

Internet Engineering Task Force
Internet-Draft
Obsoletes: 793, 879, 6093, 6528, 6691
 (if approved)
Updates: 1122 (if approved)
Intended status: Standards Track
Expires: August 10, 2015

W. Eddy, Ed.
MTI Systems
February 6, 2015

Transmission Control Protocol Specification
draft-eddy-rfc793bis-05

Abstract

This document specifies the Internet's Transmission Control Protocol (TCP). TCP is an important transport layer protocol in the Internet stack, and has continuously evolved over decades of use and growth of the Internet. Over this time, a number of changes have been made to TCP as it was specified in RFC 793, though these have only been documented in a piecemeal fashion. This document collects and brings those changes together with the protocol specification from RFC 793. This document obsoletes RFC 793 and several other RFCs (TODO: list all actual RFCs when finished).

RFC EDITOR NOTE: If approved for publication as an RFC, this should be marked additionally as "STD: 7" and replace RFC 793 in that role.

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 [1].

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 10, 2015.

Copyright Notice

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

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

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Purpose and Scope	3
2. Introduction	4
3. Functional Specification	4
3.1. Header Format	4
3.2. Terminology	9
3.3. Sequence Numbers	13
3.4. Establishing a connection	20
3.5. Closing a Connection	27
3.6. Precedence and Security	29
3.7. Segmentation	30
3.7.1. Maximum Segment Size Option	31
3.7.2. Path MTU Discovery	32
3.7.3. Interfaces with Variable MSS Values	32
3.7.4. IPv6 Jumbograms	32
3.8. Data Communication	32
3.9. Interfaces	36
3.9.1. User/TCP Interface	37

3.9.2. TCP/Lower-Level Interface	43
3.10. Event Processing	44
3.11. Glossary	67
4. Changes from RFC 793	72
5. IANA Considerations	75
6. Security and Privacy Considerations	75
7. Acknowledgements	76
8. References	76
8.1. Normative References	76
8.2. Informative References	76
Appendix A. TCP Requirement Summary	77
Author's Address	80

1. Purpose and Scope

In 1981, RFC 793 [2] was released, documenting the Transmission Control Protocol (TCP), and replacing earlier specifications for TCP that had been published in the past.

Since then, TCP has been implemented many times, and has been used as a transport protocol for numerous applications on the Internet.

For several decades, RFC 793 plus a number of other documents have combined to serve as the specification for TCP [10]. Over time, a number of errata have been identified on RFC 793, as well as deficiencies in security, performance, and other aspects. A number of enhancements has grown and been documented separately. These were never accumulated together into an update to the base specification.

The purpose of this document is to bring together all of the IETF Standards Track changes that have been made to the basic TCP functional specification and unify them into an update of the RFC 793 protocol specification. Some companion documents are referenced for important algorithms that TCP uses (e.g. for congestion control), but have not been attempted to include in this document. This is a conscious choice, as this base specification can be used with multiple additional algorithms that are developed and incorporated separately, but all TCP implementations need to implement this specification as a common basis in order to interoperate. As some additional TCP features have become quite complicated themselves (e.g. advanced loss recovery and congestion control), future companion documents may attempt to similarly bring these together.

In addition to the protocol specification that describes the TCP segment format, generation, and processing rules that are to be implemented in code, RFC 793 and other updates also contain informative and descriptive text for human readers to understand aspects of the protocol design and operation. This document does not

attempt to alter or update this informative text, and is focused only on updating the normative protocol specification. We preserve references to the documentation containing the important explanations and rationale, where appropriate.

This document is intended to be useful both in checking existing TCP implementations for conformance, as well as in writing new implementations.

2. Introduction

RFC 793 contains a discussion of the TCP design goals and provides examples of its operation, including examples of connection establishment, closing connections, and retransmitting packets to repair losses.

This document describes the basic functionality expected in modern implementations of TCP, and replaces the protocol specification in RFC 793. It does not replicate or attempt to update the examples and other discussion in RFC 793. Other documents are referenced to provide explanation of the theory of operation, rationale, and detailed discussion of design decisions. This document only focuses on the normative behavior of the protocol.

TEMPORARY EDITOR'S NOTE: This is an early revision in the process of updating RFC 793. Many planned changes are not yet incorporated.

Please do not use this revision as a basis for any work or reference.

A list of changes from RFC 793 is contained in Section 4.

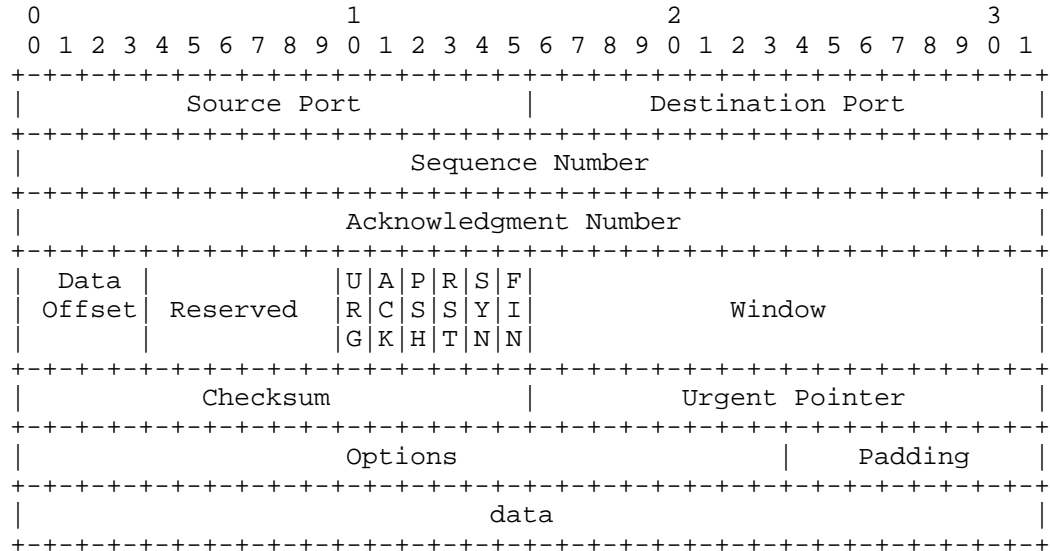
TEMPORARY EDITOR'S NOTE: the current revision of this document does not yet collect all of the changes that will be in the final version. The set of content changes planned for future revisions is kept in Section 4.

3. Functional Specification

3.1. Header Format

TCP segments are sent as internet datagrams. The Internet Protocol header carries several information fields, including the source and destination host addresses [2]. A TCP header follows the internet header, supplying information specific to the TCP protocol. This division allows for the existence of host level protocols other than TCP.

TCP Header Format



TCP Header Format

Note that one tick mark represents one bit position.

Figure 1

Source Port: 16 bits

The source port number.

Destination Port: 16 bits

The destination port number.

Sequence Number: 32 bits

The sequence number of the first data octet in this segment (except when SYN is present). If SYN is present the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.

Acknowledgment Number: 32 bits

If the ACK control bit is set this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent.

Data Offset: 4 bits

The number of 32 bit words in the TCP Header. This indicates where the data begins. The TCP header (even one including options) is an integral number of 32 bits long.

Reserved: 6 bits

Reserved for future use. Must be zero.

Control Bits: 6 bits (from left to right):

- URG: Urgent Pointer field significant
- ACK: Acknowledgment field significant
- PSH: Push Function
- RST: Reset the connection
- SYN: Synchronize sequence numbers
- FIN: No more data from sender

Window: 16 bits

The number of data octets beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept.

Checksum: 16 bits

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16 bit word for checksum purposes. The pad is not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros.

The checksum also covers a 96 bit pseudo header conceptually prefixed to the TCP header. This pseudo header contains the Source Address, the Destination Address, the Protocol, and TCP length. This gives the TCP protection against misrouted segments. This information is carried in the Internet Protocol and is transferred across the TCP/Network interface in the arguments or results of calls by the TCP on the IP.

+-----+-----+-----+-----+			
	Source Address		
+-----+-----+-----+-----+			
	Destination Address		
+-----+-----+-----+-----+			
	zero	PTCL	TCP Length
+-----+-----+-----+-----+			

The TCP Length is the TCP header length plus the data length in octets (this is not an explicitly transmitted quantity, but is computed), and it does not count the 12 octets of the pseudo header.

Urgent Pointer: 16 bits

This field communicates the current value of the urgent pointer as a positive offset from the sequence number in this segment. The urgent pointer points to the sequence number of the octet following the urgent data. This field is only be interpreted in segments with the URG control bit set.

Options: variable

Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum. An option may begin on any octet boundary. There are two cases for the format of an option:

Case 1: A single octet of option-kind.

Case 2: An octet of option-kind, an octet of option-length, and the actual option-data octets.

The option-length counts the two octets of option-kind and option-length as well as the option-data octets.

Note that the list of options may be shorter than the data offset field might imply. The content of the header beyond the End-of-Option option must be header padding (i.e., zero).

Currently defined options include (kind indicated in octal):

Kind	Length	Meaning
----	-----	-----
0	-	End of option list.
1	-	No-Operation.
2	4	Maximum Segment Size.

A TCP MUST be able to receive a TCP option in any segment. A TCP MUST ignore without error any TCP option it does not implement, assuming that the option has a length field (all TCP options except End of option list and No-Operation have length fields). TCP MUST be prepared to handle an illegal option length (e.g., zero) without crashing; a suggested procedure is to reset the connection and log the reason.

Specific Option Definitions

End of Option List

```
+-----+
|00000000|
+-----+
Kind=0
```

This option code indicates the end of the option list. This might not coincide with the end of the TCP header according to the Data Offset field. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the TCP header.

No-Operation

```
+-----+
|00000001|
+-----+
Kind=1
```

This option code may be used between options, for example, to align the beginning of a subsequent option on a word boundary. There is no guarantee that senders will use this option, so receivers must be prepared to process options even if they do not begin on a word boundary.

Maximum Segment Size (MSS)

```
+-----+-----+-----+-----+
|00000010|00000100|   max seg size   |
+-----+-----+-----+-----+
Kind=2   Length=4
```

Maximum Segment Size Option Data: 16 bits

If this option is present, then it communicates the maximum receive segment size at the TCP which sends this segment. This

field may be sent in the initial connection request (i.e., in segments with the SYN control bit set) and must not be sent in other segments. If this option is not used, any segment size is allowed.

Padding: variable

The TCP header padding is used to ensure that the TCP header ends and data begins on a 32 bit boundary. The padding is composed of zeros.

3.2. Terminology

Before we can discuss very much about the operation of the TCP we need to introduce some detailed terminology. The maintenance of a TCP connection requires the remembering of several variables. We conceive of these variables being stored in a connection record called a Transmission Control Block or TCB. Among the variables stored in the TCB are the local and remote socket numbers, the security and precedence of the connection, pointers to the user's send and receive buffers, pointers to the retransmit queue and to the current segment. In addition several variables relating to the send and receive sequence numbers are stored in the TCB.

Send Sequence Variables

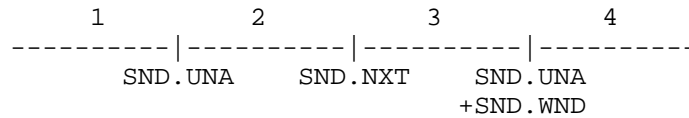
- SND.UNA - send unacknowledged
- SND.NXT - send next
- SND.WND - send window
- SND.UP - send urgent pointer
- SND.WL1 - segment sequence number used for last window update
- SND.WL2 - segment acknowledgment number used for last window update
- ISS - initial send sequence number

Receive Sequence Variables

- RCV.NXT - receive next
- RCV.WND - receive window
- RCV.UP - receive urgent pointer
- IRS - initial receive sequence number

The following diagrams may help to relate some of these variables to the sequence space.

Send Sequence Space



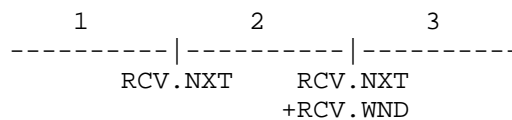
- 1 - old sequence numbers which have been acknowledged
- 2 - sequence numbers of unacknowledged data
- 3 - sequence numbers allowed for new data transmission
- 4 - future sequence numbers which are not yet allowed

Send Sequence Space

Figure 2

The send window is the portion of the sequence space labeled 3 in Figure 2.

Receive Sequence Space



- 1 - old sequence numbers which have been acknowledged
- 2 - sequence numbers allowed for new reception
- 3 - future sequence numbers which are not yet allowed

Receive Sequence Space

Figure 3

The receive window is the portion of the sequence space labeled 2 in Figure 3.

There are also some variables used frequently in the discussion that take their values from the fields of the current segment.

Current Segment Variables

- SEG.SEQ - segment sequence number
- SEG.ACK - segment acknowledgment number
- SEG.LEN - segment length
- SEG.WND - segment window
- SEG.UP - segment urgent pointer
- SEG.PRC - segment precedence value

A connection progresses through a series of states during its lifetime. The states are: LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, and the fictional state CLOSED. CLOSED is fictional because it represents the state when there is no TCB, and therefore, no connection. Briefly the meanings of the states are:

LISTEN - represents waiting for a connection request from any remote TCP and port.

SYN-SENT - represents waiting for a matching connection request after having sent a connection request.

SYN-RECEIVED - represents waiting for a confirming connection request acknowledgment after having both received and sent a connection request.

ESTABLISHED - represents an open connection, data received can be delivered to the user. The normal state for the data transfer phase of the connection.

FIN-WAIT-1 - represents waiting for a connection termination request from the remote TCP, or an acknowledgment of the connection termination request previously sent.

FIN-WAIT-2 - represents waiting for a connection termination request from the remote TCP.

CLOSE-WAIT - represents waiting for a connection termination request from the local user.

CLOSING - represents waiting for a connection termination request acknowledgment from the remote TCP.

LAST-ACK - represents waiting for an acknowledgment of the connection termination request previously sent to the remote TCP (this termination request sent to the remote TCP already included an acknowledgment of the termination request sent from the remote TCP).

TIME-WAIT - represents waiting for enough time to pass to be sure the remote TCP received the acknowledgment of its connection termination request.

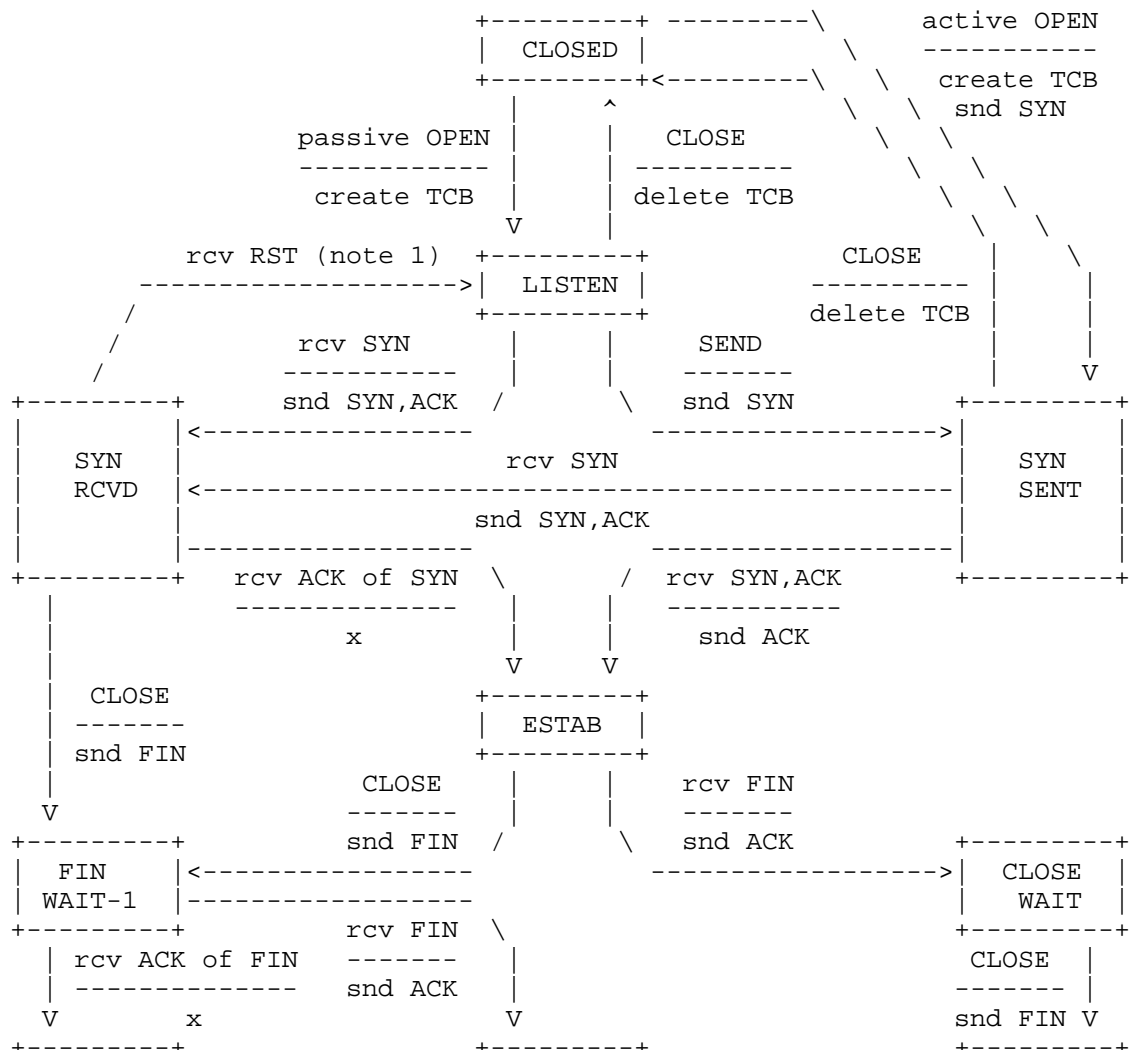
CLOSED - represents no connection state at all.

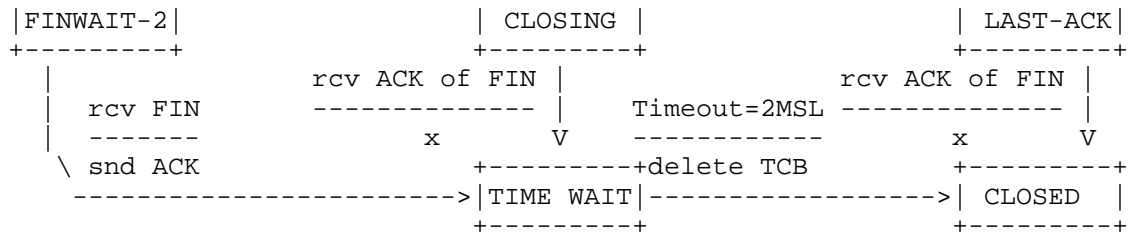
A TCP connection progresses from one state to another in response to events. The events are the user calls, OPEN, SEND, RECEIVE, CLOSE,

ABORT, and STATUS; the incoming segments, particularly those containing the SYN, ACK, RST and FIN flags; and timeouts.

The state diagram in Figure 4 illustrates only state changes, together with the causing events and resulting actions, but addresses neither error conditions nor actions which are not connected with state changes. In a later section, more detail is offered with respect to the reaction of the TCP to events.

NOTA BENE: this diagram is only a summary and must not be taken as the total specification.





note 1: The transition from SYN-RCVD to LISTEN on receiving a RST is conditional on having reached SYN-RCVD after a passive open.

note 2: An unshown transition exists from FIN-WAIT-1 to TIME-WAIT if a FIN is received and the local FIN is also acknowledged.

TCP Connection State Diagram

Figure 4

3.3. Sequence Numbers

A fundamental notion in the design is that every octet of data sent over a TCP connection has a sequence number. Since every octet is sequenced, each of them can be acknowledged. The acknowledgment mechanism employed is cumulative so that an acknowledgment of sequence number X indicates that all octets up to but not including X have been received. This mechanism allows for straight-forward duplicate detection in the presence of retransmission. Numbering of octets within a segment is that the first data octet immediately following the header is the lowest numbered, and the following octets are numbered consecutively.

It is essential to remember that the actual sequence number space is finite, though very large. This space ranges from 0 to $2^{32} - 1$. Since the space is finite, all arithmetic dealing with sequence numbers must be performed modulo 2^{32} . This unsigned arithmetic preserves the relationship of sequence numbers as they cycle from $2^{32} - 1$ to 0 again. There are some subtleties to computer modulo arithmetic, so great care should be taken in programming the comparison of such values. The symbol " $=<$ " means "less than or equal" (modulo 2^{32}).

The typical kinds of sequence number comparisons which the TCP must perform include:

- (a) Determining that an acknowledgment refers to some sequence number sent but not yet acknowledged.

(b) Determining that all sequence numbers occupied by a segment have been acknowledged (e.g., to remove the segment from a retransmission queue).

(c) Determining that an incoming segment contains sequence numbers which are expected (i.e., that the segment "overlaps" the receive window).

In response to sending data the TCP will receive acknowledgments. The following comparisons are needed to process the acknowledgments.

SND.UNA = oldest unacknowledged sequence number

SND.NXT = next sequence number to be sent

SEG.ACK = acknowledgment from the receiving TCP (next sequence number expected by the receiving TCP)

SEG.SEQ = first sequence number of a segment

SEG.LEN = the number of octets occupied by the data in the segment (counting SYN and FIN)

SEG.SEQ+SEG.LEN-1 = last sequence number of a segment

A new acknowledgment (called an "acceptable ack"), is one for which the inequality below holds:

$$\text{SND.UNA} < \text{SEG.ACK} \leq \text{SND.NXT}$$

A segment on the retransmission queue is fully acknowledged if the sum of its sequence number and length is less or equal than the acknowledgment value in the incoming segment.

When data is received the following comparisons are needed:

RCV.NXT = next sequence number expected on an incoming segments, and is the left or lower edge of the receive window

RCV.NXT+RCV.WND-1 = last sequence number expected on an incoming segment, and is the right or upper edge of the receive window

SEG.SEQ = first sequence number occupied by the incoming segment

SEG.SEQ+SEG.LEN-1 = last sequence number occupied by the incoming segment

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

Note that when the receive window is zero no segments should be acceptable except ACK segments. Thus, it is possible for a TCP to maintain a zero receive window while transmitting data and receiving ACKs. However, even when the receive window is zero, a TCP must process the RST and URG fields of all incoming segments.

We have taken advantage of the numbering scheme to protect certain control information as well. This is achieved by implicitly including some control flags in the sequence space so they can be retransmitted and acknowledged without confusion (i.e., one and only one copy of the control will be acted upon). Control information is not physically carried in the segment data space. Consequently, we must adopt rules for implicitly assigning sequence numbers to control. The SYN and FIN are the only controls requiring this protection, and these controls are used only at connection opening and closing. For sequence number purposes, the SYN is considered to occur before the first actual data octet of the segment in which it

occurs, while the FIN is considered to occur after the last actual data octet in a segment in which it occurs. The segment length (SEG.LEN) includes both data and sequence space occupying controls. When a SYN is present then SEG.SEQ is the sequence number of the SYN.

Initial Sequence Number Selection

The protocol places no restriction on a particular connection being used over and over again. A connection is defined by a pair of sockets. New instances of a connection will be referred to as incarnations of the connection. The problem that arises from this is -- "how does the TCP identify duplicate segments from previous incarnations of the connection?" This problem becomes apparent if the connection is being opened and closed in quick succession, or if the connection breaks with loss of memory and is then reestablished.

To avoid confusion we must prevent segments from one incarnation of a connection from being used while the same sequence numbers may still be present in the network from an earlier incarnation. We want to assure this, even if a TCP crashes and loses all knowledge of the sequence numbers it has been using. When new connections are created, an initial sequence number (ISN) generator is employed which selects a new 32 bit ISN. There are security issues that result if an off-path attacker is able to predict or guess ISN values.

The recommended ISN generator is based on the combination of a (possibly fictitious) 32 bit clock whose low order bit is incremented roughly every 4 microseconds, and a pseudorandom hash function (PRF). The clock component is intended to insure that with a Maximum Segment Lifetime (MSL), generated ISNs will be unique, since it cycles approximately every 4.55 hours, which is much longer than the MSL.

TCP SHOULD generate its Initial Sequence Numbers with the expression:

$$\text{ISN} = M + F(\text{localip}, \text{localport}, \text{remoteip}, \text{remoteport}, \text{secretkey})$$

where M is the 4 microsecond timer, and F() is a pseudorandom function (PRF) of the connection's identifying parameters ("localip, localport, remoteip, remoteport") and a secret key ("secretkey"). F() MUST NOT be computable from the outside, or an attacker could still guess at sequence numbers from the ISN used for some other connection. The PRF could be implemented as a cryptographic hash of the concatenation of the TCP connection parameters and some secret data. For discussion of the selection of a specific hash algorithm and management of the secret key data, please see Section 3 of [8].

For each connection there is a send sequence number and a receive sequence number. The initial send sequence number (ISS) is chosen by

the data sending TCP, and the initial receive sequence number (IRS) is learned during the connection establishing procedure.

For a connection to be established or initialized, the two TCPs must synchronize on each other's initial sequence numbers. This is done in an exchange of connection establishing segments carrying a control bit called "SYN" (for synchronize) and the initial sequence numbers. As a shorthand, segments carrying the SYN bit are also called "SYNs". Hence, the solution requires a suitable mechanism for picking an initial sequence number and a slightly involved handshake to exchange the ISN's.

The synchronization requires each side to send it's own initial sequence number and to receive a confirmation of it in acknowledgment from the other side. Each side must also receive the other side's initial sequence number and send a confirming acknowledgment.

- 1) A --> B SYN my sequence number is X
- 2) A <-- B ACK your sequence number is X
- 3) A <-- B SYN my sequence number is Y
- 4) A --> B ACK your sequence number is Y

Because steps 2 and 3 can be combined in a single message this is called the three way (or three message) handshake.

A three way handshake is necessary because sequence numbers are not tied to a global clock in the network, and TCPs may have different mechanisms for picking the ISN's. The receiver of the first SYN has no way of knowing whether the segment was an old delayed one or not, unless it remembers the last sequence number used on the connection (which is not always possible), and so it must ask the sender to verify this SYN. The three way handshake and the advantages of a clock-driven scheme are discussed in [3].

Knowing When to Keep Quiet

To be sure that a TCP does not create a segment that carries a sequence number which may be duplicated by an old segment remaining in the network, the TCP must keep quiet for a maximum segment lifetime (MSL) before assigning any sequence numbers upon starting up or recovering from a crash in which memory of sequence numbers in use was lost. For this specification the MSL is taken to be 2 minutes. This is an engineering choice, and may be changed if experience indicates it is desirable to do so. Note that if a TCP is reinitialized in some sense, yet retains its memory of sequence numbers in use, then it need not wait at all; it must only be sure to use sequence numbers larger than those recently used.

The TCP Quiet Time Concept

This specification provides that hosts which "crash" without retaining any knowledge of the last sequence numbers transmitted on each active (i.e., not closed) connection shall delay emitting any TCP segments for at least the agreed Maximum Segment Lifetime (MSL) in the internet system of which the host is a part. In the paragraphs below, an explanation for this specification is given. TCP implementors may violate the "quiet time" restriction, but only at the risk of causing some old data to be accepted as new or new data rejected as old duplicated by some receivers in the internet system.

TCPs consume sequence number space each time a segment is formed and entered into the network output queue at a source host. The duplicate detection and sequencing algorithm in the TCP protocol relies on the unique binding of segment data to sequence space to the extent that sequence numbers will not cycle through all 2^{32} values before the segment data bound to those sequence numbers has been delivered and acknowledged by the receiver and all duplicate copies of the segments have "drained" from the internet. Without such an assumption, two distinct TCP segments could conceivably be assigned the same or overlapping sequence numbers, causing confusion at the receiver as to which data is new and which is old. Remember that each segment is bound to as many consecutive sequence numbers as there are octets of data and SYN or FIN flags in the segment.

Under normal conditions, TCPs keep track of the next sequence number to emit and the oldest awaiting acknowledgment so as to avoid mistakenly using a sequence number over before its first use has been acknowledged. This alone does not guarantee that old duplicate data is drained from the net, so the sequence space has been made very large to reduce the probability that a wandering duplicate will cause trouble upon arrival. At 2 megabits/sec. it takes 4.5 hours to use up 2^{32} octets of sequence space. Since the maximum segment lifetime in the net is not likely to exceed a few tens of seconds, this is deemed ample protection for foreseeable nets, even if data rates escalate to 10's of megabits/sec. At 100 megabits/sec, the cycle time is 5.4 minutes which may be a little short, but still within reason.

The basic duplicate detection and sequencing algorithm in TCP can be defeated, however, if a source TCP does not have any memory of the sequence numbers it last used on a given connection. For example, if the TCP were to start all connections with sequence number 0, then upon crashing and restarting, a TCP might re-form an earlier connection (possibly after half-open connection resolution) and emit packets with sequence numbers identical to or overlapping with

packets still in the network which were emitted on an earlier incarnation of the same connection. In the absence of knowledge about the sequence numbers used on a particular connection, the TCP specification recommends that the source delay for MSL seconds before emitting segments on the connection, to allow time for segments from the earlier connection incarnation to drain from the system.

Even hosts which can remember the time of day and used it to select initial sequence number values are not immune from this problem (i.e., even if time of day is used to select an initial sequence number for each new connection incarnation).

Suppose, for example, that a connection is opened starting with sequence number S . Suppose that this connection is not used much and that eventually the initial sequence number function ($ISN(t)$) takes on a value equal to the sequence number, say S_1 , of the last segment sent by this TCP on a particular connection. Now suppose, at this instant, the host crashes, recovers, and establishes a new incarnation of the connection. The initial sequence number chosen is $S_1 = ISN(t)$ -- last used sequence number on old incarnation of connection! If the recovery occurs quickly enough, any old duplicates in the net bearing sequence numbers in the neighborhood of S_1 may arrive and be treated as new packets by the receiver of the new incarnation of the connection.

The problem is that the recovering host may not know for how long it crashed nor does it know whether there are still old duplicates in the system from earlier connection incarnations.

One way to deal with this problem is to deliberately delay emitting segments for one MSL after recovery from a crash- this is the "quiet time" specification. Hosts which prefer to avoid waiting are willing to risk possible confusion of old and new packets at a given destination may choose not to wait for the "quite time". Implementors may provide TCP users with the ability to select on a connection by connection basis whether to wait after a crash, or may informally implement the "quite time" for all connections. Obviously, even where a user selects to "wait," this is not necessary after the host has been "up" for at least MSL seconds.

To summarize: every segment emitted occupies one or more sequence numbers in the sequence space, the numbers occupied by a segment are "busy" or "in use" until MSL seconds have passed, upon crashing a block of space-time is occupied by the octets and SYN or FIN flags of the last emitted segment, if a new connection is started too soon and uses any of the sequence numbers in the space-time footprint of the last segment of the previous connection incarnation, there is a

potential sequence number overlap area which could cause confusion at the receiver.

3.4. Establishing a connection

The "three-way handshake" is the procedure used to establish a connection. This procedure normally is initiated by one TCP and responded to by another TCP. The procedure also works if two TCP simultaneously initiate the procedure. When simultaneous attempt occurs, each TCP receives a "SYN" segment which carries no acknowledgment after it has sent a "SYN". Of course, the arrival of an old duplicate "SYN" segment can potentially make it appear, to the recipient, that a simultaneous connection initiation is in progress. Proper use of "reset" segments can disambiguate these cases.

Several examples of connection initiation follow. Although these examples do not show connection synchronization using data-carrying segments, this is perfectly legitimate, so long as the receiving TCP doesn't deliver the data to the user until it is clear the data is valid (i.e., the data must be buffered at the receiver until the connection reaches the ESTABLISHED state). The three-way handshake reduces the possibility of false connections. It is the implementation of a trade-off between memory and messages to provide information for this checking.

The simplest three-way handshake is shown in Figure 5 below. The figures should be interpreted in the following way. Each line is numbered for reference purposes. Right arrows (-->) indicate departure of a TCP segment from TCP A to TCP B, or arrival of a segment at B from A. Left arrows (<--), indicate the reverse. Ellipsis (...) indicates a segment which is still in the network (delayed). An "XXX" indicates a segment which is lost or rejected. Comments appear in parentheses. TCP states represent the state AFTER the departure or arrival of the segment (whose contents are shown in the center of each line). Segment contents are shown in abbreviated form, with sequence number, control flags, and ACK field. Other fields such as window, addresses, lengths, and text have been left out in the interest of clarity.

TCP A	TCP B
1. CLOSED	LISTEN
2. SYN-SENT --> <SEQ=100><CTL=SYN>	--> SYN-RECEIVED
3. ESTABLISHED <-- <SEQ=300><ACK=101><CTL=SYN,ACK>	<-- SYN-RECEIVED
4. ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK>	--> ESTABLISHED
5. ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK><DATA>	--> ESTABLISHED

Basic 3-Way Handshake for Connection Synchronization

Figure 5

In line 2 of Figure 5, TCP A begins by sending a SYN segment indicating that it will use sequence numbers starting with sequence number 100. In line 3, TCP B sends a SYN and acknowledges the SYN it received from TCP A. Note that the acknowledgment field indicates TCP B is now expecting to hear sequence 101, acknowledging the SYN which occupied sequence 100.

At line 4, TCP A responds with an empty segment containing an ACK for TCP B's SYN; and in line 5, TCP A sends some data. Note that the sequence number of the segment in line 5 is the same as in line 4 because the ACK does not occupy sequence number space (if it did, we would wind up ACKing ACK's!).

Simultaneous initiation is only slightly more complex, as is shown in Figure 6. Each TCP cycles from CLOSED to SYN-SENT to SYN-RECEIVED to ESTABLISHED.

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. SYN-RECEIVED	--> <SEQ=100><ACK=301><CTL=SYN,ACK>	...
6. ESTABLISHED	<-- <SEQ=300><ACK=101><CTL=SYN,ACK>	<-- SYN-RECEIVED
7.	... <SEQ=100><ACK=301><CTL=SYN,ACK>	--> ESTABLISHED

Simultaneous Connection Synchronization

Figure 6

The principle reason for the three-way handshake is to prevent old duplicate connection initiations from causing confusion. To deal with this, a special control message, reset, has been devised. If the receiving TCP is in a non-synchronized state (i.e., SYN-SENT, SYN-RECEIVED), it returns to LISTEN on receiving an acceptable reset. If the TCP is in one of the synchronized states (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), it aborts the connection and informs its user. We discuss this latter case under "half-open" connections below.

TCP A		TCP B
1. CLOSED		LISTEN
2. SYN-SENT	--> <SEQ=100><CTL=SYN>	...
3. (duplicate)	... <SEQ=90><CTL=SYN>	--> SYN-RECEIVED
4. SYN-SENT	<-- <SEQ=300><ACK=91><CTL=SYN,ACK>	<-- SYN-RECEIVED
5. SYN-SENT	--> <SEQ=91><CTL=RST>	--> LISTEN
6.	... <SEQ=100><CTL=SYN>	--> SYN-RECEIVED
7. SYN-SENT	<-- <SEQ=400><ACK=101><CTL=SYN,ACK>	<-- SYN-RECEIVED
8. ESTABLISHED	--> <SEQ=101><ACK=401><CTL=ACK>	--> ESTABLISHED

Recovery from Old Duplicate SYN

Figure 7

As a simple example of recovery from old duplicates, consider Figure 7. At line 3, an old duplicate SYN arrives at TCP B. TCP B cannot tell that this is an old duplicate, so it responds normally (line 4). TCP A detects that the ACK field is incorrect and returns a RST (reset) with its SEQ field selected to make the segment believable. TCP B, on receiving the RST, returns to the LISTEN state. When the original SYN (pun intended) finally arrives at line 6, the synchronization proceeds normally. If the SYN at line 6 had arrived before the RST, a more complex exchange might have occurred with RST's sent in both directions.

Half-Open Connections and Other Anomalies

An established connection is said to be "half-open" if one of the TCPs has closed or aborted the connection at its end without the knowledge of the other, or if the two ends of the connection have become desynchronized owing to a crash that resulted in loss of memory. Such connections will automatically become reset if an attempt is made to send data in either direction. However, half-open connections are expected to be unusual, and the recovery procedure is mildly involved.

If at site A the connection no longer exists, then an attempt by the user at site B to send any data on it will result in the site B TCP receiving a reset control message. Such a message indicates to the

site B TCP that something is wrong, and it is expected to abort the connection.

Assume that two user processes A and B are communicating with one another when a crash occurs causing loss of memory to A's TCP. Depending on the operating system supporting A's TCP, it is likely that some error recovery mechanism exists. When the TCP is up again, A is likely to start again from the beginning or from a recovery point. As a result, A will probably try to OPEN the connection again or try to SEND on the connection it believes open. In the latter case, it receives the error message "connection not open" from the local (A's) TCP. In an attempt to establish the connection, A's TCP will send a segment containing SYN. This scenario leads to the example shown in Figure 8. After TCP A crashes, the user attempts to re-open the connection. TCP B, in the meantime, thinks the connection is open.

TCP A	TCP B
1. (CRASH)	(send 300, receive 100)
2. CLOSED	ESTABLISHED
3. SYN-SENT --> <SEQ=400><CTL=SYN>	--> (??)
4. (!!)	<-- <SEQ=300><ACK=100><CTL=ACK> <-- ESTABLISHED
5. SYN-SENT --> <SEQ=100><CTL=RST>	--> (Abort!!)
6. SYN-SENT	CLOSED
7. SYN-SENT --> <SEQ=400><CTL=SYN>	-->

Half-Open Connection Discovery

Figure 8

When the SYN arrives at line 3, TCP B, being in a synchronized state, and the incoming segment outside the window, responds with an acknowledgment indicating what sequence it next expects to hear (ACK 100). TCP A sees that this segment does not acknowledge anything it sent and, being unsynchronized, sends a reset (RST) because it has detected a half-open connection. TCP B aborts at line 5. TCP A will continue to try to establish the connection; the problem is now reduced to the basic 3-way handshake of Figure 5.

An interesting alternative case occurs when TCP A crashes and TCP B tries to send data on what it thinks is a synchronized connection.

This is illustrated in Figure 9. In this case, the data arriving at TCP A from TCP B (line 2) is unacceptable because no such connection exists, so TCP A sends a RST. The RST is acceptable so TCP B processes it and aborts the connection.

TCP A	TCP B
1. (CRASH)	(send 300, receive 100)
2. (??) <-- <SEQ=300><ACK=100><DATA=10><CTL=ACK>	<-- ESTABLISHED
3. --> <SEQ=100><CTL=RST>	--> (ABORT!!)

Active Side Causes Half-Open Connection Discovery

Figure 9

In Figure 10, we find the two TCPs A and B with passive connections waiting for SYN. An old duplicate arriving at TCP B (line 2) stirs B into action. A SYN-ACK is returned (line 3) and causes TCP A to generate a RST (the ACK in line 3 is not acceptable). TCP B accepts the reset and returns to its passive LISTEN state.

TCP A	TCP B
1. LISTEN	LISTEN
2. ... <SEQ=Z><CTL=SYN>	--> SYN-RECEIVED
3. (??) <-- <SEQ=X><ACK=Z+1><CTL=SYN,ACK>	<-- SYN-RECEIVED
4. --> <SEQ=Z+1><CTL=RST>	--> (return to LISTEN!)
5. LISTEN	LISTEN

Old Duplicate SYN Initiates a Reset on two Passive Sockets

Figure 10

A variety of other cases are possible, all of which are accounted for by the following rules for RST generation and processing.

Reset Generation

As a general rule, reset (RST) must be sent whenever a segment arrives which apparently is not intended for the current connection. A reset must not be sent if it is not clear that this is the case.

There are three groups of states:

1. If the connection does not exist (CLOSED) then a reset is sent in response to any incoming segment except another reset. In particular, SYNs addressed to a non-existent connection are rejected by this means.

If the incoming segment has the ACK bit set, the reset takes its sequence number from the ACK field of the segment, otherwise the reset has sequence number zero and the ACK field is set to the sum of the sequence number and segment length of the incoming segment. The connection remains in the CLOSED state.

2. If the connection is in any non-synchronized state (LISTEN, SYN-SENT, SYN-RECEIVED), and the incoming segment acknowledges something not yet sent (the segment carries an unacceptable ACK), or if an incoming segment has a security level or compartment which does not exactly match the level and compartment requested for the connection, a reset is sent.

If our SYN has not been acknowledged and the precedence level of the incoming segment is higher than the precedence level requested then either raise the local precedence level (if allowed by the user and the system) or send a reset; or if the precedence level of the incoming segment is lower than the precedence level requested then continue as if the precedence matched exactly (if the remote TCP cannot raise the precedence level to match ours this will be detected in the next segment it sends, and the connection will be terminated then). If our SYN has been acknowledged (perhaps in this incoming segment) the precedence level of the incoming segment must match the local precedence level exactly, if it does not a reset must be sent.

If the incoming segment has an ACK field, the reset takes its sequence number from the ACK field of the segment, otherwise the reset has sequence number zero and the ACK field is set to the sum of the sequence number and segment length of the incoming segment. The connection remains in the same state.

3. If the connection is in a synchronized state (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), any unacceptable segment (out of window sequence number or unacceptable acknowledgment number) must elicit only an empty acknowledgment segment containing the current send-sequence number

and an acknowledgment indicating the next sequence number expected to be received, and the connection remains in the same state.

If an incoming segment has a security level, or compartment, or precedence which does not exactly match the level, and compartment, and precedence requested for the connection, a reset is sent and the connection goes to the CLOSED state. The reset takes its sequence number from the ACK field of the incoming segment.

Reset Processing

In all states except SYN-SENT, all reset (RST) segments are validated by checking their SEQ-fields. A reset is valid if its sequence number is in the window. In the SYN-SENT state (a RST received in response to an initial SYN), the RST is acceptable if the ACK field acknowledges the SYN.

The receiver of a RST first validates it, then changes state. If the receiver was in the LISTEN state, it ignores it. If the receiver was in SYN-RECEIVED state and had previously been in the LISTEN state, then the receiver returns to the LISTEN state, otherwise the receiver aborts the connection and goes to the CLOSED state. If the receiver was in any other state, it aborts the connection and advises the user and goes to the CLOSED state.

3.5. Closing a Connection

CLOSE is an operation meaning "I have no more data to send." The notion of closing a full-duplex connection is subject to ambiguous interpretation, of course, since it may not be obvious how to treat the receiving side of the connection. We have chosen to treat CLOSE in a simplex fashion. The user who CLOSEs may continue to RECEIVE until he is told that the other side has CLOSED also. Thus, a program could initiate several SENDs followed by a CLOSE, and then continue to RECEIVE until signaled that a RECEIVE failed because the other side has CLOSED. We assume that the TCP will signal a user, even if no RECEIVES are outstanding, that the other side has closed, so the user can terminate his side gracefully. A TCP will reliably deliver all buffers SENT before the connection was CLOSED so a user who expects no data in return need only wait to hear the connection was CLOSED successfully to know that all his data was received at the destination TCP. Users must keep reading connections they close for sending until the TCP says no more data.

There are essentially three cases:

- 1) The user initiates by telling the TCP to CLOSE the connection

2) The remote TCP initiates by sending a FIN control signal

3) Both users CLOSE simultaneously

Case 1: Local user initiates the close

In this case, a FIN segment can be constructed and placed on the outgoing segment queue. No further SENDs from the user will be accepted by the TCP, and it enters the FIN-WAIT-1 state. RECEIVES are allowed in this state. All segments preceding and including FIN will be retransmitted until acknowledged. When the other TCP has both acknowledged the FIN and sent a FIN of its own, the first TCP can ACK this FIN. Note that a TCP receiving a FIN will ACK but not send its own FIN until its user has CLOSED the connection also.

Case 2: TCP receives a FIN from the network

If an unsolicited FIN arrives from the network, the receiving TCP can ACK it and tell the user that the connection is closing. The user will respond with a CLOSE, upon which the TCP can send a FIN to the other TCP after sending any remaining data. The TCP then waits until its own FIN is acknowledged whereupon it deletes the connection. If an ACK is not forthcoming, after the user timeout the connection is aborted and the user is told.

Case 3: both users close simultaneously

A simultaneous CLOSE by users at both ends of a connection causes FIN segments to be exchanged. When all segments preceding the FINs have been processed and acknowledged, each TCP can ACK the FIN it has received. Both will, upon receiving these ACKs, delete the connection.

TCP A		TCP B
1. ESTABLISHED		ESTABLISHED
2. (Close) FIN-WAIT-1	--> <SEQ=100><ACK=300><CTL=FIN,ACK>	--> CLOSE-WAIT
3. FIN-WAIT-2	<-- <SEQ=300><ACK=101><CTL=ACK>	<-- CLOSE-WAIT
4. TIME-WAIT	<-- <SEQ=300><ACK=101><CTL=FIN,ACK>	(Close) <-- LAST-ACK
5. TIME-WAIT	--> <SEQ=101><ACK=301><CTL=ACK>	--> CLOSED
6. (2 MSL) CLOSED		

Normal Close Sequence

Figure 11

TCP A		TCP B
1. ESTABLISHED		ESTABLISHED
2. (Close) FIN-WAIT-1	--> <SEQ=100><ACK=300><CTL=FIN,ACK> <-- <SEQ=300><ACK=100><CTL=FIN,ACK> ... <SEQ=100><ACK=300><CTL=FIN,ACK>	(Close) ... FIN-WAIT-1 <-- -->
3. CLOSING	--> <SEQ=101><ACK=301><CTL=ACK> <-- <SEQ=301><ACK=101><CTL=ACK> ... <SEQ=101><ACK=301><CTL=ACK>	... CLOSING <-- -->
4. TIME-WAIT (2 MSL) CLOSED		TIME-WAIT (2 MSL) CLOSED

Simultaneous Close Sequence

Figure 12

3.6. Precedence and Security

The intent is that connection be allowed only between ports operating with exactly the same security and compartment values and at the higher of the precedence level requested by the two ports.

The precedence and security parameters used in TCP are exactly those defined in the Internet Protocol (IP) [2]. Throughout this TCP specification the term "security/compartment" is intended to indicate the security parameters used in IP including security, compartment, user group, and handling restriction.

A connection attempt with mismatched security/compartment values or a lower precedence value must be rejected by sending a reset. Rejecting a connection due to too low a precedence only occurs after an acknowledgment of the SYN has been received.

Note that TCP modules which operate only at the default value of precedence will still have to check the precedence of incoming segments and possibly raise the precedence level they use on the connection.

The security parameters may be used even in a non-secure environment (the values would indicate unclassified data), thus hosts in non-secure environments must be prepared to receive the security parameters, though they need not send them.

3.7. Segmentation

The term "segmentation" refers to the activity TCP performs when ingesting a stream of bytes from a sending application and packetizing that stream of bytes into TCP segments.

For efficiency and performance reasons, it is desirable to send large segments that contain as many bytes of payload data as possible. However, packets that are too long will either be fragmented or dropped within the network. Some firewalls or middleboxes may drop fragmented packets. In either case, when packets are dropped, the connection can fail; hence, it is best for a TCP implementation to avoid generating fragments.

To enable a TCP sender to maximize the size of segments that it sends, without generating fragments, TCP includes the Maximum Segment Size option to convey endpoint information, and TCP implementations also support Path MTU Discovery to discover the limits and capabilities of intermediate networks.

When TCP is used in a situation where either the IP or TCP headers are not minimum, the sender must reduce the amount of TCP data in any given packet by the number of octets used by the IP and TCP options. The rationale for this is explained in RFC 6691.

3.7.1. Maximum Segment Size Option

TCP MUST implement both sending and receiving the Maximum Segment Size option.

TCP SHOULD send an MSS (Maximum Segment Size) option in every SYN segment when its receive MSS differs from the default 536, and MAY send it always.

If an MSS option is not received at connection setup, TCP MUST assume a default send MSS of 536 (576-40).

The maximum size of a segment that TCP really sends, the "effective send MSS," MUST be the smaller of the send MSS (which reflects the available reassembly buffer size at the remote host) and the largest size permitted by the IP layer:

$$\text{Eff.snd.MSS} =$$
$$\min(\text{SendMSS}+20, \text{MMS_S}) - \text{TCP}h\text{drsize} - \text{IPOptionsize}$$

where:

- o SendMSS is the MSS value received from the remote host, or the default 536 if no MSS option is received.
- o MMS_S is the maximum size for a transport-layer message that TCP may send.
- o TCPHdrsize is the size of the fixed TCP header; this is normally 20, but may be larger if TCP options are to be sent.
- o IPOptionsize is the size of any IP options that TCP will pass to the IP layer with the current message.

The MSS value to be sent in an MSS option should be equal to the effective MTU minus the fixed IP and TCP headers. By ignoring both IP and TCP options when calculating the value for the MSS option, if there are any IP or TCP options to be sent in a packet, then the sender must decrease the size of the TCP data accordingly. RFC 6691 discusses this in greater detail.

The MSS value to be sent in an MSS option must be less than or equal to:

$$\text{MMS_R} - 20$$

where MMS_R is the maximum size for a transport-layer message that can be received (and reassembled). TCP obtains MMS_R and MMS_S from the IP layer; see the generic call GET_MAXSIZES in Section 3.4 of RFC 1122.

3.7.2. Path MTU Discovery

The TCP MSS option specifies an upper bound for the size of packets that can be received. Hence, setting the value in the MSS option too small can impact the ability for Path MTU Discovery to find a larger path MTU. For more information on Path MTU Discovery, see:

- o "Path MTU Discovery" [RFC1191]
- o "TCP Problems with Path MTU Discovery" [RFC2923]
- o "Packetization Layer Path MTU Discovery" [RFC4821]

3.7.3. Interfaces with Variable MSS Values

The effective MTU can sometimes vary, as when used with variable compression, e.g., RObust Header Compression (ROHC) [RFC5795]. It is tempting for TCP to want to advertise the largest possible MSS, to support the most efficient use of compressed payloads. Unfortunately, some compression schemes occasionally need to transmit full headers (and thus smaller payloads) to resynchronize state at their endpoint compressors/decompressors. If the largest MTU is used to calculate the value to advertise in the MSS option, TCP retransmission may interfere with compressor resynchronization.

As a result, when the effective MTU of an interface varies, TCP SHOULD use the smallest effective MTU of the interface to calculate the value to advertise in the MSS option.

3.7.4. IPv6 Jumbograms

In order to support TCP over IPv6 jumbograms, implementations need to be able to send TCP segments larger than 64K. RFC 2675 [RFC2675] defines that a value of 65,535 is to be treated as infinity, and Path MTU Discovery [RFC1981] is used to determine the actual MSS.

3.8. Data Communication

Once the connection is established data is communicated by the exchange of segments. Because segments may be lost due to errors (checksum test failure), or network congestion, TCP uses retransmission (after a timeout) to ensure delivery of every segment. Duplicate segments may arrive due to network or TCP retransmission.

As discussed in the section on sequence numbers the TCP performs certain tests on the sequence and acknowledgment numbers in the segments to verify their acceptability.

The sender of data keeps track of the next sequence number to use in the variable SND.NXT. The receiver of data keeps track of the next sequence number to expect in the variable RCV.NXT. The sender of data keeps track of the oldest unacknowledged sequence number in the variable SND.UNA. If the data flow is momentarily idle and all data sent has been acknowledged then the three variables will be equal.

When the sender creates a segment and transmits it the sender advances SND.NXT. When the receiver accepts a segment it advances RCV.NXT and sends an acknowledgment. When the data sender receives an acknowledgment it advances SND.UNA. The extent to which the values of these variables differ is a measure of the delay in the communication. The amount by which the variables are advanced is the length of the data and SYN or FIN flags in the segment. Note that once in the ESTABLISHED state all segments must carry current acknowledgment information.

The CLOSE user call implies a push function, as does the FIN control flag in an incoming segment.

Retransmission Timeout

NOTE: TODO this needs to be updated in light of 1122 4.2.2.15 and errata 573; this will be done as part of RFC 1122 incorporation into this document.

Because of the variability of the networks that compose an internetwork system and the wide range of uses of TCP connections the retransmission timeout must be dynamically determined. One procedure for determining a retransmission timeout is given here as an illustration.

An Example Retransmission Timeout Procedure

Measure the elapsed time between sending a data octet with a particular sequence number and receiving an acknowledgment that covers that sequence number (segments sent do not have to match segments received). This measured elapsed time is the Round Trip Time (RTT). Next compute a Smoothed Round Trip Time (SRTT) as:

$$\text{SRTT} = (\text{ALPHA} * \text{SRTT}) + ((1-\text{ALPHA}) * \text{RTT})$$

and based on this, compute the retransmission timeout (RTO) as:

$$\text{RTO} = \min[\text{UBOUND}, \max[\text{LBOUND}, (\text{BETA} * \text{SRTT})]]$$

where UBOUND is an upper bound on the timeout (e.g., 1 minute), LBOUND is a lower bound on the timeout (e.g., 1 second), ALPHA is a smoothing factor (e.g., .8 to .9), and BETA is a delay variance factor (e.g., 1.3 to 2.0).

The Communication of Urgent Information

As a result of implementation differences and middlebox interactions, new applications SHOULD NOT employ the TCP urgent mechanism. However, TCP implementations MUST still include support for the urgent mechanism. Details can be found in RFC 6093 [7].

The objective of the TCP urgent mechanism is to allow the sending user to stimulate the receiving user to accept some urgent data and to permit the receiving TCP to indicate to the receiving user when all the currently known urgent data has been received by the user.

This mechanism permits a point in the data stream to be designated as the end of urgent information. Whenever this point is in advance of the receive sequence number (RCV.NXT) at the receiving TCP, that TCP must tell the user to go into "urgent mode"; when the receive sequence number catches up to the urgent pointer, the TCP must tell user to go into "normal mode". If the urgent pointer is updated while the user is in "urgent mode", the update will be invisible to the user.

The method employs a urgent field which is carried in all segments transmitted. The URG control flag indicates that the urgent field is meaningful and must be added to the segment sequence number to yield the urgent pointer. The absence of this flag indicates that there is no urgent data outstanding.

To send an urgent indication the user must also send at least one data octet. If the sending user also indicates a push, timely delivery of the urgent information to the destination process is enhanced.

A TCP MUST support a sequence of urgent data of any length. [3]

A TCP MUST inform the application layer asynchronously whenever it receives an Urgent pointer and there was previously no pending urgent data, or whenever the Urgent pointer advances in the data stream. There MUST be a way for the application to learn how much urgent data remains to be read from the connection, or at least to determine whether or not more urgent data remains to be read. [3]

Managing the Window

The window sent in each segment indicates the range of sequence numbers the sender of the window (the data receiver) is currently prepared to accept. There is an assumption that this is related to the currently available data buffer space available for this connection.

Indicating a large window encourages transmissions. If more data arrives than can be accepted, it will be discarded. This will result in excessive retransmissions, adding unnecessarily to the load on the network and the TCPs. Indicating a small window may restrict the transmission of data to the point of introducing a round trip delay between each new segment transmitted.

The mechanisms provided allow a TCP to advertise a large window and to subsequently advertise a much smaller window without having accepted that much data. This, so called "shrinking the window," is strongly discouraged. The robustness principle dictates that TCPs will not shrink the window themselves, but will be prepared for such behavior on the part of other TCPs.

The sending TCP must be prepared to accept from the user and send at least one octet of new data even if the send window is zero. The sending TCP must regularly retransmit to the receiving TCP even when the window is zero. Two minutes is recommended for the retransmission interval when the window is zero. This retransmission is essential to guarantee that when either TCP has a zero window the re-opening of the window will be reliably reported to the other.

When the receiving TCP has a zero window and a segment arrives it must still send an acknowledgment showing its next expected sequence number and current window (zero).

The sending TCP packages the data to be transmitted into segments which fit the current window, and may repackage segments on the retransmission queue. Such repackaging is not required, but may be helpful.

In a connection with a one-way data flow, the window information will be carried in acknowledgment segments that all have the same sequence number so there will be no way to reorder them if they arrive out of order. This is not a serious problem, but it will allow the window information to be on occasion temporarily based on old reports from the data receiver. A refinement to avoid this problem is to act on the window information from segments that carry the highest acknowledgment number (that is segments with acknowledgment number equal or greater than the highest previously received).

The window management procedure has significant influence on the communication performance. The following comments are suggestions to implementers.

Window Management Suggestions

Allocating a very small window causes data to be transmitted in many small segments when better performance is achieved using fewer large segments.

One suggestion for avoiding small windows is for the receiver to defer updating a window until the additional allocation is at least X percent of the maximum allocation possible for the connection (where X might be 20 to 40).

Another suggestion is for the sender to avoid sending small segments by waiting until the window is large enough before sending data. If the user signals a push function then the data must be sent even if it is a small segment.

Note that the acknowledgments should not be delayed or unnecessary retransmissions will result. One strategy would be to send an acknowledgment when a small segment arrives (with out updating the window information), and then to send another acknowledgment with new window information when the window is larger.

The segment sent to probe a zero window may also begin a break up of transmitted data into smaller and smaller segments. If a segment containing a single data octet sent to probe a zero window is accepted, it consumes one octet of the window now available. If the sending TCP simply sends as much as it can whenever the window is non zero, the transmitted data will be broken into alternating big and small segments. As time goes on, occasional pauses in the receiver making window allocation available will result in breaking the big segments into a small and not quite so big pair. And after a while the data transmission will be in mostly small segments.

The suggestion here is that the TCP implementations need to actively attempt to combine small window allocations into larger windows, since the mechanisms for managing the window tend to lead to many small windows in the simplest minded implementations.

3.9. Interfaces

There are of course two interfaces of concern: the user/TCP interface and the TCP/lower-level interface. We have a fairly elaborate model of the user/TCP interface, but the interface to the lower level

protocol module is left unspecified here, since it will be specified in detail by the specification of the lower level protocol. For the case that the lower level is IP we note some of the parameter values that TCPs might use.

3.9.1. User/TCP Interface

The following functional description of user commands to the TCP is, at best, fictional, since every operating system will have different facilities. Consequently, we must warn readers that different TCP implementations may have different user interfaces. However, all TCPs must provide a certain minimum set of services to guarantee that all TCP implementations can support the same protocol hierarchy. This section specifies the functional interfaces required of all TCP implementations.

TCP User Commands

The following sections functionally characterize a USER/TCP interface. The notation used is similar to most procedure or function calls in high level languages, but this usage is not meant to rule out trap type service calls (e.g., SVCs, UUOs, EMTs).

The user commands described below specify the basic functions the TCP must perform to support interprocess communication. Individual implementations must define their own exact format, and may provide combinations or subsets of the basic functions in single calls. In particular, some implementations may wish to automatically OPEN a connection on the first SEND or RECEIVE issued by the user for a given connection.

In providing interprocess communication facilities, the TCP must not only accept commands, but must also return information to the processes it serves. The latter consists of:

- (a) general information about a connection (e.g., interrupts, remote close, binding of unspecified foreign socket).
- (b) replies to specific user commands indicating success or various types of failure.

Open

Format: OPEN (local port, foreign socket, active/passive [, timeout] [, precedence] [, security/compartments] [, options])
-> local connection name

We assume that the local TCP is aware of the identity of the processes it serves and will check the authority of the process to use the connection specified. Depending upon the implementation of the TCP, the local network and TCP identifiers for the source address will either be supplied by the TCP or the lower level protocol (e.g., IP). These considerations are the result of concern about security, to the extent that no TCP be able to masquerade as another one, and so on. Similarly, no process can masquerade as another without the collusion of the TCP.

If the active/passive flag is set to passive, then this is a call to LISTEN for an incoming connection. A passive open may have either a fully specified foreign socket to wait for a particular connection or an unspecified foreign socket to wait for any call. A fully specified passive call can be made active by the subsequent execution of a SEND.

A transmission control block (TCB) is created and partially filled in with data from the OPEN command parameters.

On an active OPEN command, the TCP will begin the procedure to synchronize (i.e., establish) the connection at once.

The timeout, if present, permits the caller to set up a timeout for all data submitted to TCP. If data is not successfully delivered to the destination within the timeout period, the TCP will abort the connection. The present global default is five minutes.

The TCP or some component of the operating system will verify the users authority to open a connection with the specified precedence or security/compartiment. The absence of precedence or security/compartiment specification in the OPEN call indicates the default values must be used.

TCP will accept incoming requests as matching only if the security/compartiment information is exactly the same and only if the precedence is equal to or higher than the precedence requested in the OPEN call.

The precedence for the connection is the higher of the values requested in the OPEN call and received from the incoming request, and fixed at that value for the life of the connection. Implementers may want to give the user control of this precedence negotiation. For example, the user might be allowed to specify that the precedence must be exactly matched,

or that any attempt to raise the precedence be confirmed by the user.

A local connection name will be returned to the user by the TCP. The local connection name can then be used as a short hand term for the connection defined by the <local socket, foreign socket> pair.

Send

Format: SEND (local connection name, buffer address, byte count, PUSH flag, URGENT flag [,timeout])

This call causes the data contained in the indicated user buffer to be sent on the indicated connection. If the connection has not been opened, the SEND is considered an error. Some implementations may allow users to SEND first; in which case, an automatic OPEN would be done. If the calling process is not authorized to use this connection, an error is returned.

If the PUSH flag is set, the data must be transmitted promptly to the receiver, and the PUSH bit will be set in the last TCP segment created from the buffer. If the PUSH flag is not set, the data may be combined with data from subsequent SENDs for transmission efficiency.

New applications SHOULD NOT set the URGENT flag [7] due to implementation differences and middlebox issues.

If the URGENT flag is set, segments sent to the destination TCP will have the urgent pointer set. The receiving TCP will signal the urgent condition to the receiving process if the urgent pointer indicates that data preceding the urgent pointer has not been consumed by the receiving process. The purpose of urgent is to stimulate the receiver to process the urgent data and to indicate to the receiver when all the currently known urgent data has been received. The number of times the sending user's TCP signals urgent will not necessarily be equal to the number of times the receiving user will be notified of the presence of urgent data.

If no foreign socket was specified in the OPEN, but the connection is established (e.g., because a LISTENing connection has become specific due to a foreign segment arriving for the local socket), then the designated buffer is sent to the implied foreign socket. Users who make use of OPEN with an

unspecified foreign socket can make use of SEND without ever explicitly knowing the foreign socket address.

However, if a SEND is attempted before the foreign socket becomes specified, an error will be returned. Users can use the STATUS call to determine the status of the connection. In some implementations the TCP may notify the user when an unspecified socket is bound.

If a timeout is specified, the current user timeout for this connection is changed to the new one.

In the simplest implementation, SEND would not return control to the sending process until either the transmission was complete or the timeout had been exceeded. However, this simple method is both subject to deadlocks (for example, both sides of the connection might try to do SENDs before doing any RECEIVES) and offers poor performance, so it is not recommended. A more sophisticated implementation would return immediately to allow the process to run concurrently with network I/O, and, furthermore, to allow multiple SENDs to be in progress. Multiple SENDs are served in first come, first served order, so the TCP will queue those it cannot service immediately.

We have implicitly assumed an asynchronous user interface in which a SEND later elicits some kind of SIGNAL or pseudo-interrupt from the serving TCP. An alternative is to return a response immediately. For instance, SENDs might return immediate local acknowledgment, even if the segment sent had not been acknowledged by the distant TCP. We could optimistically assume eventual success. If we are wrong, the connection will close anyway due to the timeout. In implementations of this kind (synchronous), there will still be some asynchronous signals, but these will deal with the connection itself, and not with specific segments or buffers.

In order for the process to distinguish among error or success indications for different SENDs, it might be appropriate for the buffer address to be returned along with the coded response to the SEND request. TCP-to-user signals are discussed below, indicating the information which should be returned to the calling process.

Receive

Format: RECEIVE (local connection name, buffer address, byte count) -> byte count, urgent flag, push flag

This command allocates a receiving buffer associated with the specified connection. If no OPEN precedes this command or the calling process is not authorized to use this connection, an error is returned.

In the simplest implementation, control would not return to the calling program until either the buffer was filled, or some error occurred, but this scheme is highly subject to deadlocks. A more sophisticated implementation would permit several RECEIVES to be outstanding at once. These would be filled as segments arrive. This strategy permits increased throughput at the cost of a more elaborate scheme (possibly asynchronous) to notify the calling program that a PUSH has been seen or a buffer filled.

If enough data arrive to fill the buffer before a PUSH is seen, the PUSH flag will not be set in the response to the RECEIVE. The buffer will be filled with as much data as it can hold. If a PUSH is seen before the buffer is filled the buffer will be returned partially filled and PUSH indicated.

If there is urgent data the user will have been informed as soon as it arrived via a TCP-to-user signal. The receiving user should thus be in "urgent mode". If the URGENT flag is on, additional urgent data remains. If the URGENT flag is off, this call to RECEIVE has returned all the urgent data, and the user may now leave "urgent mode". Note that data following the urgent pointer (non-urgent data) cannot be delivered to the user in the same buffer with preceding urgent data unless the boundary is clearly marked for the user.

To distinguish among several outstanding RECEIVES and to take care of the case that a buffer is not completely filled, the return code is accompanied by both a buffer pointer and a byte count indicating the actual length of the data received.

Alternative implementations of RECEIVE might have the TCP allocate buffer storage, or the TCP might share a ring buffer with the user.

Close

Format: CLOSE (local connection name)

This command causes the connection specified to be closed. If the connection is not open or the calling process is not authorized to use this connection, an error is returned. Closing connections is intended to be a graceful operation in

the sense that outstanding SENDs will be transmitted (and retransmitted), as flow control permits, until all have been serviced. Thus, it should be acceptable to make several SEND calls, followed by a CLOSE, and expect all the data to be sent to the destination. It should also be clear that users should continue to RECEIVE on CLOSING connections, since the other side may be trying to transmit the last of its data. Thus, CLOSE means "I have no more to send" but does not mean "I will not receive any more." It may happen (if the user level protocol is not well thought out) that the closing side is unable to get rid of all its data before timing out. In this event, CLOSE turns into ABORT, and the closing TCP gives up.

The user may CLOSE the connection at any time on his own initiative, or in response to various prompts from the TCP (e.g., remote close executed, transmission timeout exceeded, destination inaccessible).

Because closing a connection requires communication with the foreign TCP, connections may remain in the closing state for a short time. Attempts to reopen the connection before the TCP replies to the CLOSE command will result in error responses.

Close also implies push function.

Status

Format: STATUS (local connection name) -> status data

This is an implementation dependent user command and could be excluded without adverse effect. Information returned would typically come from the TCB associated with the connection.

This command returns a data block containing the following information:

- local socket,
- foreign socket,
- local connection name,
- receive window,
- send window,
- connection state,
- number of buffers awaiting acknowledgment,
- number of buffers pending receipt,
- urgent state,
- precedence,
- security/compartments,
- and transmission timeout.

Depending on the state of the connection, or on the implementation itself, some of this information may not be available or meaningful. If the calling process is not authorized to use this connection, an error is returned. This prevents unauthorized processes from gaining information about a connection.

Abort

Format: ABORT (local connection name)

This command causes all pending SENDs and RECEIVES to be aborted, the TCB to be removed, and a special RESET message to be sent to the TCP on the other side of the connection. Depending on the implementation, users may receive abort indications for each outstanding SEND or RECEIVE, or may simply receive an ABORT-acknowledgment.

TCP-to-User Messages

It is assumed that the operating system environment provides a means for the TCP to asynchronously signal the user program. When the TCP does signal a user program, certain information is passed to the user. Often in the specification the information will be an error message. In other cases there will be information relating to the completion of processing a SEND or RECEIVE or other user call.

The following information is provided:

Local Connection Name	Always
Response String	Always
Buffer Address	Send & Receive
Byte count (counts bytes received)	Receive
Push flag	Receive
Urgent flag	Receive

3.9.2. TCP/Lower-Level Interface

The TCP calls on a lower level protocol module to actually send and receive information over a network. One case is that of the ARPA internetwork system where the lower level module is the Internet Protocol (IP) [2].

If the lower level protocol is IP it provides arguments for a type of service and for a time to live. TCP uses the following settings for these parameters:

Type of Service = Precedence: given by user, Delay: normal, Throughput: normal, Reliability: normal; or binary XXX00000, where XXX are the three bits determining precedence, e.g. 000 means routine precedence.

Time to Live = one minute, or 00111100.

Note that the assumed maximum segment lifetime is two minutes. Here we explicitly ask that a segment be destroyed if it cannot be delivered by the internet system within one minute.

If the lower level is IP (or other protocol that provides this feature) and source routing is used, the interface must allow the route information to be communicated. This is especially important so that the source and destination addresses used in the TCP checksum be the originating source and ultimate destination. It is also important to preserve the return route to answer connection requests.

Any lower level protocol will have to provide the source address, destination address, and protocol fields, and some way to determine the "TCP length", both to provide the functional equivalent service of IP and to be used in the TCP checksum.

3.10. Event Processing

The processing depicted in this section is an example of one possible implementation. Other implementations may have slightly different processing sequences, but they should differ from those in this section only in detail, not in substance.

The activity of the TCP can be characterized as responding to events. The events that occur can be cast into three categories: user calls, arriving segments, and timeouts. This section describes the processing the TCP does in response to each of the events. In many cases the processing required depends on the state of the connection.

Events that occur:

User Calls

- OPEN
- SEND
- RECEIVE
- CLOSE
- ABORT
- STATUS

Arriving Segments

SEGMENT ARRIVES

Timeouts

USER TIMEOUT
RETRANSMISSION TIMEOUT
TIME-WAIT TIMEOUT

The model of the TCP/user interface is that user commands receive an immediate return and possibly a delayed response via an event or pseudo interrupt. In the following descriptions, the term "signal" means cause a delayed response.

Error responses are given as character strings. For example, user commands referencing connections that do not exist receive "error: connection not open".

Please note in the following that all arithmetic on sequence numbers, acknowledgment numbers, windows, et cetera, is modulo 2^{32} the size of the sequence number space. Also note that " \leq " means less than or equal to (modulo 2^{32}).

A natural way to think about processing incoming segments is to imagine that they are first tested for proper sequence number (i.e., that their contents lie in the range of the expected "receive window" in the sequence number space) and then that they are generally queued and processed in sequence number order.

When a segment overlaps other already received segments we reconstruct the segment to contain just the new data, and adjust the header fields to be consistent.

Note that if no state change is mentioned the TCP stays in the same state.

OPEN Call

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

Create a new transmission control block (TCB) to hold connection state information. Fill in local socket identifier, foreign socket, precedence, security/compartments, and user timeout information. Note that some parts of the foreign socket may be unspecified in a passive OPEN and are to be filled in by the parameters of the incoming SYN segment. Verify the security and precedence requested are allowed for this user, if not return "error: precedence not allowed" or "error: security/compartments not allowed." If passive enter the LISTEN state and return. If active and the foreign socket is unspecified, return "error: foreign socket unspecified"; if active and the foreign socket is specified, issue a SYN segment. An initial send sequence number (ISS) is selected. A SYN segment of the form <SEQ=ISS><CTL=SYN> is sent. Set SND.UNA to ISS, SND.NXT to ISS+1, enter SYN-SENT state, and return.

If the caller does not have access to the local socket specified, return "error: connection illegal for this process". If there is no room to create a new connection, return "error: insufficient resources".

LISTEN STATE

If active and the foreign socket is specified, then change the connection from passive to active, select an ISS. Send a SYN segment, set SND.UNA to ISS, SND.NXT to ISS+1. Enter SYN-SENT state. Data associated with SEND may be sent with SYN segment or queued for transmission after entering ESTABLISHED state. The urgent bit if requested in the command must be sent with the data segments sent as a result of this command. If there is no room to queue the request, respond with "error: insufficient resources". If Foreign socket was not specified, then return "error: foreign socket unspecified".

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

Return "error: connection already exists".

SEND Call

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

If the user does not have access to such a connection, then return "error: connection illegal for this process".

Otherwise, return "error: connection 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, set SND.UNA to ISS, SND.NXT to ISS+1. Enter SYN-SENT state. Data associated with SEND may be sent with SYN segment or queued for transmission after entering ESTABLISHED state. The urgent bit if requested in the command must be sent with the data segments sent as a result of this command. If there is no room to queue the request, respond with "error: insufficient resources". If Foreign socket was not specified, then return "error: foreign socket unspecified".

SYN-SENT STATE

SYN-RECEIVED STATE

Queue the data for transmission after entering ESTABLISHED state. If no space to queue, respond with "error: insufficient resources".

ESTABLISHED STATE

CLOSE-WAIT STATE

Segmentize the buffer and send it with a piggybacked acknowledgment (acknowledgment value = RCV.NXT). If there is insufficient space to remember this buffer, simply return "error: insufficient resources".

If the urgent flag is set, then SND.UP <- SND.NXT and set the urgent pointer in the outgoing segments.

FIN-WAIT-1 STATE

FIN-WAIT-2 STATE

CLOSING STATE

LAST-ACK STATE

TIME-WAIT STATE

Return "error: connection closing" and do not service request.

RECEIVE Call

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

If the user does not have access to such a connection, return "error: connection illegal for this process".

Otherwise return "error: connection does not exist".

LISTEN STATE

SYN-SENT STATE

SYN-RECEIVED STATE

Queue for processing after entering ESTABLISHED state. If there is no room to queue this request, respond with "error: insufficient resources".

ESTABLISHED STATE

FIN-WAIT-1 STATE

FIN-WAIT-2 STATE

If insufficient incoming segments are queued to satisfy the request, queue the request. If there is no queue space to remember the RECEIVE, respond with "error: insufficient resources".

Reassemble queued incoming segments into receive buffer and return to user. Mark "push seen" (PUSH) if this is the case.

If RCV.UP is in advance of the data currently being passed to the user notify the user of the presence of urgent data.

When the TCP takes responsibility for delivering data to the user that fact must be communicated to the sender via an acknowledgment. The formation of such an acknowledgment is described below in the discussion of processing an incoming segment.

CLOSE-WAIT STATE

Since the remote side has already sent FIN, RECEIVES must be satisfied by text already on hand, but not yet delivered to the user. If no text is awaiting delivery, the RECEIVE will get a "error: connection closing" response. Otherwise, any remaining text can be used to satisfy the RECEIVE.

CLOSING STATE

LAST-ACK STATE

TIME-WAIT STATE

Return "error: connection closing".

CLOSE Call

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

If the user does not have access to such a connection, return "error: connection illegal for this process".

Otherwise, return "error: connection does not exist".

LISTEN STATE

Any outstanding RECEIVES are returned with "error: closing" responses. Delete TCB, enter CLOSED state, and return.

SYN-SENT STATE

Delete the TCB and return "error: closing" responses to any queued SENDs, or RECEIVES.

SYN-RECEIVED STATE

If no SENDs have been issued and there is no pending data to send, then form a FIN segment and send it, and enter FIN-WAIT-1 state; otherwise queue for processing after entering ESTABLISHED state.

ESTABLISHED STATE

Queue this until all preceding SENDs have been segmentized, then form a FIN segment and send it. In any case, enter FIN-WAIT-1 state.

FIN-WAIT-1 STATE

FIN-WAIT-2 STATE

Strictly speaking, this is an error and should receive a "error: connection closing" response. An "ok" response would be acceptable, too, as long as a second FIN is not emitted (the first FIN may be retransmitted though).

CLOSE-WAIT STATE

Queue this request until all preceding SENDs have been segmentized; then send a FIN segment, enter LAST-ACK state.

CLOSING STATE

LAST-ACK STATE

TIME-WAIT STATE

Respond with "error: connection closing".

ABORT Call

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

If the user should not have access to such a connection, return "error: connection illegal for this process".

Otherwise return "error: connection does not exist".

LISTEN STATE

Any outstanding RECEIVES should be returned with "error: connection reset" responses. Delete TCB, enter CLOSED state, and return.

SYN-SENT STATE

All queued SENDS and RECEIVES should be given "connection reset" notification, delete the TCB, enter CLOSED state, and return.

SYN-RECEIVED STATE

ESTABLISHED STATE

FIN-WAIT-1 STATE

FIN-WAIT-2 STATE

CLOSE-WAIT STATE

Send a reset segment:

<SEQ=SND.NXT><CTL=RST>

All queued SENDS and RECEIVES should be given "connection reset" notification; all segments queued for transmission (except for the RST formed above) or retransmission should be flushed, delete the TCB, enter CLOSED state, and return.

CLOSING STATE LAST-ACK STATE TIME-WAIT STATE

Respond with "ok" and delete the TCB, enter CLOSED state, and return.

STATUS Call

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

If the user should not have access to such a connection, return "error: connection illegal for this process".

Otherwise return "error: connection does not exist".

LISTEN STATE

Return "state = LISTEN", and the TCB pointer.

SYN-SENT STATE

Return "state = SYN-SENT", and the TCB pointer.

SYN-RECEIVED STATE

Return "state = SYN-RECEIVED", and the TCB pointer.

ESTABLISHED STATE

Return "state = ESTABLISHED", and the TCB pointer.

FIN-WAIT-1 STATE

Return "state = FIN-WAIT-1", and the TCB pointer.

FIN-WAIT-2 STATE

Return "state = FIN-WAIT-2", and the TCB pointer.

CLOSE-WAIT STATE

Return "state = CLOSE-WAIT", and the TCB pointer.

CLOSING STATE

Return "state = CLOSING", and the TCB pointer.

LAST-ACK STATE

Return "state = LAST-ACK", and the TCB pointer.

TIME-WAIT STATE

Return "state = TIME-WAIT", and the TCB pointer.

SEGMENT ARRIVES

If the state is CLOSED (i.e., TCB does not exist) then

all data in the incoming segment is discarded. An incoming segment containing a RST is discarded. An incoming segment not containing a RST causes a RST to be sent in response. The acknowledgment and sequence field values are selected to make the reset sequence acceptable to the TCP that sent the offending segment.

If the ACK bit is off, sequence number zero is used,

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

If the ACK bit is on,

<SEQ=SEG.ACK><CTL=RST>

Return.

If the state is LISTEN then

first check for an RST

An incoming RST should be ignored. Return.

second check for an ACK

Any acknowledgment is bad if it arrives on a connection still in the LISTEN state. An acceptable reset segment should be formed for any arriving ACK-bearing segment. The RST should be formatted as follows:

<SEQ=SEG.ACK><CTL=RST>

Return.

third check for a SYN

If the SYN bit is set, check the security. If the security/compartment on the incoming segment does not exactly match the security/compartment in the TCB then send a reset and return.

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

If the SEG.PRC is greater than the TCB.PRC then if allowed by the user and the system set TCB.PRC<-SEG.PRC, if not allowed send a reset and return.

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

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

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>

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

Any other control or text-bearing segment (not containing SYN) must have an ACK and thus would be discarded by the ACK processing. An incoming RST segment could not be valid, since it could not have been sent in response to anything sent by this incarnation of the connection. So you are unlikely to get here, but if you do, drop the segment, and return.

If the state is SYN-SENT then

first check the ACK bit

If the ACK bit is set

If SEG.ACK =< ISS, or SEG.ACK > SND.NXT, send a reset (unless the RST bit is set, if so drop the segment and return)

<SEQ=SEG.ACK><CTL=RST>

and discard the segment. Return.

If `SND.UNA < SEG.ACK =< SND.NXT` then the ACK is acceptable. (TODO: in processing Errata ID 3300, it was noted that some stacks in the wild that do not send data on the SYN are just checking that `SEG.ACK == SND.NXT` ... think about whether anything should be said about that here)

second check the RST bit

If the RST bit is set

If the ACK was acceptable then signal the user "error: connection reset", drop the segment, enter CLOSED state, delete TCB, and return. Otherwise (no ACK) drop the segment and return.

third check the security and precedence

If the security/compartiment in the segment does not exactly match the security/compartiment in the TCB, send a reset

If there is an ACK

`<SEQ=SEG.ACK><CTL=RST>`

Otherwise

`<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>`

If there is an ACK

The precedence in the segment must match the precedence in the TCB, if not, send a reset

`<SEQ=SEG.ACK><CTL=RST>`

If there is no ACK

If the precedence in the segment is higher than the precedence in the TCB then if allowed by the user and the system raise the precedence in the TCB to that in the segment, if not allowed to raise the prec then send a reset.

`<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>`

If the precedence in the segment is lower than the precedence in the TCB continue.

If a reset was sent, discard the segment and return.

fourth check the SYN bit

This step should be reached only if the ACK is ok, or there is no ACK, and if the segment did not contain a RST.

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. SND.UNA should be advanced to equal SEG.ACK (if there is an ACK), and any segments on the retransmission queue which are thereby acknowledged should be removed.

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. 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. Set the variables:

```
SND.WND <- SEG.WND
SND.WL1 <- SEG.SEQ
SND.WL2 <- SEG.ACK
```

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.

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 should be held for later processing.

second check the RST bit,

SYN-RECEIVED STATE

If the RST bit is set

If this connection was initiated with a passive OPEN (i.e., came from the LISTEN state), then return this connection to LISTEN state and return. The user need not be informed. If this connection was initiated with an active OPEN (i.e., came from SYN-SENT state) then the connection was refused, signal the user "connection refused". In either case, all segments on the retransmission queue should be removed. And in the active OPEN case, enter the CLOSED state and delete the TCB, and return.

ESTABLISHED

FIN-WAIT-1

FIN-WAIT-2

CLOSE-WAIT

If the RST bit is set then, any outstanding RECEIVES and SEND should receive "reset" responses. All segment queues should be flushed. Users should also receive an unsolicited general "connection reset" signal. Enter the CLOSED state, delete the TCB, and return.

CLOSING STATE

LAST-ACK STATE

TIME-WAIT

If the RST bit is set then, enter the CLOSED state, delete the TCB, and return.

third check security and precedence

SYN-RECEIVED

If the security/compartments and precedence in the segment do not exactly match the security/compartments and precedence in the TCB then send a reset, and return.

ESTABLISHED
FIN-WAIT-1
FIN-WAIT-2
CLOSE-WAIT
CLOSING
LAST-ACK
TIME-WAIT

If the security/compartments and precedence in the segment do not exactly match the security/compartments and precedence in the TCB then send a reset, any outstanding RECEIVES and SEND should receive "reset" responses. All segment queues should be flushed. Users should also receive an unsolicited general "connection reset" signal. Enter the CLOSED state, delete the TCB, and return.

Note this check is placed following the sequence check to prevent a segment from an old connection between these ports with a different security or precedence from causing an abort of the current connection.

fourth, check the SYN bit,

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

TODO: need to incorporate RFC 1122 4.2.2.20(e) here

If the SYN is in the window it is an error, send a reset, any outstanding RECEIVES and SEND should receive "reset" responses, all segment queues should be flushed, the user should also receive an unsolicited general "connection reset" signal, enter the CLOSED state, delete the TCB, and return.

If the SYN is not in the window this step would not be reached and an ack would have been sent in the first step (sequence number check).

fifth check the ACK field,

if the ACK bit is off drop the segment and return

if the ACK bit is on

SYN-RECEIVED STATE

If $\text{SND.UNA} < \text{SEG.ACK} \leq \text{SND.NXT}$ then enter ESTABLISHED state and continue processing with variables below set to:

```
SND.WND <- SEG.WND
SND.WL1 <- SEG.SEQ
SND.WL2 <- SEG.ACK
```

If the segment acknowledgment is not acceptable, form a reset segment,

```
<SEQ=SEG.ACK><CTL=RST>
```

and send it.

ESTABLISHED STATE

If $\text{SND.UNA} < \text{SEG.ACK} \leq \text{SND.NXT}$ then, set $\text{SND.UNA} \leftarrow \text{SEG.ACK}$. Any segments on the retransmission queue which are thereby entirely acknowledged are removed. Users should receive positive acknowledgments for buffers which have been SENT and fully acknowledged (i.e., SEND buffer should be returned with "ok" response). If the ACK is a duplicate ($\text{SEG.ACK} \leq \text{SND.UNA}$), it can be ignored. If the ACK acks something not yet sent ($\text{SEG.ACK} > \text{SND.NXT}$) then send an ACK, drop the segment, and return.

If $\text{SND.UNA} \leq \text{SEG.ACK} \leq \text{SND.NXT}$, the send window should be updated. If ($\text{SND.WL1} < \text{SEG.SEQ}$ or ($\text{SND.WL1} = \text{SEG.SEQ}$ and $\text{SND.WL2} \leq \text{SEG.ACK}$)), set $\text{SND.WND} \leftarrow \text{SEG.WND}$, set $\text{SND.WL1} \leftarrow \text{SEG.SEQ}$, and set $\text{SND.WL2} \leftarrow \text{SEG.ACK}$.

Note that SND.WND is an offset from SND.UNA , that SND.WL1 records the sequence number of the last segment used to update SND.WND , and that SND.WL2 records the acknowledgment number of the last segment used to update SND.WND . The check here prevents using old segments to update the window.

FIN-WAIT-1 STATE

In addition to the processing for the ESTABLISHED state, if our FIN is now acknowledged then enter FIN-WAIT-2 and continue processing in that state.

FIN-WAIT-2 STATE

In addition to the processing for the ESTABLISHED state, if the retransmission queue is empty, the user's CLOSE can be acknowledged ("ok") but do not delete the TCB.

CLOSE-WAIT STATE

Do the same processing as for the ESTABLISHED state.

CLOSING STATE

In addition to the processing for the ESTABLISHED state, if the ACK acknowledges our FIN then enter the TIME-WAIT state, otherwise ignore the segment.

LAST-ACK STATE

The only thing that can arrive in this state is an acknowledgment of our FIN. If our FIN is now acknowledged, delete the TCB, enter the CLOSED state, and return.

TIME-WAIT STATE

The only thing that can arrive in this state is a retransmission of the remote FIN. Acknowledge it, and restart the 2 MSL timeout.

sixth, check the URG bit,

ESTABLISHED STATE

FIN-WAIT-1 STATE

FIN-WAIT-2 STATE

If the URG bit is set, $RCV.UP \leftarrow \max(RCV.UP, SEG.UP)$, and signal the user that the remote side has urgent data if the urgent pointer (RCV.UP) is in advance of the data consumed. If the user has already been signaled (or is still in the "urgent mode") for this continuous sequence of urgent data, do not signal the user again.

CLOSE-WAIT STATE

CLOSING STATE
LAST-ACK STATE
TIME-WAIT

This should not occur, since a FIN has been received from the remote side. Ignore the URG.

seventh, process the segment text,

ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

Once in the ESTABLISHED state, it is possible to deliver segment text to user RECEIVE buffers. Text from segments can be moved into buffers until either the buffer is full or the segment is empty. If the segment empties and carries an PUSH flag, then the user is informed, when the buffer is returned, that a PUSH has been received.

When the TCP takes responsibility for delivering the data to the user it must also acknowledge the receipt of the data.

Once the TCP takes responsibility for the data it advances RCV.NXT over the data accepted, and adjusts RCV.WND as appropriate to the current buffer availability. The total of RCV.NXT and RCV.WND should not be reduced.

Please note the window management suggestions in section 3.7.

Send an acknowledgment of the form:

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

This acknowledgment should be piggybacked on a segment being transmitted if possible without incurring undue delay.

CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

This should not occur, since a FIN has been received from the remote side. Ignore the segment text.

eighth, check the FIN bit,

Do not process the FIN if the state is CLOSED, LISTEN or SYN-SENT since the SEG.SEQ cannot be validated; drop the segment and return.

If the FIN bit is set, signal the user "connection closing" and return any pending RECEIVES with same message, advance RCV.NXT over the FIN, and send an acknowledgment for the FIN. Note that FIN implies PUSH for any segment text not yet delivered to the user.

SYN-RECEIVED STATE
ESTABLISHED STATE

Enter the CLOSE-WAIT state.

FIN-WAIT-1 STATE

If our FIN has been ACKed (perhaps in this segment), then enter TIME-WAIT, start the time-wait timer, turn off the other timers; otherwise enter the CLOSING state.

FIN-WAIT-2 STATE

Enter the TIME-WAIT state. Start the time-wait timer, turn off the other timers.

CLOSE-WAIT STATE

Remain in the CLOSE-WAIT state.

CLOSING STATE

Remain in the CLOSING state.

LAST-ACK STATE

Remain in the LAST-ACK state.

TIME-WAIT STATE

Remain in the TIME-WAIT state. Restart the 2 MSL time-wait timeout.

and return.

USER TIMEOUT

USER TIMEOUT

For any state if the user timeout expires, flush all queues, signal the user "error: connection aborted due to user timeout" in general and for any outstanding calls, delete the TCB, enter the CLOSED state and return.

RETRANSMISSION TIMEOUT

For any state if the retransmission timeout expires on a segment in the retransmission queue, send the segment at the front of the retransmission queue again, reinitialize the retransmission timer, and return.

TIME-WAIT TIMEOUT

If the time-wait timeout expires on a connection delete the TCB, enter the CLOSED state and return.

3.11. Glossary

1822 BBN Report 1822, "The Specification of the Interconnection of a Host and an IMP". The specification of interface between a host and the ARPANET.

ACK A control bit (acknowledge) occupying no sequence space, which indicates that the acknowledgment field of this segment specifies the next sequence number the sender of this segment is expecting to receive, hence acknowledging receipt of all previous sequence numbers.

ARPANET message The unit of transmission between a host and an IMP in the ARPANET. The maximum size is about 1012 octets (8096 bits).

ARPANET packet A unit of transmission used internally in the ARPANET between IMPs. The maximum size is about 126 octets (1008 bits).

connection A logical communication path identified by a pair of sockets.

datagram A message sent in a packet switched computer communications network.

Destination Address The destination address, usually the network and host identifiers.

FIN A control bit (finis) occupying one sequence number, which indicates that the sender will send no more data or control occupying sequence space.

fragment A portion of a logical unit of data, in particular an internet fragment is a portion of an internet datagram.

FTP A file transfer protocol.

header Control information at the beginning of a message, segment, fragment, packet or block of data.

host
A computer. In particular a source or destination of messages from the point of view of the communication network.

Identification
An Internet Protocol field. This identifying value assigned by the sender aids in assembling the fragments of a datagram.

IMP
The Interface Message Processor, the packet switch of the ARPANET.

internet address
A source or destination address specific to the host level.

internet datagram
The unit of data exchanged between an internet module and the higher level protocol together with the internet header.

internet fragment
A portion of the data of an internet datagram with an internet header.

IP
Internet Protocol.

IRS
The Initial Receive Sequence number. The first sequence number used by the sender on a connection.

ISN
The Initial Sequence Number. The first sequence number used on a connection, (either ISS or IRS). Selected in a way that is unique within a given period of time and is unpredictable to attackers.

ISS
The Initial Send Sequence number. The first sequence number used by the sender on a connection.

leader
Control information at the beginning of a message or block of data. In particular, in the ARPANET, the control information on an ARPANET message at the host-IMP interface.

left sequence
This is the next sequence number to be acknowledged by the data receiving TCP (or the lowest currently unacknowledged

sequence number) and is sometimes referred to as the left edge of the send window.

local packet

The unit of transmission within a local network.

module

An implementation, usually in software, of a protocol or other procedure.

MSL

Maximum Segment Lifetime, the time a TCP segment can exist in the internetwork system. Arbitrarily defined to be 2 minutes.

octet

An eight bit byte.

Options

An Option field may contain several options, and each option may be several octets in length. The options are used primarily in testing situations; for example, to carry timestamps. Both the Internet Protocol and TCP provide for options fields.

packet

A package of data with a header which may or may not be logically complete. More often a physical packaging than a logical packaging of data.

port

The portion of a socket that specifies which logical input or output channel of a process is associated with the data.

process

A program in execution. A source or destination of data from the point of view of the TCP or other host-to-host protocol.

PUSH

A control bit occupying no sequence space, indicating that this segment contains data that must be pushed through to the receiving user.

RCV.NXT

receive next sequence number

RCV.UP

receive urgent pointer

RCV.WND

receive window

receive next sequence number

This is the next sequence number the local TCP is expecting to receive.

receive window

This represents the sequence numbers the local (receiving) TCP is willing to receive. Thus, the local TCP considers that segments overlapping the range RCV.NXT to RCV.NXT + RCV.WND - 1 carry acceptable data or control. Segments containing sequence numbers entirely outside of this range are considered duplicates and discarded.

RST

A control bit (reset), occupying no sequence space, indicating that the receiver should delete the connection without further interaction. The receiver can determine, based on the sequence number and acknowledgment fields of the incoming segment, whether it should honor the reset command or ignore it. In no case does receipt of a segment containing RST give rise to a RST in response.

RTP

Real Time Protocol: A host-to-host protocol for communication of time critical information.

SEG.ACK

segment acknowledgment

SEG.LEN

segment length

SEG.PRC

segment precedence value

SEG.SEQ

segment sequence

SEG.UP

segment urgent pointer field

SEG.WND

segment window field

segment

A logical unit of data, in particular a TCP segment is the unit of data transferred between a pair of TCP modules.

segment acknowledgment

The sequence number in the acknowledgment field of the arriving segment.

segment length

The amount of sequence number space occupied by a segment, including any controls which occupy sequence space.

segment sequence

The number in the sequence field of the arriving segment.

send sequence

This is the next sequence number the local (sending) TCP will use on the connection. It is initially selected from an initial sequence number curve (ISN) and is incremented for each octet of data or sequenced control transmitted.

send window

This represents the sequence numbers which the remote (receiving) TCP is willing to receive. It is the value of the window field specified in segments from the remote (data receiving) TCP. The range of new sequence numbers which may be emitted by a TCP lies between SND.NXT and SND.UNA + SND.WND - 1. (Retransmissions of sequence numbers between SND.UNA and SND.NXT are expected, of course.)

SND.NXT

send sequence

SND.UNA

left sequence

SND.UP

send urgent pointer

SND.WL1

segment sequence number at last window update

SND.WL2

segment acknowledgment number at last window update

SND.WND

send window

socket

An address which specifically includes a port identifier, that is, the concatenation of an Internet Address with a TCP port.

Source Address

The source address, usually the network and host identifiers.

SYN

A control bit in the incoming segment, occupying one sequence number, used at the initiation of a connection, to indicate where the sequence numbering will start.

TCB

Transmission control block, the data structure that records the state of a connection.

TCB.PRC

The precedence of the connection.

TCP

Transmission Control Protocol: A host-to-host protocol for reliable communication in internetwork environments.

TOS

Type of Service, an Internet Protocol field.

Type of Service

An Internet Protocol field which indicates the type of service for this internet fragment.

URG

A control bit (urgent), occupying no sequence space, used to indicate that the receiving user should be notified to do urgent processing as long as there is data to be consumed with sequence numbers less than the value indicated in the urgent pointer.

urgent pointer

A control field meaningful only when the URG bit is on. This field communicates the value of the urgent pointer which indicates the data octet associated with the sending user's urgent call.

4. Changes from RFC 793

This document obsoletes RFC 793 as well as RFC 6093 and 6528, which updated 793. In all cases, only the normative protocol specification and requirements have been incorporated into this document, and the

informational text with background and rationale has not been carried in. The informational content of those documents is still valuable in learning about and understanding TCP, and they are valid Informational references, even though their normative content has been incorporated into this document.

The main body of this document was adapted from RFC 793's Section 3, titled "FUNCTIONAL SPECIFICATION", with an attempt to keep formatting and layout as close as possible.

The collection of applicable RFC Errata that have been reported and either accepted or held for an update to RFC 793 were incorporated (Errata IDs: 573, 574, 700, 701, 1283, 1561, 1562, 1564, 1565, 1571, 1572, 2296, 2297, 2298, 2748, 2749, 2934, 3213, 3300, 3301). Some errata were not applicable due to other changes (Errata IDs: 572, 575, 1569, 3602). TODO: 3305

Changes to the specification of the Urgent Pointer described in RFC 1122 and 6093 were incorporated. See RFC 6093 for detailed discussion of why these changes were necessary.

The more secure Initial Sequence Number generation algorithm from RFC 6528 was incorporated. See RFC 6528 for discussion of the attacks that this mitigates, as well as advice on selecting PRF algorithms and managing secret key data.

RFC EDITOR'S NOTE: the content below is for detailed change tracking and planning, and not to be included with the final revision of the document.

The -00 revision of this document was merely a proposal and rough plan for updating RFC 793.

The -01 revision of this document incorporates the content of RFC 793 Section 3 titled "FUNCTIONAL SPECIFICATION". Other content from RFC 793 has not been incorporated. The -01 revision of this document makes some minor formatting changes to the RFC 793 content in order to convert the content into XML2RFC format and account for left-out parts of RFC 793. For instance, figure numbering differs and some indentation is not exactly the same.

The -02 revision of this document incorporates errata that have been verified:

Errata ID 573: Reported by Bob Braden (note: This errata basically is just a reminder that RFC 1122 updates 793. Some of the associated changes are left pending to a separate revision that incorporates 1122. Bob's mention of PUSH in 793 section 2.8 was

not applicable here because that section was not part of the "functional specification". Also the 1122 text on the retransmission timeout also has been updated by subsequent RFCs, so the change here deviates from Bob's suggestion to apply the 1122 text.)

Errata ID 574: Reported by Yin Shuming

Errata ID 700: Reported by Yin Shuming

Errata ID 701: Reported by Yin Shuming

Errata ID 1283: Reported by Pei-chun Cheng

Errata ID 1561: Reported by Constantin Hagemeier

Errata ID 1562: Reported by Constantin Hagemeier

Errata ID 1564: Reported by Constantin Hagemeier

Errata ID 1565: Reported by Constantin Hagemeier

Errata ID 1571: Reported by Constantin Hagemeier

Errata ID 1572: Reported by Constantin Hagemeier

Errata ID 2296: Reported by Vishwas Manral

Errata ID 2297: Reported by Vishwas Manral

Errata ID 2298: Reported by Vishwas Manral

Errata ID 2748: Reported by Mykyta Yevstifeyev

Errata ID 2749: Reported by Mykyta Yevstifeyev

Errata ID 2934: Reported by Constantin Hagemeier

Errata ID 3213: Reported by EugnJun Yi

Errata ID 3300: Reported by Botong Huang

Errata ID 3301: Reported by Botong Huang

Note: Some verified errata were not used in this update, as they relate to sections of RFC 793 elided from this document. These include Errata ID 572, 575, and 1569.

Note: Errata ID 3602 was not applied in this revision as it is duplicative of the 1122 corrections.

There is an errata 3305 currently reported that need to be verified, held, or rejected by the ADs; it is addressing the same issue as draft-gont-tcpm-tcp-seq-validation and was not attempted to be applied to this document.

Not related to RFC 793 content, this revision also makes small tweaks to the introductory text, fixes indentation of the pseudoheader diagram, and notes that the Security Considerations should also include privacy, when this section is written.

The -03 revision of this document revises all discussion of the urgent pointer in order to comply with RFC 6093, 1122, and 1011. Since 1122 held requirements on the urgent pointer, the full list of requirements was brought into an appendix of this document, so that it can be updated as-needed.

The -04 revision of this document includes the ISN generation changes from RFC 6528.

The -05 revision of this document incorporates MSS requirements and definitions from RFC 879, 1122, and 6691, as well as option-handling requirements from RFC 1122.

TODO: Incomplete list of planned changes - these need to be added to and made more specific, as the document proceeds:

1. incorporate 1122 additions
2. point to major additional docs like 1323bis and 5681
3. incorporate relevant parts of 3168 (ECN)
4. incorporate Fernando's new number-checking fixes (if past the IESG in time)
5. point to PMTUD?
6. point to 5461 (soft errors)
7. mention 5961 state machine option
8. mention 6161 (reducing TIME-WAIT)
9. incorporate 6429 (ZWP/persist)
10. look at Tony Sabatini suggestion for describing DO field
11. clearly specify treatment of reserved bits (see TCPM thread on EDO draft April 25, 2014)
12. look at possible mention of draft-minshall-nagle (e.g. as in Linux)
13. make sure that clarifications in RFC 1011 are captured
14. per TCPM discussion, discussion of checking reserved bits may need to be altered from 793
15. MSL acronymn is defined multiple times

5. IANA Considerations

This memo includes no request to IANA. Existing IANA registries for TCP parameters are sufficient.

TODO: check whether entries pointing to 793 and other documents obsoleted by this one should be updated to point to this one instead.

6. Security and Privacy Considerations

TODO

See RFC 6093 [7] for discussion of security considerations related to the urgent pointer field.

Editor's Note: Scott Brim mentioned that this should include a PERPASS/privacy review.

7. Acknowledgements

This document is largely a revision of RFC 793, which Jon Postel was the editor of. Due to his excellent work, it was able to last for three decades before we felt the need to revise it.

Andre Oppermann was a contributor and helped to edit the first revision of this document.

We are thankful for the assistance of the IETF TCPM working group chairs:

Michael Scharf
Yoshifumi Nishida
Pasi Sarolahti

On the TCPM mailing list, and at the IETF 88 meeting in Vancouver, helpful comments, critiques, and reviews were received from (listed alphabetically): David Borman, Yuchung Cheng, Martin Duke, Kevin Lahey, Kevin Mason, Matt Mathis, Hagen Paul Pfeifer, Anthony Sabatini, Joe Touch, Reji Varghese, Lloyd Wood, and Alex Zimmermann.

This document includes content from errata that were reported by (listed chronologically): Yin Shuming, Bob Braden, Morris M. Keesan, Pei-chun Cheng, Constantin Hagemeier, Vishwas Manral, Mykyta Yevstifeyev, EungJun Yi, Botong Huang.

8. References

8.1. Normative References

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

8.2. Informative References

- [2] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
- [3] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.
- [4] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, November 1990.
- [5] Lahey, K., "TCP Problems with Path MTU Discovery", RFC 2923, September 2000.

- [6] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.
- [7] Gont, F. and A. Yourtchenko, "On the Implementation of the TCP Urgent Mechanism", RFC 6093, January 2011.
- [8] Gont, F. and S. Bellovin, "Defending against Sequence Number Attacks", RFC 6528, February 2012.
- [9] Borman, D., "TCP Options and Maximum Segment Size (MSS)", RFC 6691, July 2012.
- [10] Duke, M., Braden, R., Eddy, W., Blanton, E., and A. Zimmermann, "A Roadmap for Transmission Control Protocol (TCP) Specification Documents", RFC 7414, February 2015.

Appendix A. TCP Requirement Summary

This section is adapted from RFC 1122.

TODO: this needs to be seriously redone, to use 793bis section numbers instead of 1122 ones, and all 1122 requirements need to be reflected in 793bis text.

RFC EDITOR'S NOTE: 793bis in the heading below should be replaced by the number of this RFC

FEATURE	RFC1122 SECTION	T	D	Y	O	O	T	e
-----	-----	-	-	-	-	-	-	--
Push flag								
Aggregate or queue un-pushed data	4.2.2.2				x			
Sender collapse successive PSH flags	4.2.2.2				x			
SEND call can specify PUSH	4.2.2.2					x		
If cannot: sender buffer indefinitely	4.2.2.2							x
If cannot: PSH last segment	4.2.2.2	x						
Notify receiving ALP of PSH	4.2.2.2				x			1
Send max size segment when possible	4.2.2.2			x				

Window						
Treat as unsigned number	4.2.2.3	x				
Handle as 32-bit number	4.2.2.3		x			
Shrink window from right	4.2.2.16				x	
Robust against shrinking window	4.2.2.16	x				
Receiver's window closed indefinitely	4.2.2.17			x		
Sender probe zero window	4.2.2.17	x				
First probe after RTO	4.2.2.17		x			
Exponential backoff	4.2.2.17		x			
Allow window stay zero indefinitely	4.2.2.17	x				
Sender timeout OK conn with zero wind	4.2.2.17					x
Urgent Data						
Pointer indicates first non-urgent octet	4.2.2.4	x				
Arbitrary length urgent data sequence	4.2.2.4	x				
Inform ALP asynchronously of urgent data	4.2.2.4	x				1
ALP can learn if/how much urgent data Q'd	4.2.2.4	x				1
TCP Options						
Receive TCP option in any segment	4.2.2.5	x				
Ignore unsupported options	4.2.2.5	x				
Cope with illegal option length	4.2.2.5	x				
Implement sending & receiving MSS option	4.2.2.6	x				
Send MSS option unless 536	4.2.2.6		x			
Send MSS option always	4.2.2.6			x		
Send-MSS default is 536	4.2.2.6	x				
Calculate effective send seg size	4.2.2.6	x				
TCP Checksums						
Sender compute checksum	4.2.2.7	x				
Receiver check checksum	4.2.2.7	x				
ISN Selection						
Include a clock-driven ISN generator component	4.2.2.9	x				
Secure ISN generator with a PRF component	N/A		x			
Opening Connections						
Support simultaneous open attempts	4.2.2.10	x				
SYN-RCVD remembers last state	4.2.2.11	x				
Passive Open call interfere with others	4.2.2.18					x
Function: simultan. LISTENs for same port	4.2.2.18	x				
Ask IP for src address for SYN if necc.	4.2.3.7	x				
Otherwise, use local addr of conn.	4.2.3.7	x				
OPEN to broadcast/multicast IP Address	4.2.3.14					x
Silently discard seg to bcast/mcast addr	4.2.3.14	x				
Closing Connections						

RST can contain data	4.2.2.12	x			
Inform application of aborted conn	4.2.2.13	x			
Half-duplex close connections	4.2.2.13		x		
Send RST to indicate data lost	4.2.2.13	x			
In TIME-WAIT state for 2xMSL seconds	4.2.2.13	x			
Accept SYN from TIME-WAIT state	4.2.2.13		x		
Retransmissions					
Jacobson Slow Start algorithm	4.2.2.15	x			
Jacobson Congestion-Avoidance algorithm	4.2.2.15	x			
Retransmit with same IP ident	4.2.2.15		x		
Karn's algorithm	4.2.3.1	x			
Jacobson's RTO estimation alg.	4.2.3.1	x			
Exponential backoff	4.2.3.1	x			
SYN RTO calc same as data	4.2.3.1		x		
Recommended initial values and bounds	4.2.3.1		x		
Generating ACK's:					
Queue out-of-order segments	4.2.2.20		x		
Process all Q'd before send ACK	4.2.2.20	x			
Send ACK for out-of-order segment	4.2.2.21			x	
Delayed ACK's	4.2.3.2		x		
Delay < 0.5 seconds	4.2.3.2	x			
Every 2nd full-sized segment ACK'd	4.2.3.2	x			
Receiver SWS-Avoidance Algorithm	4.2.3.3	x			
Sending data					
Configurable TTL	4.2.2.19	x			
Sender SWS-Avoidance Algorithm	4.2.3.4	x			
Nagle algorithm	4.2.3.4		x		
Application can disable Nagle algorithm	4.2.3.4	x			
Connection Failures:					
Negative advice to IP on R1 retxs	4.2.3.5	x			
Close connection on R2 retxs	4.2.3.5	x			
ALP can set R2	4.2.3.5	x			1
Inform ALP of R1<=retxs<R2	4.2.3.5		x		1
Recommended values for R1, R2	4.2.3.5		x		
Same mechanism for SYNs	4.2.3.5	x			
R2 at least 3 minutes for SYN	4.2.3.5	x			
Send Keep-alive Packets:	4.2.3.6		x		
- Application can request	4.2.3.6	x			
- Default is "off"	4.2.3.6	x			
- Only send if idle for interval	4.2.3.6	x			
- Interval configurable	4.2.3.6	x			
- Default at least 2 hrs.	4.2.3.6	x			
- Tolerant of lost ACK's	4.2.3.6	x			

IP Options						
Ignore options TCP doesn't understand	4.2.3.8	x				
Time Stamp support	4.2.3.8			x		
Record Route support	4.2.3.8			x		
Source Route:						
ALP can specify	4.2.3.8	x				1
Overrides src rt in datagram	4.2.3.8	x				
Build return route from src rt	4.2.3.8	x				
Later src route overrides	4.2.3.8		x			
Receiving ICMP Messages from IP	4.2.3.9	x				
Dest. Unreach (0,1,5) => inform ALP	4.2.3.9		x			
Dest. Unreach (0,1,5) => abort conn	4.2.3.9					x
Dest. Unreach (2-4) => abort conn	4.2.3.9		x			
Source Quench => slow start	4.2.3.9		x			
Time Exceeded => tell ALP, don't abort	4.2.3.9		x			
Param Problem => tell ALP, don't abort	4.2.3.9		x			
Address Validation						
Reject OPEN call to invalid IP address	4.2.3.10	x				
Reject SYN from invalid IP address	4.2.3.10	x				
Silently discard SYN to bcast/mcast addr	4.2.3.10	x				
TCP/ALP Interface Services						
Error Report mechanism	4.2.4.1	x				
ALP can disable Error Report Routine	4.2.4.1		x			
ALP can specify TOS for sending	4.2.4.2	x				
Passed unchanged to IP	4.2.4.2		x			
ALP can change TOS during connection	4.2.4.2		x			
Pass received TOS up to ALP	4.2.4.2			x		
FLUSH call	4.2.4.3			x		
Optional local IP addr parm. in OPEN	4.2.4.4	x				
-----	-----	-	-	-	-	--

FOOTNOTES: (1) "ALP" means Application-Layer program.

Author's Address

Wesley M. Eddy (editor)
MTI Systems
US

Email: wes@mti-systems.com

TCP Maintenance and Minor Extensions (tcpm)
Internet-Draft
Updates: 793 (if approved)
Intended status: Standards Track
Expires: September 12, 2019

F. Gont
UTN-FRH / SI6 Networks
D. Borman
Quantum Corporation
March 11, 2019

On the Validation of TCP Sequence Numbers
draft-gont-tcpm-tcp-seq-validation-04.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 <https://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 September 12, 2019.

Copyright Notice

Copyright (c) 2019 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 (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. 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	6
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. Alternative fix for TCP sequence number validation	14
4.3. TCP self connects	14
5. IANA Considerations	14
6. Security Considerations	14
7. Acknowledgements	14
8. References	14
8.1. Normative References	15
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]:

----- cut here ----- cut here -----

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

----- cut here ----- cut here -----

is replaced with:

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

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

----- cut here ----- cut here -----

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.

----- cut here ----- cut here -----

is replaced with:

----- cut here ----- cut here -----

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.
----- cut here ----- cut here -----

4.2. Alternative fix for TCP sequence number validation

The Linux kernel performs a slightly different TCP sequence number validation check, that can accommodate window probes of any size (as opposed to the de facto standard 1-byte window probes). This makes the code more general, at the expense of additional state in the TCB (e.g., the TCP sequence number employed in the last window probe).

4.3. 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 leading to connection failures, system crashes, or other undesirable behaviors. This document formally updates RFC 793, such that the aforementioned issues are eliminated.

7. Acknowledgements

The authors of this document would like to thank Theo de Raadt, Rui Paulo and Michael Scharf for providing valuable comments on earlier versions of this document.

8. References

8.1. Normative References

- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

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
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

TCP Maintenance and Minor Extensions (tcpm)
Internet-Draft
Intended status: Informational
Expires: September 10, 2015

M. Kuehlewind, Ed.
ETH Zurich
R. Scheffenegger
NetApp, Inc.
B. Briscoe
BT
March 9, 2015

Problem Statement and Requirements for a More Accurate ECN Feedback
draft-ietf-tcpm-accecn-reqs-08

Abstract

Explicit Congestion Notification (ECN) is a mechanism where network nodes can mark IP packets instead of dropping them to indicate congestion to the end-points. An ECN-capable receiver will feed this information back to the sender. ECN is specified for TCP in such a way that it can only feed back one congestion signal per Round-Trip Time (RTT). In contrast, ECN for other transport protocols, such as RTP/UDP and SCTP, is specified with more accurate ECN feedback. Recent new TCP mechanisms (like ConEx or DCTCP) need more accurate ECN feedback in the case where more than one marking is received in one RTT. This document specifies requirements for an update to the TCP protocol to provide more accurate ECN feedback.

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 September 10, 2015.

Copyright Notice

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

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

Table of Contents

1. Introduction	2
1.1. Terminology	4
2. Recap of Classic ECN and ECN Nonce in IP/TCP	4
3. Use Cases	5
4. Requirements	7
5. Design Approaches	10
5.1. Re-Definition of ECN/NS Header Bits	11
5.2. Using Other Header Bits	12
5.3. Using a TCP Option	12
6. Acknowledgements	13
7. IANA Considerations	13
8. Security Considerations	13
9. References	14
9.1. Normative References	14
9.2. Informative References	14
Appendix A. Ambiguity of the More Accurate ECN Feedback in DCTCP	15
Authors' Addresses	16

1. Introduction

Explicit Congestion Notification (ECN) [RFC3168] is a mechanism where network nodes can mark IP packets instead of dropping them to indicate congestion to the end-points. An ECN-capable receiver will feed this information back 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). This is sufficient for pre-existing TCP congestion control mechanisms that perform only one reduction in sending rate per RTT, independent of the number of ECN congestion marks. But recently proposed or deployed mechanisms like Congestion Exposure (ConEx) [RFC6789] or Data Center TCP (DCTCP) [I-D.bensley-tcpm-dctcp] need more accurate ECN feedback than 'classic' ECN [RFC3168] to work correctly in the case where more than one marking is received in any one RTT.

For an in-depth discussion of the application benefits of using ECN (including with sufficiently granular feedback) see [I-D.welzl-ecn-benefits].

ECN is also defined for transport protocols beside TCP. ECN feedback as defined for RTP/UDP [RFC6679] provides a very detailed level of information, delivering individual counters for all four ECN codepoints as well as lost and duplicate segments, but at the cost of high signaling overhead. ECN feedback for SCTP has been proposed in [I-D.stewart-tsvwg-sctpecn]. This delivers a counter for the number of CE marked segments between CWR chunks, but also comes at the cost of increased overhead.

Today, implementations of DCTCP already exist that alter TCP's ECN feedback protocol in proprietary ways (DCTCP was released in Microsoft Windows 8, and implementations exist for Linux and FreeBSD). The changes DCTCP makes to TCP are not currently the subject of any IETF standardization activity, and they omit capability negotiation, relying instead on uniform configuration across all hosts and network devices with ECN capability. A primary motivation for this document is to intervene before each proprietary implementation invents its own non-interoperable handshake, which could lead to de facto consumption of the few flags or codepoints that remain available for standardizing capability negotiation.

This document lists requirements for a robust and interoperable TCP/ECN feedback protocol that is more accurate than classic ECN [RFC3168] and that all implementations of new TCP extensions, like ConEx and/or DCTCP, can use. While a new feedback scheme should still deliver as much information as classic ECN, this document also clarifies what has to be taken into consideration in addition. Thus the listed requirements should be addressed in the specification of a more accurate ECN feedback scheme. A few solutions have already been proposed. Section 5 demonstrates how to use the requirements to compare them, by briefly sketching their high level design choices and discussing the benefits and drawbacks of each.

The scope of these requirements is not limited to any specific environment and is intended for general deployment over public and private IP networks. Candidate solutions should try to adhere to all these requirements, but where this is not possible they should justify the deviation. The ordering of the requirements listed in this document is not to be taken as an order of importance, because each requirement might have different weight in different deployment scenarios.

These requirements are only concerned with the type and quality of the ECN feedback signal. The requirements do not stipulate how a TCP

sender might react to the improved ECN signal. The requirements also do not imply that any modifications to TCP senders or receivers are obligatory.

1.1. Terminology

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

The ECN field in the IP header:

Not-ECT: the not ECN-Capable Transport codepoint,
CE: the Congestion Experienced codepoint,
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, the ECN feedback scheme as specified in [RFC3168] is called 'classic ECN' and any new proposal is called a 'more accurate ECN feedback' scheme. A 'congestion mark' is defined as an IP packet where the CE codepoint is set. A 'congestion episode' refers to one or more congestion marks that belong to the same overload situation in the network (usually during one RTT). A TCP segment with the acknowledgment flag set is simply called an ACK.

2. Recap of Classic 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. A network node can set both bits simultaneously when it experiences congestion. This leads to the four codepoints (not-ECT, ECT(0), ECT(1), and CE) as listed above.

In the TCP header the first two bits in byte 14 are defined as ECN feedback for each half-connection. A TCP receiver signals the

reception of a congestion mark using the ECN-Echo (ECE) flag in the TCP header. For reliability, the receiver continues to set the ECE flag on every ACK. To enable the TCP receiver to determine when to stop setting the ECN-Echo flag, the sender sets the CWR flag upon reception of an ECE feedback signal. This always leads to a full RTT of ACKs with ECE set. Thus the receiver cannot signal back any additional CE markings arriving within the same RTT.

The ECN Nonce [RFC3540] is an experimental addition to ECN that the TCP sender can use to protect itself against accidental or malicious concealment of CE-marked or dropped packets. This addition defines the last bit of byte 13 in the TCP header as the Nonce Sum (NS) flag. The receiver maintains a nonce sum that counts the occurrence of ECT(1) packets, and signals the least significant bit of this sum on the NS flag. There are no known deployments of a TCP stack that makes use of the ECN Nonce extension.

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

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

An alternative for a sender to assure feedback integrity has been proposed where the sender occasionally inserts a CE mark or reordering itself, and checks that the receiver feeds it back faithfully [I-D.moncaster-tcpm-rcv-cheat]. This alternative consumes no header bits or codepoints, as well as releasing the ECT(1) codepoint in the IP header and the NS flag in the TCP header for other uses.

3. Use Cases

The following two examples serve to show where existing mechanisms would already benefit from more accurate ECN feedback information. However, as it is hard to predict the future, once a more accurate ECN feedback mechanism that adheres to the requirements stated in this document is widely deployed, it's very likely that additional uses are found. The examples listed below are in no particular order.

ConEx is an experimental approach that allows a sender to relay congestion feedback provided by the receiver into the network along the forward data path. ConEx information can be used for traffic management to limit traffic proportionate to the actual congestion

being caused, rather than limiting traffic based on rate or volume [RFC6789]. A ConEx sender uses selective acknowledgements (SACK) [RFC2018] for accurate feedback of loss signals, but currently TCP offers no equivalent accurate feedback for ECN.

DCTCP offers very low and predictable queuing delay. DCTCP changes the reaction to congestion of a TCP sender and additionally requires switches/routers to have ECN enabled and configured with a low step threshold and no signal smoothing, so it is currently only used in private networks, e.g. internal to data centers. DCTCP was released in Microsoft Windows 8, and implementations exist for Linux and FreeBSD. To retrieve sufficient congestion information, the different DCTCP implementations use a proprietary ECN feedback protocol, but they omit capability negotiation. Moreover, the feedback protocol proposed in [I-D.bensley-tcpm-dctcp] only works if there are no losses at all, and otherwise it gets very confused (see Appendix A). Therefore, if a generic more accurate ECN feedback scheme were available, it would solve two problems for DCTCP: i) need for a consistent variant of DCTCP to be deployed network-wide and ii) inability to cope with ACK loss.

Classic ECN-TCP would not benefit from more accurate ECN feedback, but it would not suffer either. The same signal that is currently conveyed with ECN following the specification given in [RFC3168] would be available.

The following scenarios should briefly show where accurate ECN feedback is needed or adds value:

A sender with standardised TCP congestion control that supports ConEx:

In this case the ConEx mechanism uses the extra information per RTT to re-echo the precise congestion information, but the congestion control algorithm still ignores multiple marks per RTT [RFC5681].

A sender using DCTCP congestion control without ConEx:

The congestion control algorithm uses the extra info per RTT to perform its decrease depending on the number of congestion marks.

A sender using DCTCP congestion control and supporting ConEx:

Both the congestion control algorithm and ConEx use the more accurate ECN feedback mechanism.

As-yet-unspecified sender mechanisms:

The above are two examples of more general interest in sender mechanisms that respond to the extent of congestion feedback,

not just its existence. It will greatly simplify incremental deployment if the sender can unilaterally deploy new behaviours, and rely on the presence of generic receivers that have already implemented more accurate feedback.

An RFC5681 TCP sender without ConEx:

No accurate feedback is necessary here. The congestion control algorithm still reacts to only one signal per RTT. But it is best to feed back all the information the receiver gets, whether the sender uses it or not -- at least as long as overhead is low or zero.

Using CE for checking integrity:

If a more accurate ECN feedback scheme feeds all occurrences of CE marks back, a sender could perform integrity checking by occasionally injecting CE marks itself. Specifically, a sender can send packets which it randomly marks with CE (at low frequency), then check if feedback is received for these packets. The congestion notification feedback for these self-injected markings, would not require a congestion control reaction [I-D.moncaster-tcpm-rcv-cheat].

4. Requirements

The requirements of the accurate ECN feedback protocol are to have fairly accurate (not necessarily perfect), timely and protected signaling. This leads to the following requirements, which should be discussed for any proposed more accurate ECN feedback scheme:

Resilience

The ECN feedback signal is carried within the ACK. Pure TCP ACKs can get lost without recovery (not just due to congestion, but also due to deliberate ACK thinning). Moreover, delayed ACKs are commonly used with TCP. Typically, an ACK is triggered after two data segments (or more e.g., due to receive segment coalescing, ACK compression, ACK congestion control [RFC5690] or other phenomena, see [RFC3449]). In a high congestion situation where most of the packets are marked with CE, an accurate feedback mechanism should still be able to signal sufficient congestion information. Thus the accurate ECN feedback extension has to take delayed ACKs and ACK loss into account. Also, a more accurate feedback protocol should still provide more accurate feedback than classic ECN when delayed ACKs cover more than two segments, or when a thin stream disables Nagle's algorithm [RFC0896]. Finally, the feedback mechanism should not be impacted by reordering of ACKs, even when the ACK'ed sequence number does not increase.

Timeliness

A CE mark can be induced by the sending host, or more commonly a network node on the transmission path, and is then echoed by the receiver in the TCP ACK. Thus when this information arrives at the sender, it is naturally already about one RTT old. With a sufficient ACK rate a further delay of a small number of packets can be tolerated. However, this information will become stale with large delays, given the dynamic nature of networks. TCP congestion control (which itself partly introduces these dynamics) operates on a time scale of one RTT. Thus, to be timely, congestion feedback information should be delivered within about one RTT.

Integrity

The integrity of the feedback in a more accurate ECN feedback scheme should be assured, at least as well as the ECN Nonce. Alternatively, it should at least be possible to give strong incentives for the receiver and network nodes to cooperate honestly.

Given there are known problems with ECN Nonce deployment, this document only requires that the integrity of the more accurate ECN feedback can be assured; it does not require that the ECN Nonce mechanism is employed to achieve this. Indeed, if integrity could be provided else-wise, a more accurate ECN feedback protocol might re-purpose the nonce sum (NS) flag in the TCP header.

If the more accurate ECN feedback scheme provides sufficient information, the integrity check could e.g. be performed by deterministically setting the CE in the sender and monitoring the respective feedback (similar to ECT(1) and the ECN Nonce sum). Whether a sender should enforce when it detects wrong feedback information, and what kind of enforcement it should apply, are policy issues that need not be specified as part of more accurate ECN feedback signal scheme itself, but rather when specifying an update to core TCP mechanisms like congestion control that makes use of the more accurate ECN signal.

Accuracy

Classic ECN feeds back one congestion notification per RTT, which is sufficient for classic TCP congestion control which reduces the sending rate at most once per RTT. Thus the more accurate ECN feedback scheme should ensure that, if a congestion episode occurs, at least one congestion notification is echoed and received per RTT as classic ECN

would do. Of course, the goal of a more accurate ECN extension is to reconstruct the number of CE markings more accurately. In the best case the new scheme should even allow reconstruction of the exact number of payload bytes that a CE marked packet was carrying. However, it is accepted that it may be too complex for a sender to get the exact number of congestion markings or marked bytes in all situations. Ideally, the feedback scheme should preserve the order in which any (of the four) ECN signals were received. And, ideally, it would even be possible for the sender to determine which of the packets covered by one delayed ACK were congestion marked, e.g. if the flow consists of packets of different sizes, or to allow for future protocols where the order of the markings may be important.

In the best case, a sender that sees more accurate ECN feedback information would be able to reconstruct the occurrence of any of the four code points (non-ECT, CE, ECT(0), ECT(1)). However, assuming the sender marks all data packets as ECN-capable and uses a default setting of ECT(0) (as with [RFC3168], solely feeding back the occurrence of CE and ECT(1) might be sufficient. Because the sender can keep account of the transmitted segments with any of the three ECN codepoints, conveying any two of these back to the sender is sufficient for it to reconstruct the third as observed by the receiver. Thus a more accurate ECN feedback scheme should at least provide information on two of these signals, e.g. CE and ECT(1).

If a more accurate ECN scheme can reliably deliver feedback in most but not all circumstances, ideally the scheme should at least not introduce bias. In other words, undetected loss of some ACKs should be as likely to increase as decrease the sender's estimate of the probability of ECN marking.

Complexity

Implementation should be as simple as possible and only a minimum of additional state information should be needed. This will enable more accurate ECN feedback to be used as the default feedback mechanism, even if only one ECN feedback signal per RTT is needed.

Overhead

A more accurate ECN feedback signal should limit the additional network load, because ECN feedback is ultimately not critical information (in the worst case, loss will still be available as a congestion signal of last resort). As feedback information has to be provided frequently and in a

timely fashion, potentially all or a large fraction of TCP acknowledgments might carry this information. Ideally, no additional segments should be exchanged compared to an RFC3168 TCP session, and the overhead in each segment should be minimized.

Backward and forward compatibility

Given more accurate ECN feedback will involve a change to the TCP protocol, it should be negotiated between the two TCP endpoints. If either end does not support the more accurate feedback, they should both be able to fall-back to classic ECN feedback.

A more accurate ECN feedback extension should aim to traverse most middleboxes, including firewalls and network address translators (NAT). Further, a feedback mechanism should provide a method to fall back to classic ECN signaling if the new signal is suppressed by certain middleboxes.

In order to avoid a fork in the TCP protocol specifications, if experiments with the new ECN feedback protocol are successful, it is intended to eventually update RFC3168 for any TCP/ECN sender, not just for ConEx or DCTCP senders. Then future senders will be able to unilaterally deploy new behaviours that exploit the existence of more accurate ECN feedback in receivers (forward compatibility). Conversely, even if another sender only needs one ECN feedback signal per RTT, it should be able to use more accurate ECN feedback, and simply ignore the excess information.

Furthermore, the receiver should not make assumptions about the mechanism that was used to set the markings nor about any interpretation or reaction to the congestion signal. The receiver only needs to faithfully reflect congestion information back to the sender.

5. Design Approaches

This section introduces some possible TCP ECN feedback design approaches. The purpose of this section is to give examples of how trade-offs might be needed between the requirements, as input to future IETF work to specify a protocol. The order is not significant and there is no intention to endorse any particular approach.

All approaches presented below (and proposed so far) are able to provide accurate ECN feedback information as long as no ACK loss occurs and the congestion rate is reasonable. In the case of a high ACK loss rate or very high congestion (CE marking) rate, the proposed

schemes have different resilience characteristics depending on the number of bits used for the encoding. While classic ECN provides reliable (but inaccurate) feedback of a maximum of one congestion signal per RTT, the proposed schemes do not implement an explicit acknowledgement mechanism for the feedback (as e.g. the ECE / CWR exchange of [RFC3168]).

5.1. Re-Definition of ECN/NS Header Bits

Schemes in this category can additionally use the NS bit for capability negotiation during the TCP handshake exchange. Thus a more accurate ECN could be negotiated without changing the classic ECN negotiation and thus being backwards compatible.

Schemes in this category can simply re-define the ECN header flags, ECE and CWR, to encode the occurrence of a CE marking at the receiver. This approach provides very limited resilience against loss of ACK, particularly pure ACKs (no payload and therefore delivered unreliably).

A couple of schemes have been proposed so far:

- o A naive one-bit scheme that sends one ECE for each CE received could use CWR to increase robustness against ACK loss by introducing redundant information on the next ACK, but this is still vulnerable to ACK loss.
- o The scheme defined for DCTCP [I-D.bensley-tcpm-dctcp], which toggles the ECE feedback on an immediate ACK whenever the CE marking changes, and otherwise feeds back delayed ACKs with the ECE value unchanged. Appendix A demonstrates that this scheme is still ambiguous to the sender if the ACKs are pure ACKs, and if some may have been lost.

Alternatively, the receiver uses the three ECN/NS header flags, ECE, CWR and NS to represent a counter that signals the accumulated number of CE markings it has received. Resilience against loss is better than the flag-based schemes, but may not suffice in the presence of extended ACK loss that otherwise would not affect the TCP sender's performance.

A number of coding schemes have been proposed so far in this category:

- o A 3-bit counter scheme continuously feeds back the three least significant bits of a CE counter;

- o A scheme that defines a standardised lookup table to map the 8 codepoints onto either a CE counter or an ECT(1) counter.

These proposed schemes provide accumulated information on ECN-CE marking feedback, similar to the number of acknowledged bytes in the TCP header. Due to the limited number of bits the ECN feedback information will wrap much more often than the acknowledgement field. Thus feedback information could be lost due to a relatively small sequence of pure-ACK losses. Resilience could be increased by introducing redundancy, e.g. send each counter increase two or more times. Of course any of these additional mechanisms will increase the complexity. If the congestion rate is greater than the ACK rate (multiplied by the number of congestion marks that can be signaled per ACK), the congestion information cannot correctly be fed back. Covering the worst case where every packet is CE marked can potentially be realized by dynamically adapting the ACK rate and redundancy. This again increases complexity and perhaps the signaling overhead as well. Schemes that do not re-purpose the ECN NS bit, could still support the ECN Nonce.

5.2. Using Other Header Bits

As seen in Figure 1, there are currently three unused flags in the TCP header. The proposed 3-bit counter or codepoint schemes could be extended by one or more bits to add higher resilience against ACK loss. The relative gain would be exponentially higher resilience against ACK loss, while the respective drawbacks would remain identical.

Alternatively, a new method could standardise the use of the bits in the Urgent Pointer field (see [RFC6093]) to signal more bits of its congestion signal counter, but only whenever it does not set the Urgent Flag. As this is often the case, resilience could be increased without additional header overhead.

Any proposal to use such bits would need to check the likelihood that some middleboxes might discard or 'normalize' the currently unused flag bits or a non-zero Urgent Pointer when the Urgent Flag is cleared. If during experimentation certain bits have been proven to be usable, the assignment of any of these bits would then require an IETF standards action.

5.3. Using a TCP Option

Alternatively, a new TCP option could be introduced, to help maintain the accuracy and integrity of ECN feedback between receiver and sender. Such an option could provide higher resilience and even more information, perhaps as much as ECN for RTP/UDP [RFC6679], which

explicitly provides the number of ECT(0), ECT(1), CE, non-ECT marked and lost packets, or as much as a proposal for SCTP that counts the number of ECN marks [I-D.stewart-tsvwg-sctpecn] between CWR chunks. However, deploying new TCP options has its own challenges. Moreover, to actually achieve high resilience, this option would need to be carried by most or all ACKs as the receiver cannot know if and when ACKs may be dropped. Thus this approach would introduce considerable signaling overhead even though ECN feedback is not extremely critical information (in the worst case, loss will still be available to provide a strong congestion feedback signal). Whatever, such a TCP option could be used in addition to a more accurate ECN feedback scheme in the TCP header or in addition to classic ECN, only when needed and when space is available.

6. Acknowledgements

Thanks to Gorry Fairhurst for his review and for ideas on CE-based integrity checking and to Mohammad Alizadeh for suggesting the need to avoid bias.

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

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

ECN feedback information must only be used if the other information contained in a received TCP segment indicates that the congestion was genuinely part of the flow and not spoofed - i.e. the normal TCP acceptance techniques have to be used to verify that the segment is part of the flow before returning any contained ECN information, and similarly ECN feedback is only accepted on valid ACKs.

Given ECN feedback is used as input for congestion control, the respective algorithm would not react appropriately if ECN feedback were lost and the resilience mechanism to recover it was inadequate. This resilience requirement is articulated in Section 4. However, it should be noted that ECN feedback is not the last resort against congestion collapse, because if there is insufficient response to ECN, loss will ensue, and TCP will still react appropriately to loss.

A receiver could suppress ECN feedback information leading to its connections consuming excess sender or network resources. This problem is similar to that seen with the classic ECN feedback scheme and should be addressed by integrity checking as required in Section 4.

9. References

9.1. Normative References

- [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.

9.2. Informative References

- [I-D.bensley-tcpm-dctcp] sbens@microsoft.com, s., Eggert, L., and D. Thaler, "Microsoft's Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters", draft-bensley-tcpm-dctcp-02 (work in progress), January 2015.
- [I-D.moncaster-tcpm-rcv-cheat] Moncaster, T., Briscoe, B., and A. Jacquet, "A TCP Test to Allow Senders to Identify Receiver Non-Compliance", draft-moncaster-tcpm-rcv-cheat-03 (work in progress), July 2014.
- [I-D.stewart-tsvwg-sctpecn] Stewart, R., Tuexen, M., and X. Dong, "ECN for Stream Control Transmission Protocol (SCTP)", draft-stewart-tsvwg-sctpecn-05 (work in progress), January 2014.
- [I-D.welzl-ecn-benefits] Welzl, M. and G. Fairhurst, "The Benefits to Applications of using Explicit Congestion Notification (ECN)", draft-welzl-ecn-benefits-01 (work in progress), July 2014.
- [RFC0896] Nagle, J., "Congestion control in IP/TCP internetworks", RFC 896, January 1984.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.

- [RFC3449] Balakrishnan, H., Padmanabhan, V., Fairhurst, G., and M. Sooriyabandara, "TCP Performance Implications of Network Path Asymmetry", BCP 69, RFC 3449, December 2002.
- [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.
- [RFC6093] Gont, F. and A. Yourtchenko, "On the Implementation of the TCP Urgent Mechanism", RFC 6093, January 2011.
- [RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", RFC 6679, August 2012.
- [RFC6789] Briscoe, B., Woundy, R., and A. Cooper, "Congestion Exposure (ConEx) Concepts and Use Cases", RFC 6789, December 2012.

Appendix A. Ambiguity of the More Accurate ECN Feedback in DCTCP

As defined in [I-D.bensley-tcpm-dctcp], a DCTCP receiver feeds back ECE=0 on delayed ACKs as long as CE remains 0, and also immediately sends an ACK with ECE=0 when CE transitions to 1. Similarly, it continually feeds back ECE=1 on delayed ACKs while CE remains 1 and immediately feeds back ECE=1 when CE transitions to 0. A sender can unambiguously decode this scheme if there is never any ACK loss, and the sender assumes there will never be any ACK loss.

The following two examples show that the feedback sequence becomes highly ambiguous to the sender, if either of these conditions is broken. Below, '0' will represent ECE=0, '1' will represent ECE=1 and '.' will represent a gap of one segment between delayed ACKs. Now imagine that the sender receives the following sequence of feedback on 3 pure ACKs:

0.0.0

When the receiver sent this sequence it could have been any of the following four sequences:

- a. 0.0.0 (0 x CE)
- b. 010.0 (1 x CE)

c. 0.010 (1 x CE)

d. 01010 (2 x CE)

where any of the 1s represent a possible pure ACK carrying ECE feedback that could have been lost. If the sender guesses (a), it might be correct, or it might miss 1 or 2 congestion marks over 5 packets. Therefore, when confronted with this simple sequence (that is not contrived), a sender can guess that congestion might have been 0%, 20% or 40%, but it doesn't know which.

Sequences with a longer gap (e.g. 0...0.0) become far more ambiguous. It helps a little if the sender knows the distance the receiver uses between delayed ACKs, and it helps a lot if the distance is 1, i.e. no delayed ACKs, but even then there will still be ambiguity whenever there are pure ACK losses.

Authors' Addresses

Mirja Kuehlewind (editor)
ETH Zurich
Gloriastrasse 35
Zurich 8092
Switzerland

Email: mirja.kuehlewind@tik.ee.ethz.ch

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

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

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

Phone: +44 1473 645196
Email: bob.briscoe@bt.com
URI: <http://bobbbriscoe.net/>

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

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

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

Abstract

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

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

Status of This Memo

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

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

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

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

Copyright Notice

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

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

Table of Contents

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

1. Introduction

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

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

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

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

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

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

Section 3 defines relevant terminology.

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

The goals of this update are:

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

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

1.1. Implementation of new CWV

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

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

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

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

1.2. Standards Status of this Document

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

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

2. Reviewing experience with TCP-CWV

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

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

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

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

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

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

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

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

3. Terminology

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

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

The following additional terminology is introduced in this document:

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

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

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

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

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

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

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

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

4. A New Congestion Window Validation method

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

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

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

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

4.1. Initialisation

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

4.2. Estimating the validated capacity supported by a path

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

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

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

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

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

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

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

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

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

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

4.3. Preserving cwnd during a rate-limited period.

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

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

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

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

4.4. TCP congestion control during the non-validated phase

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

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

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

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

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

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

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

4.4.1. Response to congestion in the non-validated phase

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

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

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

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

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

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

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

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

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

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

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

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

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

4.4.2. Sender burst control during the non-validated phase

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

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

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

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

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

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

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

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

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

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

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

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

4.5. Examples of Implementation

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

4.5.1. Implementing the pipeACK measurement

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

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

pipeACK sample (Bytes)

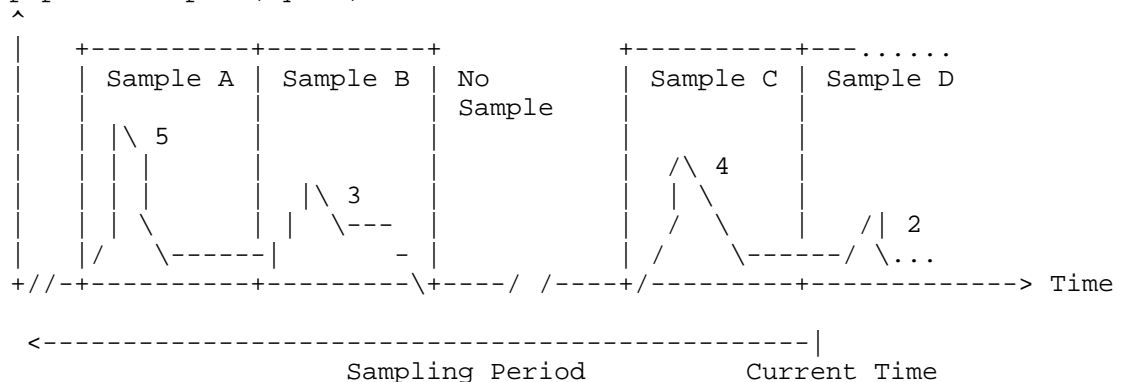


Figure 1: Example of measuring pipeACK samples

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

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

4.5.2. Measurement of the NVP and pipeACK samples

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

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

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

4.5.3. Implementing detection of the cwnd-limited condition

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

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

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

5. Determining a safe period to preserve cwnd

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

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

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

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

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

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

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

6. Security Considerations

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

7. IANA Considerations

There are no IANA considerations.

8. Acknowledgments

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

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

9. Author Notes

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

9.1. Other related work

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

There are several issues to be discussed more widely:

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

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

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

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

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

The behaviour differs in three ways:

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

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

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

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

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

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

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

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

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

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

10. Revision notes

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

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

Draft 04 contained the first set of clarifications after feedback:

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

Draft 05 contained various updates:

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

Draft 06 contained various updates:

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

WG draft 00 contained various updates:

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

WG draft 01 contained:

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

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

WG draft 02 contained:

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

WG draft 03 contained:

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

WG draft 04 contained:

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

WG draft 05 contained:

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

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

WG draft 06 contained:

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

WG draft 07 contained:

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

WG draft 08 contained:

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

WG draft 09 contained:

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

WG draft 10 contained edits following WGLC:

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

WG draft 11 contained edits following IETF LC:

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

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

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

WG draft 13 contained edits following IESG Review:

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

11. References

11.1. Normative References

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

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

11.2. Informative References

- [All05] Allman, M. and E. Blanton, "Notes on burst mitigation for
transport protocols", March 2005.
- [Bis08] Biswas, I. and G. Fairhurst, "A Practical Evaluation of
Congestion Window Validation Behaviour, 9th Annual
Postgraduate Symposium in the Convergence of
Telecommunications, Networking and Broadcasting (PGNet),
Liverpool, UK", June 2008.
- [Bis10] Biswas, I., Sathiaselalan, A., Secchi, R., and G.
Fairhurst, "Analysing TCP for Bursty Traffic, Int'l J. of
Communications, Network and System Sciences, 7(3)", June
2010.
- [Bis11] Biswas, I., "PhD Thesis, Internet congestion control for
variable rate TCP traffic, School of Engineering,
University of Aberdeen", June 2011.
- [Fail2] Sathiaselalan, A., Secchi, R., Fairhurst, G., and I.
Biswas, "Enhancing TCP Performance to support Variable-
Rate Traffic, 2nd Capacity Sharing Workshop, ACM CoNEXT,
Nice, France, 10th December 2012.", June 2008.
- [Hos15] Hossain, Z., "PhD Thesis, A Study of Mechanisms to Support
Variable-rate Internet Applications over a Multi-service
Satellite Platform, School of Engineering, University of
Aberdeen", January 2015.
- [Hug01] Hughes, A., Touch, J., and J. Heidemann, "Issues in TCP
Slow-Start Restart After Idle (Work-in-Progress)",
December 2001.
- [Liu07] Liu, D., Allman, M., Jin, S., and L. Wang, "Congestion
Control without a Startup Phase, 5th International
Workshop on Protocols for Fast Long-Distance Networks
(PFLDnet), Los Angeles, California, USA", February 2007.

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

Authors' Addresses

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

Email: gorry@erg.abdn.ac.uk
URI: <http://www.erg.abdn.ac.uk>

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

Email: arjuna@erg.abdn.ac.uk
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
URI: <http://www.erg.abdn.ac.uk>

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

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

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

Abstract

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

Status of This Memo

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

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

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

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

Copyright Notice

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

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

1. Introduction

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

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

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

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

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

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

2. Terminology

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

This document introduces the following variables:

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

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

3. RTO Overview and Rationale for RTOR

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

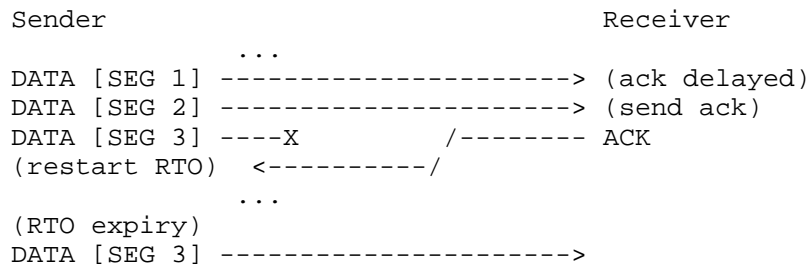


Figure 1: RTO restart example

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

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

4. RTOR Algorithm

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

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

When an ACK is received that acknowledges new data:

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

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

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

5. Discussion

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

5.1. Applicability

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

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

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

5.2. Spurious Timeouts

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

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

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

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

5.3. Tracking Outstanding and Previously Unsent Segments

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

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

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

6. Related Work

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

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

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

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

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

7. SCTP Socket API Considerations

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

Please note that this section is informational only.

7.1. Data Types

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

7.2. Socket Option for Controlling the RTO Restart Support (SCTP_RTO_RESTART)

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

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

This socket option uses IPPROTO_SCTP as its level and SCTP_RTO_RESTART as its name. It can be used with getsockopt() and setsockopt(). The socket option value uses the following structure defined in [RFC6458]:

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

assoc_id: This parameter is ignored for one-to-one style sockets. For one-to-many style sockets, this parameter indicates upon which association the user is performing an action. The special sctp_assoc_t SCTP_{FUTURE|CURRENT|ALL}_ASSOC can also be used in assoc_id for setsockopt(). For getsockopt(), the special value SCTP_FUTURE_ASSOC can be used in assoc_id, but it is an error to use SCTP_{CURRENT|ALL}_ASSOC in assoc_id.

assoc_value: A non-zero value encodes the enabling of RTO restart whereas a value of 0 encodes the disabling of RTO restart.

sctp_opt_info() needs to be extended to support SCTP_RTO_RESTART.

8. IANA Considerations

This memo includes no request to IANA.

9. Security Considerations

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

10. Acknowledgements

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

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

11. Changes from Previous Versions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

12. References

12.1. Normative References

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

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

12.2. Informative References

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

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

Authors' Addresses

Per Hurtig
Karlstad University
Universitetsgatan 2
Karlstad 651 88
Sweden

Phone: +46 54 700 23 35
Email: per.hurtig@kau.se

Anna Brunstrom
Karlstad University
Universitetsgatan 2
Karlstad 651 88
Sweden

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

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

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

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

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

TCP Maintenance and Minor Extensions
(TCPM) WG
Internet-Draft
Obsoletes: 4614 (if approved)
Intended status: Informational
Expires: February 13, 2015

M. Duke
F5
R. Braden
ISI
W. Eddy
MTI Systems
E. Blanton

A. Zimmermann
NetApp, Inc.
August 12, 2014

A Roadmap for Transmission Control Protocol (TCP) Specification
Documents
draft-ietf-tcpm-tcp-rfc4614bis-08

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.

This document obsoletes RFC 4614.

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 February 13, 2015.

Copyright Notice

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

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

Table of Contents

1. Introduction	4
2. Core Functionality	5
3. Strongly Encouraged Enhancements	8
3.1. Fundamental Changes	8
3.2. Congestion Control Extensions	9
3.3. Loss Recovery Extensions	11
3.4. Detection and Prevention of Spurious Retransmissions	12
3.5. Path MTU Discovery	13
3.6. Header Compression	14
3.7. Defending Spoofing and Flooding Attacks	15
4. Experimental Extensions	17
4.1. Architectural Guidelines	17
4.2. Fundamental Changes	18
4.3. Congestion Control Extensions	18
4.4. Loss Recovery Extensions	20
4.5. Detection and Prevention of Spurious Retransmissions	20
4.6. TCP Timeouts	21
4.7. Multipath TCP	21
5. TCP Parameters at IANA	22
6. Historic and Undeployed Extensions	23
7. Support Documents	26
7.1. Foundational Works	26
7.2. Architectural Guidelines	28
7.3. Difficult Network Environments	29
7.4. Guidance for Developing, Analyzing, and Evaluating TCP	32
7.5. Implementation Advice	33
7.6. Tools and Tutorials	35
7.7. MIB Modules	36
7.8. Case Studies	37
8. Undocumented TCP Features	38
9. Security Considerations	40
10. IANA Considerations	40
11. Acknowledgments	40
12. References	40
12.1. Normative References	40
12.2. Informative References	50
Authors' Addresses	51

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 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 [RFC1122] 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 [RFC5681] is considered part of the required core functionality of TCP, although the RFC 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 three 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 [RFC2119]), 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.

Section 5 describes both the procedures that the Internet Assigned Numbers Authority (IANA) uses and an RFC author should follow when new TCP parameters are requested and finally assigned.

A small number of older experimental extensions that have not been widely implemented, deployed, and used are noted in Section 6. Many other supporting documents that are relevant to the development, implementation, and deployment of TCP are described in Section 7.

A small number of fairly ubiquitous important implementation practices that are not currently documented in the RFC series are listed in Section 8.

Within each section, RFCs are listed in the chronological order of their publication dates.

2. Core Functionality

A small number of documents compose the core specification of TCP. These define the required core 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 (see Section 2) 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 (see Section 2), 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 (see Section 3.1) 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 Section 2 and Differentiated Services [RFC2474].

RFC 5681 S: "TCP Congestion Control" (August 2009)

Although RFC 793 (see Section 2) did not contain any congestion control mechanisms, today congestion control is a required component of TCP implementations. This document [RFC5681] defines congestion avoidance and control mechanism for TCP, based on Van Jacobson's 1988 SIGCOMM paper [Jac88].

A number of behaviors that together constitute what the community refers to as "Reno TCP" is 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 5681 details the currently accepted congestion control mechanism, while RFC 1122 Section 2 mandates that such a congestion control mechanism must be implemented. 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] (see Section 3.2).
- (b) During slow start and congestion avoidance, the usage of Appropriate Byte Counting [RFC3465] (see Section 3.2) is explicitly recommended.
- (c) The use of Limited Transmit [RFC3042] (see Section 3.3) 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. The document updates the relevant specifications such that it accommodates 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 (see Section 2) and upgrades the requirement of supporting the algorithm from a SHOULD to a MUST." [RFC6298]. RFC 6298 updates RFC 2988 by changing 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. Strongly Encouraged Enhancements

This section describes recommended TCP modifications that improve performance and security. Section 3.1 represents fundamental changes to the protocol. Section 3.2 and Section 3.3 list improvements over the congestion control and loss recovery mechanisms as specified in RFC 5681 (see Section 2). Section 3.4 describes algorithms that allow a TCP sender to detect whether it has entered loss recovery spuriously. Section 3.5 comprises Path MTU Discovery mechanisms. Schemes for TCP/IP header compression are listed in Section 3.6. Finally, Section 3.7 deals with the problem of preventing acceptance of forged segments and flooding attacks.

3.1. Fundamental Changes

RFCs 2675 and 7323 represent fundamental changes to TCP by redefining how parts of the basic TCP header and options are interpreted. RFC 7323 defines the Window Scale Option, which re-interprets the advertised receive window. RFC 2675 specifies that MSS option and

urgent pointer fields with a value of 65,535 are to be treated specially.

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 7323 S: "TCP Extensions for High Performance" (July 2014)

This document [I-D.ietf-tcpm-1323bis] 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. The predecessor of this document, RFC 1323, was published in 1992, and is deployed in most TCP implementations. This document includes fixes and clarifications based on the gained deployment experience. One specific issued addressed in this specification is a recommendation how to modify the algorithm for estimating the mean RTT when timestamps are used. RFC 1072, RFC 1185, and RFC 1323 are the conceptual precursors of RFC 7323.

3.2. Congestion Control Extensions

Two of the most important aspects of TCP are its congestion control and loss recovery features. TCP 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 these two sub-sections, we group enhancements to TCP's congestion control, while the next sub-section focus on TCP's loss recovery.

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 (see Section 2) to define two previously unused flag bits in the TCP header for ECN support. RFC 3540 (see Section 4.3) provides a supplementary (experimental) means for more secure use of ECN, and RFC 2884 (see Section 7.8) provides some sample results from using ECN.

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 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. This change improves the performance of TCP in situations where is no one-to-one relationship between data segments and acknowledgments (e.g. delayed ACKs or ACK loss) and closes a security hole TCP receivers can use to induce the sender into increasing the sending rate too rapidly (ACK-division [SCWA99][RFC3449]). ABC is recommended by RFC 5681 (see Section 2).

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 recommends against the implementation of [RFC1016].

3.3. Loss Recovery Extensions

For the typical implementation of the TCP fast recovery algorithm described in RFC 5681 (see Section 2), a TCP sender only retransmits a segment after a retransmit timeout has occurred, or after three duplicate ACKs have arrived triggering the fast retransmit. A single RTO might result in the retransmission of several segments, while the fast retransmit algorithm in RFC 5681 leads only to a single retransmission. Hence, multiple losses from a single window of data can lead to a performance degradation. Documents listed in this section aim to improve the overall performance of TCP's standard loss recovery algorithms. In particular, some of them allow TCP senders to recover more effectively when multiple segments are lost from a single flight of data.

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 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]. Limited Transmit is recommended by RFC 5681 (see Section 2).

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.

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] (see Section 3.3). The algorithm conforms to the spirit of the congestion control specification in RFC 5681 (see Section 2), 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 concerns about IPR status. The Standards Track documents in this section are closely related to the Experimental documents in Section 4.5 also addressing this topic.

RFC 2883 S: "An Extension to the Selective Acknowledgement (SACK) Option for TCP" (July 2000)

This document [RFC2883] extends RFC 2018 (see Section 3.3). 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. A TCP sender could then use this information to detect spurious retransmissions (see [RFC3708]).

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 (see Section 4.5), the algorithm in RFC 3708 (see Section 4.5), or F-RTO in RFC 5682 (see Section 3.4).

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 described in RFC 4138, provides an option for inferring spurious retransmission timeouts. Unlike some similar detection methods (e.g. RFC 3522 in Section 4.5 and RFC 3708 in Section 4.5), 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 signaling 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 (see Section 3.5), 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 (see Section 3.5) and RFC 1981 (see Section 3.5), 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 compression schemes for TCP/IP like "RObust Header Compression (ROHC) can significantly compress this 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 from 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 reset (send RST) spoofing attacks. The document analyzes 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 analyzes 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] (see Section 3.7).

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) [RFC5925] (see Section 3.7) 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, 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 (see Section 2).

4. Experimental Extensions

The RFCs in this section are either experimental and may become proposed standards in the future or are proposed standard (or informational), but can be considered as experimental due to lack of wide deployment. 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.

If the experimental RFC is a proposal for a new protocol capability or service, i.e., it requires a new TCP option code point, the implementation and experimentation should follow [RFC6994] (see Section 5), which describes how the experimental TCP option code points can concurrently support multiple TCP extensions.

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 (see Section 4.1). 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. 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. Fundamental Changes

Like the standard documents listed in Section 3.1, there also exist new Experimental RFCs that specify fundamental changes to TCP. At the time of writing, the only example so far is TCP Fast Open that deviates from the standard TCP semantics of [RFC0793].

RFC XXX E: "TCP Fast Open" (XXX 2014)

This document [I-D.ietf-tcpm-fastopen] describes TCP Fast Open that allows data to be carried in the SYN and SYN-ACK packets and consumed by the receiver during the initial connection handshake. It saves up to one RTT compared to the standard TCP, which requires a three-way handshake to complete before data can be exchanged.

4.3. Congestion Control Extensions

TCP congestion control has been an extremely active research area for many years (see RFC 5783, Section 7.6), as it determines the performance of many applications that use TCP. A number of experimental RFCs address issues with flow start-up, overshoot, and steady-state behavior in the basic RFC 5681 (see Section 2) algorithms. In these sub-sections, enhancements to TCP's congestion control are listed. The next sub-section focuses on TCP's loss recovery.

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 (see Section 2), 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] (see Section 3.2) 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 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 (see Section 3.2), to 10 segments with a fallback to the existing recommendation when performance issues are detected.

4.4. Loss Recovery Extensions

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 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.

4.5. 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.

4.6. TCP Timeouts

Besides the well-known retransmission timeout the TCP standard [RFC0793] defines other timeouts. This section lists documents that deal with TCP's various timeouts.

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.

4.7. 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 7.2, 7.4, and 7.5 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. TCP Parameters at IANA

RFCs listed here describes both the procedures that the Internet Assigned Numbers Authority (IANA) uses when handling assignments and the procedures an RFC author should follow when requesting new TCP option codepoints.

RFC 2780 B: "IANA Allocation Guidelines For Values In the Internet Protocol and Related Headers" (March 2000)

Abstract: "This memo provides guidance for the IANA to use in assigning parameters for fields in the IPv4, IPv6, ICMP, UDP and TCP protocol headers." [RFC2780]

RFC 4727 S: "Experimental Values" (November 2006)

This document [RFC4727] reserves both TCP options 253 and 254 for experimentation purposes. When such experiments are deployed in the Internet, they should follow the additional requirements in RFC 6994 (see Section 5).

RFC 6335 B: "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry" (August 2011)

From abstract: "This document defines the procedures that the Internet Assigned Numbers Authority (IANA) uses when handling assignment and other requests related to the Service Name and Transport Protocol Port Number registry." [RFC6335]

RFC 6994 S: "Shared Use of Experimental TCP Options (August 2013)

This document [RFC6994] describes how the experimental TCP option code points can concurrently support multiple TCP extensions, even within the same connection. It creates an IANA registry for extensions to the experimental code points.

6. Historic and Undeployed Extensions

The RFCs listed here define extensions that have thus far failed to arouse substantial interest from implementers and have never seen widespread deployment, or were found to be defective for general use. Most of them are reclassified by [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 was not included in the final TCP specification.

RFC 1078 U: "TCP Port Service Multiplexer (TCPMUX)" (November 1988): lack of interest

This document [RFC1078] proposes 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 acknowledgment" or NAK option. There is a comparison of NAK and SACK methods, and early discussion of TCP over satellite issues. RFC 1110 (see Section 6) 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 (see Section 6) 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, Section 6.

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. RFC 1379 [RFC1379] (see Section 6) and RFC 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 (see Section 4.1), 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 specialized transport protocols other than TCP, such as PR-SCTP [RFC3758].

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. Later work on splitting locator and identifier values is summarized well in [RFC6115], but no resulting changes to TCP have occurred.

RFC 6013 E: "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 negotiate 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 a method to extend the option space after the connection establishment.

Although RFC 6013 was published in 2011, the authors of this document places it in this section of the roadmap document due to two factors.

- (a) The authors are 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.

7. 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 7.1 describes several foundational RFCs that give modern readers a better understanding of the principles underlying TCP's behaviors and development over the years. Section 7.2 contains architectural guidelines and principles for TCP architects and designers. The documents listed in Section 7.3 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 7.4. Some implementation notes and implementation advice can be found in Section 7.5. RFCs that describe tools for testing and debugging TCP implementations or that contain high-level tutorials on the protocol are listed in Section 7.6. The TCP Management Information Bases are described in Section 7.7, and Section 7.8 lists a number of case studies that have explored TCP performance.

7.1. 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 fundamental RFC 793 (see Section 2), 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 (see Section 2).

RFC 761 U: "DoD standard Transmission Control Protocol" (January 1980)

This document [RFC0761] is the immediate precursor of RFC 793 (see

Section 2). The header format, the connection establishment including the different connection states, and the overall API correspond mostly to the final Standard RFC 793 (see Section 2).

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. Furthermore, it defined an algorithm for efficient transmission of small packets that is today known as the Nagle Algorithm.

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 (see Section 2). This serves to remind us of the difficulty in specification writing (even when we work from existing documents!).

7.2. 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 designs 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. Later work on TCP has included several more aggressive mechanisms than Reno TCP includes, and RFC 5033 (see Section 7.4) provides additional guidance on use of such algorithms. The fundamental architectural discussion in RFC 2914 remains valid, regarding the standards process role in defining protocol aspects that are critical to performance and avoiding congestion collapse scenarios.

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 3439 I: "Some Internet Architectural Guidelines and Philosophy" (December 2002)

This document [RFC3439] updates RFC 1958 (see Section 7.2) 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 impediment to efficient scaling.

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 with routers that do not understand the defined alternate semantics.

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) (see Section 4.7). This document lists certain high-level design decisions that provide foundations for the design of the MPTCP protocol, based upon these architectural requirements" [RFC6182]

7.3. 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 (i.e., networks with low bandwidth and high delay), 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 (see Section 7.3) and RFC 3155 (see Section 7.3), and these documents should be preferred where there is overlap between them and RFC 2757 (see Section 7.3).

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 (see Section 7.3) 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. RFC 3366 (see Section 7.3) specifically focuses on ARQ mechanisms, while RFC 3819 more widely covers additional aspects of the underlying layers

7.4. 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 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 (see Section 7.2). 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 topics, the document discusses throughput, delay, loss rates, response times, fairness and robustness for challenging environments.

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.

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) (see Section 4.7). 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]

7.5. Implementation Advice

RFC 794 U: "PRE-EMPTION" (September 1981)

This document [RFC0794] clarifies that operating systems need to manage their limited resources, which may include TCP connection state, and that these decisions can be made with application input, but they do not need to be part of the TCP protocol specification itself.

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 purpose 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 (see Section 7.5).

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 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 acknowledgments (ACKs) due to confusion between Maximum Segment Size (MSS) and segment size, and MSS advertisement based on PMTU." [RFC2923]

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 (RFC 7323, see Section 3.1) 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 take 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) (see Section 4.7) 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.

7.6. 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 usable.

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 is on end-host-based congestion control.

7.7. MIB Modules

The first MIB module defined for use with Simple Network Management Protocol (SNMP) was a single monolithic MIB module, called MIB-I, defined in RFC 1156. This evolved over time to the MIB-II specification in RFC 1213, which obsoletes RFC 1156. 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. For current TCP implementers, RFC 4022 should be supported.

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. It is the immediate successor of RFC 1158, with minor modifications. It obsoletes the MIB-I, defined in RFC 1156 (see Section 7.7).

RFC 2012 S: "SNMPv2 Management Information Base for the Transmission Control Protocol using SMIV2" (November 1996)

In an update to RFC 1213 (see Section 7.7), this document [RFC2012] defines the TCP MIB by splitting out the TCP-specific portions. It is now obsoleted by RFC 4022 (see Section 7.7).

RFC 2452 S: "IP Version 6 Management Information Base for the Transmission Control Protocol" (December 1998)

This document [RFC2452] augments RFC 2012 (see Section 7.7) by adding an IPv6-specific connection table. The rest of RFC 2012 holds for any IP version. RFC 2452 is now obsoleted by RFC 4022 (see Section 7.7).

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 (see Section 7.7) and RFC 2452 (see Section 7.7) and specifies the current standard for the TCP MIB that should be deployed.

RFC 4898 S: "TCP Extended Statistics MIB" (May 2007)

This document [RFC4898] describes extended performance statistics for TCP. They are designed to use TCP's ideal vantage point to diagnose performance problems in both the network and the application.

7.8. 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.

8. 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] includes an alternate algorithm for triggering fast retransmit [RFC5681], 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 duplicate ACK 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 duplicate ACK threshold. FACK is implemented in Linux and turned on per default.

Congestion Control for High Rate Flows

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] (see Section 4.3), Limited Slow-Start [RFC3742] (see Section 4.3), and Quick-Start [RFC4782] (see Section 4.3), 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).

9. 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.

10. IANA Considerations

This document contains no IANA considerations.

11. 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 Mark Allman, Yuchung Cheng, Ted Faber, Fairhurst, Sally Floyd, Janardhan Iyengar, Reiner Ludwig, Pekka Savola, and Joe Touch for their contributions, in particular. Keith McCloghrie provided some useful notes and clarification on the various MIB-related RFCs.

12. References

12.1. Normative References

- [I-D.ietf-tcpm-1323bis]
Borman, D., Braden, R., Jacobson, V., and R. Scheffenegger, "TCP Extensions for High Performance", draft-ietf-tcpm-1323bis-21 (work in progress), April 2014.
- [I-D.ietf-tcpm-fastopen]
Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", draft-ietf-tcpm-fastopen-09 (work in progress), July 2014.
- [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.
- [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.
- [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] Rijsinghani, 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] McCloghrie, 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.
- [RFC2780] Bradner, S. and V. Paxson, "IANA Allocation Guidelines For Values In the Internet Protocol and Related Headers", BCP 37, RFC 2780, March 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.
- [RFC4653] Bhandarkar, S., Reddy, A., Allman, M., and E. Blanton, "Improving the Robustness of TCP to Non-Congestion Events", RFC 4653, August 2006.
- [RFC4727] Fenner, B., "Experimental Values In IPv4, IPv6, ICMPv4, ICMPv6, UDP, and TCP Headers", RFC 4727, November 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.
- [RFC4898] Mathis, M., Heffner, J., and R. Raghunarayan, "TCP Extended Statistics MIB", RFC 4898, May 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.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, August 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., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, April 2013.
- [RFC6937] Mathis, M., Dukkipati, N., and Y. Cheng, "Proportional Rate Reduction for TCP", RFC 6937, May 2013.
- [RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, August 2013.

12.2. 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.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [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.
- [RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, May 2004.
- [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.
- [RFC6115] Li, T., "Recommendation for a Routing Architecture", RFC 6115, February 2011.
- [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
F5 Networks
401 Elliott Ave W
Seattle, WA 98119

Phone: 206-272-7537
Email: m.duke@f5.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
Email: elb@psg.com

Alexander Zimmermann
NetApp, Inc.
Sonnenallee 1
Kirchheim 85551
Germany

Phone: +49 89 900594712
Email: alexander.zimmermann@netapp.com

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

Y. Nishida
GE Global Research
October 17, 2015

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

Abstract

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

Status of This Memo

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

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

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

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

Copyright Notice

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

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

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

Table of Contents

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

1. Introduction

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

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

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

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

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

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

2. Conventions and Terminology

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

3. The A-PAWS Design

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

3.1. Signaling Methods

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

1) Option Exchange in SYN

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

2) Option Exchange in non-SYN Segments

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

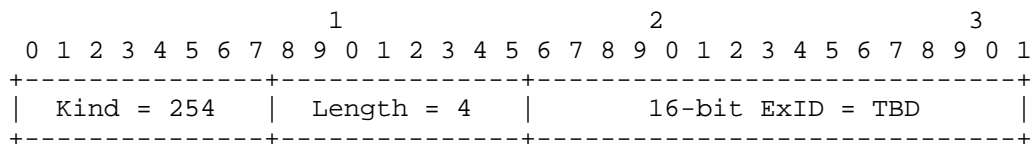


Figure 1: A-PAWS option format

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

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

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

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

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

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

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

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

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

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

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

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

3.3. Sending Behavior

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

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

3.4. Receiving Behavior

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

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

4. When To Activate A-PAWS

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

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

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

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

5. Discussion

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

5.1. Protection Against Early Incarnations

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

5.2. Protection Against Security Threats

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

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

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

5.3. Middlebox Considerations

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

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

5.4. Aggressive Mode in A-PAWS

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

6. Security Considerations

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

7. IANA Considerations

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

8. References

8.1. Normative References

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

8.2. Informative References

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

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

Author's Address

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

Email: nishida@wide.ad.jp