4341] Floyd, S. and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control", RFC 4341, March 2006.
- [SCWA99] Savage, S., Cardwell, N., Wetherall, D., and T. Anderson, "TCP Congestion Control with a Misbehaving Receiver", ACM Computer Communication Review, 29 (5), pp. 71-78, October 1999.

#### Authors' Addresses

Martin Duke  
Boeing Phantom Works  
PO Box 3707, MC 7L-49  
Seattle, WA 98124-2207  
  
Phone: 425-865-1182  
Email: martin.duke@boeing.com

Robert Braden  
USC Information Sciences Institute  
Marina del Rey, CA 90292-6695  
  
Phone: 310-448-9173  
Email: braden@isi.edu

Wesley M. Eddy  
MTI Systems  
MS 500-ASRC; 21000 Brookpark Rd  
Cleveland, OH 44135  
  
Phone: 216-433-6682  
Email: wes@mti-systems.com

Ethan Blanton  
Purdue University Computer Science  
305 N. University St.  
West Lafayette, IN 47907  
  
Email: elb@psg.com

Alexander Zimmermann  
NetApp, Inc.  
Sonnenallee 1  
Kirchheim 85551  
Germany  
  
Email: alexander.zimmermann@netapp.com

