

Internet Engineering Task Force  
Internet-Draft  
Intended status: Standards Track  
Expires: May 21, 2021

G. Fairhurst  
University of Aberdeen  
November 17, 2020

Guidelines for Internet Congestion Control at Endpoints  
draft-fairhurst-tsvwg-cc-05

Abstract

This document is for discussion by the TSVWG. It provides guidance on the design of methods to avoid congestion collapse and to provide congestion control. Recommendations and requirements on this topic are distributed across many documents in the RFC series. This therefore seeks to gather and consolidate these recommendations in an annexe. Based on these specifications, and Internet engineering experience, the document provides input to the design of new congestion control methods in protocols.

The present document is for discussion and comment by the IETF. If published, it plans to update or replace the Best Current Practice in BCP 41, which currently includes "Congestion Control Principles" provided in RFC2914.

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 May 21, 2021.

Copyright Notice

Copyright (c) 2020 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. Terminology . . . . .	3
3. Principles of Congestion Control . . . . .	4
3.1. Internet Requirements . . . . .	4
3.1.1. Tolerance to a Diversity of Path Characteristics . . . . .	5
3.2. Avoiding Congestion Collapse and Flow Starvation . . . . .	5
3.3. Connection Initialization . . . . .	6
3.4. Using Path Capacity . . . . .	7
3.5. Timers and Retransmission . . . . .	8
3.6. Responding to Potential Congestion . . . . .	10
3.7. Using More Capacity . . . . .	11
3.8. Network Signals . . . . .	12
3.9. Protection of Protocol Mechanisms . . . . .	12
4. IETF Guidelines on Evaluation of Congestion Control . . . . .	13
5. Acknowledgements . . . . .	13
6. IANA Considerations . . . . .	13
7. Security Considerations . . . . .	13
8. References . . . . .	14
8.1. Normative References . . . . .	14
8.2. Informative References . . . . .	15
Appendix A. Internet Congestion Control . . . . .	19
A.1. Flow Multiplexing and Congestion . . . . .	20
A.2. Avoiding Congestion Collapse and Flow Starvation . . . . .	22
A.3. Adjusting the Rate . . . . .	22
Appendix B. Best Current Practice in the RFC-Series . . . . .	23
Appendix C. Revision Notes . . . . .	25
Author's Address . . . . .	26

## 1. Introduction

The IETF has specified Internet transports (e.g., TCP [I-D.ietf-tcpm-rfc793bis], UDP [RFC0768], UDP-Lite [RFC3828], SCTP [RFC4960], and DCCP [RFC4340]) as well as protocols layered on top of these transports (e.g., RTP [RFC3550], QUIC [I-D.ietf-quic-transport], SCTP/UDP [RFC6951], DCCP/UDP [RFC6773]) and transports that work directly over the IP network layer. These

transports are implemented in endpoints (either Internet hosts or routers acting as endpoints), and are designed to detect and react to network congestion. TCP was the first transport to provide this, although the TCP specifications found in RFC 793 predates the inclusion of congestion control and did not contain any discussion of using or managing a congestion window. RFC 793.bis [I-D.ietf-tcpm-rfc793bis] seek to address this.

Recommendations and requirements on this topic are distributed across many documents in the RFC series. The appendix of this document therefore seeks to gather and consolidate these recommendations. This, and Internet engineering experience are used as a basis to provide overall guidelines as input to the design of congestion control methods that are implemented in Internet protocols. The focus of the present document is upon unicast point-to-point transports, this includes migration from using one path to another path.

Some recommendations [RFC5783] and requirements in this document apply to point-to-multipoint transports (e.g., multicast), however this topic extends beyond the current document's scope. [RFC2914] provides additional guidance on the use of multicast.

## 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 [RFC2119].

The path between endpoints (sometimes called "Internet Hosts" or called source and destination nodes in IPv6) consists of the endpoint protocol stack at the sender and the receiver (which together implement the transport service), and a succession of links and network devices (routers or middleboxes) that provide connectivity across the network. The set of network devices forming the path is not usually fixed, and it should generally be assumed that this set can change over arbitrary lengths of time.

[RFC5783] defines congestion control as "the feedback-based adjustment of the rate at which data is sent into the network. Congestion control is an indispensable set of principles and mechanisms for maintaining the stability of the Internet." [RFC5783] also provides an informational snapshot taken by the IRTF's Internet Congestion Control Research Group (ICCRG) from October 2008.

The text draws on language used in the specifications of TCP and other IETF transports. For example, a protocol timer is generally needed to detect persistent congestion, and this document uses the

term Retransmission Timeout (RTO) to refer to the operation of this timer. Similarly, the document refers to a congestion window as the variable that controls the rate of transmission by the congestion controller. The use of these terms does not imply that endpoints need to implement functions in the way that TCP currently does. Each new transport needs to make its own design decisions about how to meet the recommendations and requirements for congestion control.

Other terminology is directly copied from the cited RFCs.

### 3. Principles of Congestion Control

This section summarises the principles for providing congestion control.

#### 3.1. Internet Requirements

Principles include:

- o Endpoints **MUST** perform congestion control [RFC1122] and **SHOULD** leverage existing congestion control techniques [RFC8085].
- o If an application or protocol chooses not to use a congestion-controlled transport protocol, it **SHOULD** control the rate at which it sends datagrams to a destination host, in order to fulfil the requirements of [RFC2914], as stated in [RFC8085].
- o Transports **SHOULD** control the aggregate traffic they send on a path [RFC8085]. They ought not to use multiple congestion-controlled flows between the same endpoints to gain a performance advantage. An endpoint can become aware of congestion by various means (including, delay variation, timeout, ECN, packet loss). A signal that indicates congestion on the end-to-end network path, **SHOULD** result in a congestion control reaction by the transport that reduces the current rate of the sending endpoint [RFC8087]).
- o Although network devices can be configured to reduce the impact of flow multiplexing on other flows, endpoints **MUST NOT** rely solely on the presence and correct configuration of these methods, except when constrained to operate in a controlled environment. Transports that do not target Internet deployment need to be constrained to only operate in a controlled environment (e.g., see Section 3.6 of [RFC8085]) and provide appropriate mechanisms to prevent traffic accidentally leaving the controlled environment [RFC8084].

### 3.1.1. Tolerance to a Diversity of Path Characteristics

Principles include:

- o A transport design is REQUIRED be robust to a change in the set of devices forming the network path. A reconfiguration, reset or other event could interrupt this path or trigger a change in the set of network devices forming the path.
- o Transports are REQUIRED to operate safely over the wide range of path characteristics presented by Internet paths.
- o Path characteristics can change over relatively short intervals of time (i.e., characteristics discovered do not necessarily remain valid for multiple Round Trip Times, RTTs). In particular, the transport SHOULD measure and adapt to the characteristics of the path(s) being used.

### 3.2. Avoiding Congestion Collapse and Flow Starvation

Principles include:

- o Transports MUST avoid inducing flow starvation to other flows that share resources along the path they use.
- o Endpoints MUST treat a loss of all feedback (e.g., expiry of a retransmission time out, RTO) as an indication of persistent congestion (i.e., an indication of potential congestion collapse).
- o When an endpoint detects persistent congestion, it MUST reduce the maximum rate (e.g., reduce its congestion window). This normally involves the use of protocol timers to detect a lack of acknowledgment for transmitted data (Section 3.5).
- o Protocol timers (e.g., used for retransmission or to detect persistent congestion) need to be appropriately initialised. A transport SHOULD adapt its protocol timers to follow the measured the path Round Trip Time (RTT) (e.g., Section 3.1.1 of [RFC8085]).
- o A transport MUST employ exponential backoff each time persistent congestion is detected [RFC1122], until the path characteristics can again be confirmed.
- o Network devices MAY provide mechanisms to mitigate the impact of congestion collapse by transport flows (e.g., priority forwarding of control information, and starvation detection), and SHOULD mitigate the impact of non-conformant and malicious flows

[RFC7567])). These mechanisms complement, but do not replace, the endpoint congestion avoidance mechanisms.

### 3.3. Connection Initialization

When a connection or flow to a new destination is established, the endpoints have little information about the characteristics of the network path they will use. This section describes how a flow starts transmission over such a path.

**Flow Start:** A new flow between two endpoints needs to initialise a congestion controller for the path it will use. It **MUST NOT** assume that capacity is available at the start of the flow, unless it uses a mechanism to explicitly reserve capacity. In the absence of a capacity signal, a flow might therefore start slowly. The TCP slow-start algorithm is an accepted standard for flow startup [RFC5681]. TCP uses the notion of an Initial Window (IW) [RFC3390], updated by [RFC6928] to define the initial volume of data that can be sent on a path. This is not the smallest burst, or the smallest window, but it is considered a safe starting point for a path that is not suffering persistent congestion, and is applicable until feedback about the path is received. The initial sending rate (e.g., determined by the IW) needs to be viewed as tentative until the capacity is confirmed to be available.

**Initial RTO Interval:** When a flow sends the first packet(s), it typically has no way to know the actual RTT of the path it will use. An initial value needs to be used to initialise the principal retransmission timer, which will be used to detect lack of responsiveness from the remote endpoint. In TCP, this is the starting value of the RTO. The selection of a safe initial value is a trade off that has important consequences on the overall Internet stability [RFC6928] [RFC8085]. In the absence of any knowledge about the latency of a path (including the initial value), the RTO **MUST** be conservatively set to no less than 1 second. Values shorter than 1 second can be problematic (see the appendix of [RFC6298]). (Note: Linux TCP has deployed a smaller initial RTO value).

[[Author note: It could be useful to discuss cached values]].

**Initial RTO Expiry:** If the RTO timer expires while awaiting completion of a connection setup, or handshake (e.g., the three-way handshake in TCP, the ACK of a SYN segment), and the implementation is using an RTO less than 3 seconds, the local endpoint can resend the connection setup. [[Author note: It would be useful to discuss how the timer is managed to protect from multiple handshake failure]].

The RTO MUST then be re-initialized to increase it to 3 seconds when data transmission begins (i.e., after the handshake completes) [RFC6298] [RFC8085]. This conservative increase is necessary to avoid congestion collapse when many flows retransmit across a shared bottleneck with restricted capacity.

**Initial Measured RTO:** Once an RTT measurement is available (e.g., through reception of an acknowledgement), the timeout value must be adjusted. This adjustment MUST take into account the RTT variance. For the first sample, this variance cannot be determined, and a local endpoint MUST therefore initialise the variance to  $RTT/2$  (see equation 2.2 of [RFC6928] and related text for UDP in section 3.1.1 of [RFC8085]).

**Current State:** A congestion controller MAY assume that recently used capacity between a pair of endpoints is an indication of future capacity available in the next RTT between the same endpoints. It MUST react (reduce its rate) if this is not (later) confirmed to be true. [[Author note: do we need to bound this]].

**Cached State:** A congestion controller that recently used a specific path could use additional state that lets a flow take-over the capacity that was previously consumed by another flow (e.g., in the last RTT) which it understands is using the same path and no will longer use the capacity it recently used. In TCP, this mechanism is referred to as TCP Control Block (TCB) sharing [RFC2140] [I-D.ietf-tcpm-2140bis]. The capacity and other information can be used to suggest a faster initial sending rate.

Any information used to accelerate the growth of the cwnd MUST be viewed as tentative until the path capacity is confirmed by receiving a confirmation that actual traffic has been sent across the path. (i.e., the new flow needs to either use or loose the capacity that has been tentatively offered to it). A sender MUST reduce its rate if this capacity is not confirmed within the current RTO interval.

### 3.4. Using Path Capacity

This section describes how a sender needs to regulate the maximum volume of data in flight over the interval of the current RTT, and how it manages transmission of the capacity that it perceives is available.

**Transient Path:** Unless managed by a resource reservation protocol, path capacity information is transient. A sender that does not use capacity has no understanding whether previously used capacity remains available to use, or whether that capacity has disappeared

(e.g., a change in the path that causes a flow to experience a smaller bottleneck, or when more traffic emerges that consumes previously available capacity resulting in a new bottleneck). For this reason, a transport that is limited by the volume of data available to send MUST NOT continue to grow its congestion window when the current congestion window is more than twice the volume of data acknowledged in the last RTT.

**Validating the congestion window** Standard TCP states that a TCP sender "SHOULD set the congestion window to no more than the Restart Window (R)" before beginning transmission, if the sender has not sent data in an interval that exceeds the current retransmission timeout, i.e., when an application becomes idle [RFC5681]. An experimental specification [RFC7661] permits TCP senders to tentatively maintain a congestion window larger than the path supported in the last RTT when application-limited, provided that they appropriately and rapidly collapse the congestion window when potential congestion is detected. This mechanism is called Congestion Window Validation (CWV).

**Collateral Damage:** Even in the absence of congestion, statistical multiplexing of flows can result in transient effects for flows sharing common resources. A sender therefore SHOULD avoid inducing excessive congestion to other flows (collateral damage).

**Burst Mitigation:** While a congestion controller ought to limit sending at the granularity of the current RTT, this can be insufficient to satisfy the goals of preventing starvation and mitigating collateral damage. This requires moderating the burst rate of the sender to avoid significant periods where a flow(s) consume all buffer capacity at the path bottleneck, which would otherwise prevent other flows from gaining a reasonable share. Endpoints SHOULD provide mechanisms to regulate the bursts of transmission that the application/protocol sends to the network (section 3.1.6 of [RFC8085]). ACK-Clocking [RFC5681] can help mitigate bursts for protocols that receive continuous feedback of reception (such as TCP). Sender pacing can mitigate this [RFC8085], (See Section 4.6 of [RFC3449]), and has been recommended for TCP in conditions where ACK-Clocking is not effective, (e.g., [RFC3742], [RFC7661]). SCTP [RFC4960] defines a maximum burst length (Max.Burst) with a recommended value of 4 segments to limit the SCTP burst size.

### 3.5. Timers and Retransmission

This section describes mechanisms to detect and provide retransmission, and to protect the network in the absence of timely feedback.



**Loss Detection:** Loss detection occurs after a sender determines there is no delivery confirmation within an expected period of time (e.g., by observing the time-ordering of the reception of ACKs, as in TCP DupACK) or by utilising a timer to detect loss (e.g., a transmission timer with a period less than the RTO, [RFC8085] [I-D.ietf-tcpm-rack]) or a combination of using a timer and ordering information to trigger retransmission of data.

**Retransmission:** Retransmission of lost packets or messages is a common reliability mechanism. When loss is detected, the sender can choose to retransmit the lost data, ignore the loss, or send other data (e.g., [RFC8085] [I-D.ietf-quic-recovery]), depending on the reliability model provided by the transport service. Any transmission consumes network capacity, therefore retransmissions MUST NOT increase the network load in response to congestion loss (which worsens that congestion) [RFC8085]. Any method that sends additional data following loss is therefore responsible for congestion control of the retransmissions (and any other packets sent, including FEC information) as well as the original traffic.

**Measuring the RTT:** Once an endpoint has started communicating with its peer, the RTT be MUST adjusted by measuring the actual path RTT. This adjustment MUST include adapting to the measured RTT variance (see equation 2.3 of [RFC6928]).

**Maintaining the RTO:** The RTO SHOULD be set based on recent RTT observations (including the RTT variance) [RFC8085].

**RTO Expiry:** Persistent lack of feedback (e.g., detected by an RTO timer, or other means) MUST be treated an indication of potential congestion collapse. A failure to receive any specific response within a RTO interval could potentially be a result of a RTT change, change of path, excessive loss, or even congestion collapse. If there is no response within the RTO interval, TCP collapses the congestion window to one segment [RFC5681]. Other transports MUST similarly respond when they detect loss of feedback.

An endpoint needs to exponentially backoff the RTO interval [RFC8085] each time the RTO expires. That is, the RTO interval MUST be set to at least the  $RTO * 2$  [RFC6298] [RFC8085].

**Maximum RTO:** A maximum value MAY be placed on the RTO interval. This maximum limit to the RTO interval MUST NOT be less than 60 seconds [RFC6298].

[[ Author Note: Check RTO-Consider. ]]

### 3.6. Responding to Potential Congestion

The safety and responsiveness of new proposals need to be evaluated [RFC5166]. In determining an appropriate congestion response, designs could take into consideration the size of the packets that experience congestion [RFC4828].

**Congestion Response:** An endpoint MUST promptly reduce the rate of transmission when it receive or detects an indication of congestion (e.g., loss) [RFC2914].

TCP Reno established a method that relies on multiplicative-decrease to halve the sending rate while congestion is detected. This response to congestion indications is considered sufficient for safe Internet operation, but other decrease factors have also been published in the RFC Series [RFC8312].

**ECN Response:** A congestion control design should provide the necessary mechanisms to support Explicit Congestion Notification (ECN) [RFC3168] [RFC6679], as described in section 3.1.7 of [RFC8085]. This can help determine an appropriate congestion window when supported by routers on the path [RFC7567] to enable rapid early indication of incipient congestion.

The early detection of incipient congestion justifies a different reaction to an explicit congestion signal compared to the reaction to detected packet loss [RFC8311] [RFC8087]. Simple feedback of received Congestion Experienced (CE) marks [RFC3168], relies only on an indication that congestion has been experienced within the last RTT. This style of response is appropriate when a flow uses ECT(0). The reaction to reception of this indication was modified in TCP ABE [RFC8511]. Further detail about the received CE-marking can be obtained by using more accurate receiver feedback (e.g., [I-D.ietf-tcpm-accurate-ecn] and extended RTP feedback). The more detailed feedback provides an opportunity for a finer-granularity of congestion response.

Current work-in-progress [I-D.ietf-tsvwg-l4s-arch] defines a reaction for packets marked with ECT(1), building on the style of detailed feedback provided by [I-D.ietf-tcpm-accurate-ecn] and a modified marking system [I-D.ietf-tsvwg-aqm-dualq-coupled].

**Robustness to Path Change:** The detection of congestion and the resulting reduction MUST NOT solely depend upon reception of a signal from the remote endpoint, because congestion indications could themselves be lost under persistent congestion.

The only way to surely confirm that a sending endpoint has successfully communicated with a remote endpoint is to utilise a timer (seeSection 3.5) to detect a lack of response that could result from a change in the path or the path characteristics (usually called the RTO). Congestion controllers that are unable to react after one (or at most a few) RTTs after receiving a congestion indication should observe the guidance in section 3.3 of the UDP Guidelines [RFC8085].

**Persistent Congestion:** Persistent congestion can result in congestion collapse, which **MUST** be aggressively avoided [RFC2914]. Endpoints that experience persistent congestion and have already exponentially reduced their congestion window to the restart window (e.g., one packet), **MUST** further reduce the rate if the RTO timer continues to expire. For example, TFRC [RFC5348] continues to reduce its sending rate under persistent congestion to one packet per RT, and then exponentially backs off the time between single packet transmissions if the congestion continues to persist [RFC2914].

[RFC8085] provides guidelines for a sender that does not, or is unable to, adapt the congestion window.

### 3.7. Using More Capacity

In the absence of persistent congestion, an endpoint **MAY** increase its congestion window and hence the sending rate. An increase should only occur when there is additional data available to send across the path (i.e., the sender will utilise the additional capacity in the next RTT).

**Increasing Congestion Window:** A sender **MUST NOT** continue to increase its rate for more than an RTT after a congestion indication is received. The transport **SHOULD** stop increasing its congestion window as soon as it receives indication of congestion.

While the sender is increasing the congestion window, a sender will transmit faster than the last confirmed safe rate. Any increase above the last confirmed rate needs to be regarded as tentative and the sender reduce their rate below the last confirmed safe rate when congestion is experienced (a congestion event).

**Congestion:** An endpoint **MUST** utilise a method that assures the sender will keep the rate below the previously confirmed safe rate for multiple RTT periods after an observed congestion event. In TCP, this is performed by using a linear increase from a slow

start threshold that is re-initialised when congestion is experienced.

**Avoiding Overshoot:** Overshoot of the congestion window beyond the point of congestion can significantly impact other flows sharing resources along a path. It is important to note that as endpoints experience more paths with a large BDP and a wider range of potential path RTT, that variability or changes in the path can have very significant constraints on appropriate dynamics for increasing the congestion window (see also burst mitigation, Section 3.4).

### 3.8. Network Signals

An endpoint can utilise signals from the network to help determine how to regulate the traffic it sends.

**Network Signals:** Mechanisms **MUST NOT** solely rely on transport messages or specific signalling messages to perform safely. (See section 5.2 of [RFC8085] describing use of ICMP messages). They need to be designed so that they safely operate when path characteristics change at any time. Transport mechanisms **MUST** be robust to potential black-holing of any signals (i.e., need to be robust to loss or modification of packets, noting that this can occur even after successful first use of a signal by a flow, as occurs when the path changes, see Section 3.1.1).

A mechanism that utilises signals originating in the network (e.g., RSVP, NSIS, Quick-Start, ECN), **MUST** assume that the set of network devices on the path can change. This motivates the use of soft-state when designing protocols that interact with signals originating from network devices [I-D.irtf-panrg-what-not-to-do] (e.g., ECN). This can include context-sensitive treatment of "soft" signals provided to the endpoint [RFC5164].

### 3.9. Protection of Protocol Mechanisms

An endpoint needs to provide protection from attacks on the traffic it generates, or attacks that seek to increase the capacity it consumes (impacting other traffic that shared a bottleneck).

**Off Path Attack:** A design **MUST** protect from off-path attack to the protocol [RFC8085] (i.e., one by an attacker that is unable to see the contents of packets exchanged across the path). An attack on the congestion control can lead to a Denial of Service (DoS) vulnerability for the flow being controlled and/or other flows that share network resources along the path.

Validation of Signals: Network signalling and control messages (e.g., ICMP [RFC0792]) MUST be validated before they are used to protect from malicious abuse. This MUST at least include protection from off-path attack [RFC8085].

On Path Attack: A protocol can be designed to protect from on-path attacks, but this requires more complexity and the use of encryption/authentication mechanisms (e.g., IPsec [RFC4301], QUIC [I-D.ietf-quic-transport]).

#### 4. IETF Guidelines on Evaluation of Congestion Control

The IETF has provided guidance [RFC5033] for considering alternate congestion control algorithms.

The IRTF has also described a set of metrics and related trade-off between metrics that can be used to compare, contrast, and evaluate congestion control techniques [RFC5166]. [RFC5783] provides a snapshot of congestion-control research in 2008.

#### 5. Acknowledgements

This document owes much to the insight offered by Sally Floyd, both at the time of writing of RFC2914 and her help and review in the many years that followed this.

Nicholas Kuhn helped develop the first draft of these guidelines. Tom Jones and Ana Custura reviewed the first version of this draft. The University of Aberdeen received funding to support this work from the European Space Agency.

#### 6. IANA Considerations

This memo includes no request to IANA.

RFC Editor Note: If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

#### 7. Security Considerations

This document introduces no new security considerations. Each RFC listed in this document discusses the security considerations of the specification it contains. The security considerations for the use of transports are provided in the references section of the cited RFCs. Security guidance for applications using UDP is provided in the UDP Usage Guidelines [RFC8085].

Section 3.9 describes general requirements relating to the design of safe protocols and their protection from on and off path attack.

Section 3.8 follows current best practice to validate ICMP messages prior to use.

## 8. References

### 8.1. Normative References

- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://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, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, DOI 10.17487/RFC2914, September 2000, <<https://www.rfc-editor.org/info/rfc2914>>.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, DOI 10.17487/RFC3390, October 2002, <<https://www.rfc-editor.org/info/rfc3390>>.
- [RFC3742] Floyd, S., "Limited Slow-Start for TCP with Large Congestion Windows", RFC 3742, DOI 10.17487/RFC3742, March 2004, <<https://www.rfc-editor.org/info/rfc3742>>.
- [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <<https://www.rfc-editor.org/info/rfc5348>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/info/rfc5681>>.

- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <<https://www.rfc-editor.org/info/rfc6298>>.
- [RFC6928] Chu, J., Dukkupati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <<https://www.rfc-editor.org/info/rfc6928>>.
- [RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <<https://www.rfc-editor.org/info/rfc7567>>.
- [RFC7661] Fairhurst, G., Sathiaselalan, A., and R. Secchi, "Updating TCP to Support Rate-Limited Traffic", RFC 7661, DOI 10.17487/RFC7661, October 2015, <<https://www.rfc-editor.org/info/rfc7661>>.
- [RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <<https://www.rfc-editor.org/info/rfc8085>>.

## 8.2. Informative References

- [Flow-Rate-Fairness]  
Briscoe, Bob., "Flow Rate Fairness: Dismantling a Religion", ACM Computer Communication Review 37(2):63-74", April 2007.
- [I-D.ietf-quic-recovery]  
Iyengar, J. and I. Swett, "QUIC Loss Detection and Congestion Control", draft-ietf-quic-recovery-32 (work in progress), October 2020.
- [I-D.ietf-quic-transport]  
Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-32 (work in progress), October 2020.
- [I-D.ietf-tcpm-2140bis]  
Touch, J., Welzl, M., and S. Islam, "TCP Control Block Interdependence", draft-ietf-tcpm-2140bis-05 (work in progress), April 2020.

- [I-D.ietf-tcpm-accurate-ecn]  
Briscoe, B., Kuehlewind, M., and R. Scheffenegger, "More Accurate ECN Feedback in TCP", draft-ietf-tcpm-accurate-ecn-13 (work in progress), November 2020.
- [I-D.ietf-tcpm-rack]  
Cheng, Y., Cardwell, N., Dukkkipati, N., and P. Jha, "The RACK-TLP loss detection algorithm for TCP", draft-ietf-tcpm-rack-13 (work in progress), November 2020.
- [I-D.ietf-tcpm-rfc793bis]  
Eddy, W., "Transmission Control Protocol (TCP) Specification", draft-ietf-tcpm-rfc793bis-19 (work in progress), October 2020.
- [I-D.ietf-tsvwg-aqm-dualq-coupled]  
Schepper, K., Briscoe, B., and G. White, "DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)", draft-ietf-tsvwg-aqm-dualq-coupled-13 (work in progress), November 2020.
- [I-D.ietf-tsvwg-l4s-arch]  
Briscoe, B., Schepper, K., Bagnulo, M., and G. White, "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture", draft-ietf-tsvwg-l4s-arch-08 (work in progress), November 2020.
- [I-D.irtf-panrg-what-not-to-do]  
Dawkins, S., "Path Aware Networking: Obstacles to Deployment (A Bestiary of Roads Not Taken)", draft-irtf-panrg-what-not-to-do-14 (work in progress), November 2020.
- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <<https://www.rfc-editor.org/info/rfc768>>.
- [RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, DOI 10.17487/RFC0792, September 1981, <<https://www.rfc-editor.org/info/rfc792>>.
- [RFC0896] Nagle, J., "Congestion Control in IP/TCP Internetworks", RFC 896, DOI 10.17487/RFC0896, January 1984, <<https://www.rfc-editor.org/info/rfc896>>.
- [RFC0970] Nagle, J., "On Packet Switches With Infinite Storage", RFC 970, DOI 10.17487/RFC0970, December 1985, <<https://www.rfc-editor.org/info/rfc970>>.



- [RFC2140] Touch, J., "TCP Control Block Interdependence", RFC 2140, DOI 10.17487/RFC2140, April 1997, <<https://www.rfc-editor.org/info/rfc2140>>.
- [RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, DOI 10.17487/RFC2309, April 1998, <<https://www.rfc-editor.org/info/rfc2309>>.
- [RFC2525] Paxson, V., Allman, M., Dawson, S., Fenner, W., Griner, J., Heavens, I., Lahey, K., Semke, J., and B. Volz, "Known TCP Implementation Problems", RFC 2525, DOI 10.17487/RFC2525, March 1999, <<https://www.rfc-editor.org/info/rfc2525>>.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <<https://www.rfc-editor.org/info/rfc2616>>.
- [RFC3449] Balakrishnan, H., Padmanabhan, V., Fairhurst, G., and M. Sooriyabandara, "TCP Performance Implications of Network Path Asymmetry", BCP 69, RFC 3449, DOI 10.17487/RFC3449, December 2002, <<https://www.rfc-editor.org/info/rfc3449>>.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <<https://www.rfc-editor.org/info/rfc3550>>.
- [RFC3819] Karn, P., Ed., 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, DOI 10.17487/RFC3819, July 2004, <<https://www.rfc-editor.org/info/rfc3819>>.
- [RFC3828] Larzon, L-A., Degermark, M., Pink, S., Jonsson, L-E., Ed., and G. Fairhurst, Ed., "The Lightweight User Datagram Protocol (UDP-Lite)", RFC 3828, DOI 10.17487/RFC3828, July 2004, <<https://www.rfc-editor.org/info/rfc3828>>.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<https://www.rfc-editor.org/info/rfc4301>>.

- [RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI 10.17487/RFC4340, March 2006, <<https://www.rfc-editor.org/info/rfc4340>>.
- [RFC4828] Floyd, S. and E. Kohler, "TCP Friendly Rate Control (TFRC): The Small-Packet (SP) Variant", RFC 4828, DOI 10.17487/RFC4828, April 2007, <<https://www.rfc-editor.org/info/rfc4828>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<https://www.rfc-editor.org/info/rfc4960>>.
- [RFC5033] Floyd, S. and M. Allman, "Specifying New Congestion Control Algorithms", BCP 133, RFC 5033, DOI 10.17487/RFC5033, August 2007, <<https://www.rfc-editor.org/info/rfc5033>>.
- [RFC5164] Melia, T., Ed., "Mobility Services Transport: Problem Statement", RFC 5164, DOI 10.17487/RFC5164, March 2008, <<https://www.rfc-editor.org/info/rfc5164>>.
- [RFC5166] Floyd, S., Ed., "Metrics for the Evaluation of Congestion Control Mechanisms", RFC 5166, DOI 10.17487/RFC5166, March 2008, <<https://www.rfc-editor.org/info/rfc5166>>.
- [RFC5783] Welzl, M. and W. Eddy, "Congestion Control in the RFC Series", RFC 5783, DOI 10.17487/RFC5783, February 2010, <<https://www.rfc-editor.org/info/rfc5783>>.
- [RFC6363] Watson, M., Begen, A., and V. Roca, "Forward Error Correction (FEC) Framework", RFC 6363, DOI 10.17487/RFC6363, October 2011, <<https://www.rfc-editor.org/info/rfc6363>>.
- [RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", RFC 6679, DOI 10.17487/RFC6679, August 2012, <<https://www.rfc-editor.org/info/rfc6679>>.
- [RFC6773] Phelan, T., Fairhurst, G., and C. Perkins, "DCCP-UDP: A Datagram Congestion Control Protocol UDP Encapsulation for NAT Traversal", RFC 6773, DOI 10.17487/RFC6773, November 2012, <<https://www.rfc-editor.org/info/rfc6773>>.

- [RFC6951] Tuexen, M. and R. Stewart, "UDP Encapsulation of Stream Control Transmission Protocol (SCTP) Packets for End-Host to End-Host Communication", RFC 6951, DOI 10.17487/RFC6951, May 2013, <<https://www.rfc-editor.org/info/rfc6951>>.
- [RFC8084] Fairhurst, G., "Network Transport Circuit Breakers", BCP 208, RFC 8084, DOI 10.17487/RFC8084, March 2017, <<https://www.rfc-editor.org/info/rfc8084>>.
- [RFC8087] Fairhurst, G. and M. Welzl, "The Benefits of Using Explicit Congestion Notification (ECN)", RFC 8087, DOI 10.17487/RFC8087, March 2017, <<https://www.rfc-editor.org/info/rfc8087>>.
- [RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <<https://www.rfc-editor.org/info/rfc8311>>.
- [RFC8312] Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", RFC 8312, DOI 10.17487/RFC8312, February 2018, <<https://www.rfc-editor.org/info/rfc8312>>.
- [RFC8511] Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst, "TCP Alternative Backoff with ECN (ABE)", RFC 8511, DOI 10.17487/RFC8511, December 2018, <<https://www.rfc-editor.org/info/rfc8511>>.

#### Appendix A. Internet Congestion Control

Internet transports can reserve capacity at routers or on the links being used. This is sometimes used in controlled environments, but most uses across the Internet do not rely upon prior reservation of capacity along the path they use. In the absence of such a reservation, endpoints are unable to determine a safe rate at which to start or continue their transmission. The use of an Internet path therefore requires a combination of end-to-end transport mechanisms to detect and then respond to changes in the capacity that it discovers is available across the network path.

Buffering (an increase in latency) or congestion loss (discard of a packet) arises when the traffic arriving at a link or network exceeds the resources available. Loss can also occur for other reasons, but it is usually not possible for an endpoint to reliably disambiguate the cause of packet loss (e.g., loss could be due to link corruption, receiver overrun, etc. [RFC3819]). A network device typically uses

a drop-tail policy to drop excess IP packets when its queue(s) becomes full. This use of buffers can also be managed using Active Queue Management (AQM) [RFC7567], which can be combined withb Explicit Congestion Notification signalling.

Internet transports need to react to avoid congestion that impacts other flows sharing a path. The Requirements for Internet Hosts [RFC1122] formally mandates that endpoints perform congestion control. "Because congestion control is critical to the stable operation of the Internet, applications and other protocols that choose to use UDP as an Internet transport must employ mechanisms to prevent congestion collapse and to establish some degree of fairness with concurrent traffic [RFC2914].

The general recommendation in the UDP Guidelines [RFC8085] is that applications SHOULD leverage existing congestion control techniques, such as those defined for TCP [RFC5681], TCP-Friendly Rate Control (TFRC) [RFC5348], SCTP [RFC4960], and other IETF-defined transports. This is because there are many trade offs and details that can have a serious impact on the performance of congestion control for the application they support and other traffic that seeks to share the resources along the path over which they communicate.

Network devices can be configured to isolate the queuing of packets for different flows, or aggregates of flows, and thereby assist in reducing the impact of flow multiplexing on other flows. This could include methods seeking to equally distribute resources between sharing flows, but this is explicitly not a requirement for a network device [Flow-Rate-Fairness]. Endpoints can not rely on the presence and correct configuration of these methods, and therefore even when a path is expected to support such methods, also need to employ methods that work end-to-end.

Experience has shown that successful protocols developed in a specific context or for a particular application tend to also become used in a wider range of contexts. Therefore, IETF specifications by default target deployment on the general Internet, or need to be defined for use only within a controlled environment.

#### A.1. Flow Multiplexing and Congestion

When a transport uses a path to send packets (i.e. a flow), this impacts any other Internet flows (possibly from or to other endpoints) that share the capacity of any common network device or link (i.e., are multiplexed) along the path. As with loss, latency can also be incurred for other reasons [RFC3819] (Quality of Service link scheduling, link radio resource management/bandwidth on demand,

transient outages, link retransmission, and connection/resource setup below the IP layer, etc).

When choosing an appropriate sending rate, packet loss needs to be considered. Although losses are not always due to congestion, endpoint congestion control needs to conservatively react to loss as a potential signal of reduced available capacity and reduce the sending rate. Many designs place the responsibility of rate-adaption at the sender (source) endpoint, utilising feedback information provided by the remote endpoint (receiver). Congestion control can also be implemented by determining an appropriate rate limit at the receiver and using this limit to control the maximum transport rate (e.g., using methods such as [RFC5348] and [RFC4828]).

It is normal to observe some perturbation in latency and/or loss when flows shares a common network bottleneck with other traffic. This impact needs to be considered and Internet flows ought to implement appropriate safeguards to avoid inappropriate impact on other flows that share the resources along a path. Congestion control methods satisfy this requirement and therefore can help avoid congestion collapse.

"This raises the issue of the appropriate granularity of a "flow", where we define a 'flow' as the level of granularity appropriate for the application of both fairness and congestion control. [RFC2309] states: "There are a few 'natural' answers: 1) a TCP or UDP connection (source address/port, destination address/port); 2) a source/destination host pair; 3) a given source host or a given destination host. We would guess that the source/destination host pair gives the most appropriate granularity in many circumstances. The granularity of flows for congestion management is, at least in part, a policy question that needs to be addressed in the wider IETF community." [RFC2914]

Endpoints can send more than one flow. "The specific issue of a browser opening multiple connections to the same destination has been addressed by [RFC2616]. Section 8.1.4 states that "Clients that use persistent connections SHOULD limit the number of simultaneous connections that they maintain to a given server. A single-user client SHOULD NOT maintain more than 2 connections with any server or proxy." [RFC2140].

This suggests that there are opportunities for transport connections between the same endpoints (from the same or differing applications) might share some information, including their congestion control state, if they are known to share the same path. [RFC8085] adds "An application that forks multiple worker processes or otherwise uses

multiple sockets to generate UDP datagrams SHOULD perform congestion control over the aggregate traffic."

In the absence of persistent congestion, an endpoint is permitted to increase its congestion window and hence the sending rate. An increase should only occur when there is additional data available to send across the path (i.e., the sender will utilise the additional capacity in the next RTT).

TCP Reno [RFC5681] defines an algorithm, known as the Additive-Increase/ Multiplicative-Decrease (AIMD) algorithm, which allows a sender to exponentially increase the congestion window each RTT from the initial window to the first detected congestion event. This is designed to allow new flows to rapidly acquire a suitable congestion window. Where the bandwidth delay product (BDP) is large, it can take many RTT periods to determine a suitable share of the path capacity. Such high BDP paths benefit from methods that more rapidly increase the congestion window, but in compensation these need to be designed to also react rapidly to any detected congestion (e.g., TCP Cubic [RFC8312]).

#### A.2. Avoiding Congestion Collapse and Flow Starvation

A significant pathology can arise when a poorly designed transport creates congestion. This can result in severe service degradation or "Internet meltdown". This phenomenon was first observed during the early growth phase of the Internet in the mid 1980s [RFC0896] [RFC0970]. It is technically called "Congestion Collapse". [RFC2914] notes that informally, "congestion collapse occurs when an increase in the network load results in a decrease in the useful work done by the network."

Transports need to be specifically designed with measures to avoid starving other flows of capacity (e.g., [RFC7567]). [RFC2309] also discussed the dangers of congestion-unresponsive flows, and states that "all UDP-based streaming applications should incorporate effective congestion avoidance mechanisms." [RFC7567] and [RFC8085] both reaffirm this, encouraging development of methods to prevent starvation.

#### A.3. Adjusting the Rate

Congestion Management: The capacity available to a flow could be expressed as the number of bytes in flight, the sending rate or a limit on the number of unacknowledged segments. When determining the capacity used, all data sent by a sender needs to be accounted, this includes any additional overhead or data generated by the transport. A transport performing congestion management

will usually optimise performance for its application by avoiding excessive loss or delay and maintain a congestion window. In steady-state this congestion window reflects a safe limit to the sending rate that has not resulted in persistent congestion. A congestion controller for a flow that uses packet Forward Error Correction (FEC) encoding (e.g., [RFC6363]) needs to consider all additional overhead introduced by packet FEC when setting and managing its congestion window.

One common model views the path between two endpoints as a "pipe". New packets enter the pipe at the sending endpoint, older ones leave the pipe at the receiving endpoint. Congestion and other forms of loss result in "leakage" from this pipe. Received data (leaving the network path at the remote endpoint) is usually acknowledged to the congestion controller.

The rate that data leaves the pipe indicates the share of the capacity that has been utilised by the flow. If, on average (over an RTT), the sending rate equals the receiving rate, this indicates the path capacity. This capacity can be safely used again in the next RTT. If the average receiving rate is less than the sending rate, then the path is either queuing packets, the RTT/path has changed, or there is packet loss.

#### Appendix B. Best Current Practice in the RFC-Series

Like RFC2119, this document borrows heavily from earlier publications addressing the need for end-to-end congestion control, and this subsection provides an overview of key topics.

[RFC2914] provides a general discussion of the principles of congestion control. Section 3 discussed Fairness, stating "The equitable sharing of bandwidth among flows depends on the fact that all flows are running compatible congestion control algorithms". Section 3.1 describes preventing congestion collapse.

Congestion collapse was first reported in the mid 1980s [RFC0896], and at that time was largely due to TCP connections unnecessarily retransmitting packets that were either in transit or had already been received at the receiver. We call the congestion collapse that results from the unnecessary retransmission of packets classical congestion collapse. Classical congestion collapse is a stable condition that can result in throughput that is a small fraction of normal [RFC0896]. Problems with classical congestion collapse have generally been corrected by improvements to timer and congestion control mechanisms, implemented in modern implementations of TCP [Jacobson88]. This classical congestion collapse was a key focus of [RFC2309].

A second form of congestion collapse occurs due to undelivered packets, where Section 5 of [RFC2914] notes: "Congestion collapse from undelivered packets arises when bandwidth is wasted by delivering packets through the network that are dropped before reaching their ultimate destination. This is probably the largest unresolved danger with respect to congestion collapse in the Internet today. Different scenarios can result in different degrees of congestion collapse, in terms of the fraction of the congested links' bandwidth used for productive work. The danger of congestion collapse from undelivered packets is due primarily to the increasing deployment of open-loop applications not using end-to-end congestion control. Even more destructive would be best-effort applications that \*increase\* their sending rate in response to an increased packet drop rate (e.g., automatically using an increased level of FEC (Forward Error Correction))."

Section 3.3 of [RFC2914] notes: "In addition to the prevention of congestion collapse and concerns about fairness, a third reason for a flow to use end-to-end congestion control can be to optimize its own performance regarding throughput, delay, and loss. In some circumstances, for example in environments with high statistical multiplexing, the delay and loss rate experienced by a flow are largely independent of its own sending rate. However, in environments with lower levels of statistical multiplexing or with per-flow scheduling, the delay and loss rate experienced by a flow is in part a function of the flow's own sending rate. Thus, a flow can use end-to-end congestion control to limit the delay or loss experienced by its own packets. We would note, however, that in an environment like the current best-effort Internet, concerns regarding congestion collapse and fairness with competing flows limit the range of congestion control behaviors available to a flow."

In addition to the prevention of congestion collapse and concerns about fairness, a flow using end-to-end congestion control can optimize its own performance regarding throughput, delay, and loss [RFC2914].

The standardization of congestion control in new transports can avoid a congestion control "arms race" among competing protocols [RFC2914]. That is, avoid designs of transports that could compete for Internet resource in a way that significantly reduces the ability of other flows to use the Internet. The use of standard methods is therefore encouraged.

The popularity of the Internet has led to a proliferation in the number of TCP implementations [RFC2914]. A variety of non-TCP transports have also been deployed. Some transport implementations fail to use standardised congestion avoidance mechanisms correctly



because of poor implementation [RFC2525]. However, this is not the only reason for not using standard methods. Some transports have chosen mechanisms that are not presently standardised, or have adopted approaches to their design that differ from present standards. Guidance is needed therefore not only for future standardisation, but to ensure safe and appropriate evolution of transports that have not presently been submitted for standardisation.

#### Appendix C. Revision Notes

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

Individual draft -00:

- o Comments and corrections are welcome directly to the authors or via the IETF TSVWG, working group mailing list.

Individual draft -01:

- o This update is proposed for initial WG comments.
- o If there is interest in progressing this document, the next version will include more complete referencing to cited material.

Individual draft -02:

- o Correction of typos.

Individual draft -03:

- o Added section 1.1 with text on current BCP status with additional alignment and updates to RFC2914 on Congestion Control Principles (after question from M. Scharf).
- o Edits to consolidate starvation text.
- o Added text that multicast currently noting that this is out of scope.
- o Revised sender-based CC text after comment from C. Perkins (Section 3.1, 3.3 and other places).
- o Added more about FEC after comment from C. Perkins.
- o Added an explicit reference to RFC 5783 and updated this text (after question from M. Scharf).

- o To avoid doubt, added a para about "Each new transport needs to make its own design decisions about how to meet the recommendations and requirements for congestion control."
- o Upated references.

Individual draft -04:

- o Correction of NiTs. Further clarifications.
- o This draft does not attempt to address further alignment with draft-ietf-tcpm-rto-consider. This will form part of a future revision.

Individual draft -05:

- o Moved intro to appendix and re-issued as a live draft.
- o This draft does not attempt to address further alignment with draft-ietf-tcpm-rto-consider. This will form part of a future revision.

#### Author's Address

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

Email: [gorry@erg.abdn.ac.uk](mailto:gorry@erg.abdn.ac.uk)

TCPM Working Group  
Internet-Draft  
Intended status: Experimental  
Expires: January 2, 2020

C. Gomez  
UPC  
J. Crowcroft  
University of Cambridge  
July 1, 2019

TCP ACK Pull  
draft-gomez-tcpm-ack-pull-00

Abstract

Delayed Acknowledgments (ACKs) allow reducing protocol overhead in many scenarios. However, in some cases, Delayed ACKs may significantly degrade network and device performance in terms of link utilization, latency, memory usage and/or energy consumption. This document defines the TCP ACK Pull (AKP) mechanism, which allows a sender to request the ACK for a data segment to be sent without additional delay by the receiver. AKP makes use of one of the reserved bits in the TCP header, which is defined in this specification as the AKP flag.

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 January 2, 2020.

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. Conventions used in this document . . . . .	3
3. ACK Pull Mechanism . . . . .	4
4. The ACK Pull Flag . . . . .	4
5. IANA Actions . . . . .	4
6. Security Considerations . . . . .	5
7. Acknowledgments . . . . .	5
8. References . . . . .	5
8.1. Normative References . . . . .	5
8.2. Informative References . . . . .	5
Authors' Addresses . . . . .	6

## 1. Introduction

Delayed Acknowledgments (ACKs) were specified with the aim to reduce protocol overhead [RFC1122]. With Delayed ACKs, a TCP delays sending an ACK by up to 500 ms (typically, 200 ms), and typically sends an ACK for at least every second segment received in a stream of full-sized segments. This allows combining several segments into a single one (e.g. the application layer response to an application layer data message, and the corresponding ACK), and it also saves up to one of every two ACKs under many traffic patterns (e.g. bulk transfers). The "SHOULD" requirement level for implementing Delayed ACKs in RFC 1122, along with its expected benefits, has led to a widespread deployment of this mechanism.

However, there exist traffic patterns and scenarios for which Delayed ACKs can actually be detrimental to performance. When a segment carrying a message of a size up to one Maximum Segment Size (MSS) is transferred, if the message does not elicit an application-layer response, and a second data segment is not transferred earlier than the Delayed ACK timeout, the ACK is unnecessarily delayed, with a number of negative consequences.

When the Nagle algorithm is used, in some cases the sender may be prevented from sending more data while awaiting the Delayed ACK. In some high bit rate environment (e.g. Gigabit Ethernet) use cases, such a delay may be very large, and link utilization may be dramatically reduced, as the Delayed ACK timeout is several orders of magnitude greater than the Round Trip Time (RTT) [RFC8490].

Delayed ACKs are also detrimental in Internet of Things (IoT) scenarios, where TCP is being increasingly used [I-D.ietf-lwig-tcp-constrained-node-networks]. Many IoT devices, such as sensors, transfer small messages (e.g. containing sensor readings) rather infrequently, therefore if the receiver uses Delayed ACKs, the ACK will often be unnecessarily delayed. The sender cannot release the memory resources associated to a transferred data segment until the ACK is received and processed. This may be a problem for many IoT devices, which are typically memory-constrained, and may even lead to subsequent packet drops if their scarce memory resources are blocked while awaiting an ACK. Moreover, if the IoT device uses a radio interface for communication, in some scenarios Delayed ACKs will lead to increased energy consumption (e.g. with the radio interface of the device staying in receive mode while awaiting the ACK). Since many IoT devices run on small batteries, the device lifetime may be significantly decreased. Furthermore, the delay suffered by the ACK may interact negatively with layer two mechanisms, especially in wireless network technologies where devices remain in low-power states for long intervals [RFC 8352], potentially leading to a further exacerbated delay (by even one or more orders of magnitude).

One approach that cannot be recommended as a general solution to solve the described problems is disabling Delayed ACKs at the receiving TCP. In fact, the latter may interact with a wide variety of devices and many of those may still benefit from the advantages of Delayed ACKs. In addition, in some cases, a sender may offer a mixed traffic pattern comprising single data segments that will lead to unnecessarily delayed ACKs, with other data segments upon which Delayed ACKs will act as intended. Therefore, the solution has to be provided at a per-segment granularity.

This document defines the TCP ACK Pull (AKP) mechanism and an AKP flag in the TCP header. AKP allows a sender to request an ACK to be sent by a receiving TCP without additional delay upon reception of a data segment, by setting the AKP flag in that data segment. The AKP flag uses one of the reserved bits in the TCP header. More specifically, the AKP flag uses bit 6 of byte 13 of the TCP header.

## 2. Conventions used in this document

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

### 3. ACK Pull Mechanism

When a TCP sender needs a data segment to be acknowledged by the receiving TCP without additional delay, the sender sets the AKP flag of the data segment TCP header. A receiving TCP conforming to this specification MUST process the AKP flag of a received segment. If the AKP flag is set, the receiving TCP MUST send an ACK without additional delay, regardless of whether the receiving TCP uses the Delayed ACKs mechanism.

### 4. The ACK Pull Flag

The AKP flag is defined as bit number 6 of the 13th byte of the TCP header. Figure 1 illustrates bytes 13 and 14 of the TCP header, including the AKP flag.

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

Figure 1: Definition of the AKP field within bytes 13 and 14 of the TCP Header.

(Note: as of the writing, bit 7 in the above figure is reserved, although this may change with the publication of [I-D.ietf-tcpm-accurate-ecn].)

### 5. IANA Actions

This document assigns bit 6 of the TCP header flags to the AKP flag. This flag will be defined as shown in Figure 2:

Bit	Name	Reference
6	AKP (ACK Pull)	RFC XXXX

Figure 2

[TO BE REMOVED: IANA is requested to update the existing entry in the Transmission Control Protocol (TCP) Header Flags registration (<https://www.iana.org/assignments/tcp-header-flags/tcp-header-flags.xhtml#tcp-header-flags-1>) for Bit 6 to 'AKP (ACK Pull)'.]

## 6. Security Considerations

TCP ACK Pull introduces a possible Denial of Service (DoS) attack on a resource-constrained receiver. An attacker might send a large number of messages to a victim node, requesting an immediate ACK in response to each one of them. This attack is easily avoided by ignoring the TCP ACK Pull flag.

## 7. Acknowledgments

Stuart Cheshire, Ted Lemon, Michael Scharf, and Christoph Paasch participated in a discussion that was seminal to this document.

Carles Gomez has been funded in part by the Spanish Government (Ministerio de Ciencia, Innovacion y Universidades) through the Jose Castillejo grant CAS18/00170 and by European Regional Development Fund (ERDF) and the Spanish Government through project TEC2016-79988-P, AEI/FEDER, UE. His contribution to this work has been carried out during his stay as a visiting scholar at the Computer Laboratory of the University of Cambridge.

## 8. References

### 8.1. Normative References

- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://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, <<https://www.rfc-editor.org/info/rfc2119>>.

### 8.2. Informative References

- [I-D.ietf-lwig-tcp-constrained-node-networks] Gomez, C., Crowcroft, J., and M. Scharf, "TCP Usage Guidance in the Internet of Things (IoT)", draft-ietf-lwig-tcp-constrained-node-networks-08 (work in progress), June 2019.

- [I-D.ietf-tcpm-accurate-ecn]  
Briscoe, B., Kuehlewind, M., and R. Scheffenegger, "More Accurate ECN Feedback in TCP", draft-ietf-tcpm-accurate-ecn-08 (work in progress), March 2019.
- [RFC8352] Gomez, C., Kovatsch, M., Tian, H., and Z. Cao, Ed., "Energy-Efficient Features of Internet of Things Protocols", RFC 8352, DOI 10.17487/RFC8352, April 2018, <<https://www.rfc-editor.org/info/rfc8352>>.
- [RFC8490] Bellis, R., Cheshire, S., Dickinson, J., Dickinson, S., Lemon, T., and T. Pusateri, "DNS Stateful Operations", RFC 8490, DOI 10.17487/RFC8490, March 2019, <<https://www.rfc-editor.org/info/rfc8490>>.

## Authors' Addresses

Carles Gomez  
UPC  
C/Esteve Terradas, 7  
Castelldefels 08860  
Spain  
  
Email: [carlesgo@entel.upc.edu](mailto:carlesgo@entel.upc.edu)

Jon Crowcroft  
University of Cambridge  
JJ Thomson Avenue  
Cambridge, CB3 0FD  
United Kingdom  
  
Email: [jon.crowcroft@cl.cam.ac.uk](mailto:jon.crowcroft@cl.cam.ac.uk)



NETCONF Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: 8 September 2022

K. Watsen  
Watsen Networks  
M. Scharf  
Hochschule Esslingen  
7 March 2022

YANG Groupings for TCP Clients and TCP Servers  
draft-ietf-netconf-tcp-client-server-12

Abstract

This document defines three YANG 1.1 modules to support the configuration of TCP clients and TCP servers. The modules include basic parameters of a TCP connection relevant for client or server applications, as well as client configuration required for traversing proxies. The modules can be used either standalone or in conjunction with configuration of other stack protocol layers.

Editorial Note (To be removed by RFC Editor)

This draft contains placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed. No other RFC Editor instructions are specified elsewhere in this document.

Artwork in this document contains shorthand references to drafts in progress. Please apply the following replacements:

- \* AAAA --> the assigned RFC value for draft-ietf-netconf-crypto-types
- \* DDDD --> the assigned RFC value for this draft

Artwork in this document contains placeholder values for the date of publication of this draft. Please apply the following replacement:

- \* 2022-03-07 --> the publication date of this draft

The following Appendix section is to be removed prior to publication:

- \* Appendix A. Change Log

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 8 September 2022.

#### Copyright Notice

Copyright (c) 2022 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

#### Table of Contents

1. Introduction . . . . .	3
1.1. Relation to other RFCs . . . . .	3
1.2. Specification Language . . . . .	5
1.3. Adherence to the NMDA . . . . .	5
1.4. Conventions . . . . .	5
2. The "ietf-tcp-common" Module . . . . .	6
2.1. Data Model Overview . . . . .	6
2.2. Example Usage . . . . .	8
2.3. YANG Module . . . . .	8
3. The "ietf-tcp-client" Module . . . . .	11
3.1. Data Model Overview . . . . .	11
3.2. Example Usage . . . . .	13
3.3. YANG Module . . . . .	14
4. The "ietf-tcp-server" Module . . . . .	21
4.1. Data Model Overview . . . . .	21
4.2. Example Usage . . . . .	23
4.3. YANG Module . . . . .	23
5. Security Considerations . . . . .	26
5.1. The "ietf-tcp-common" YANG Module . . . . .	26
5.2. The "ietf-tcp-client" YANG Module . . . . .	26

5.3. The "ietf-tcp-server" YANG Module . . . . .	27
6. IANA Considerations . . . . .	28
6.1. The "IETF XML" Registry . . . . .	28
6.2. The "YANG Module Names" Registry . . . . .	28
7. References . . . . .	29
7.1. Normative References . . . . .	29
7.2. Informative References . . . . .	29
Appendix A. Change Log . . . . .	31
A.1. 00 to 01 . . . . .	31
A.2. 01 to 02 . . . . .	32
A.3. 02 to 03 . . . . .	32
A.4. 03 to 04 . . . . .	32
A.5. 04 to 05 . . . . .	32
A.6. 05 to 06 . . . . .	32
A.7. 06 to 07 . . . . .	32
A.8. 07 to 08 . . . . .	32
A.9. 08 to 09 . . . . .	33
A.10. 09 to 10 . . . . .	33
A.11. 10 to 11 . . . . .	33
A.12. 11 to 12 . . . . .	33
Acknowledgements . . . . .	33
Authors' Addresses . . . . .	33

## 1. Introduction

This document defines three YANG 1.1 [RFC7950] modules to support the configuration of TCP clients and TCP servers (TCP is defined in [RFC0793]), either as standalone or in conjunction with configuration of other stack protocol layers.

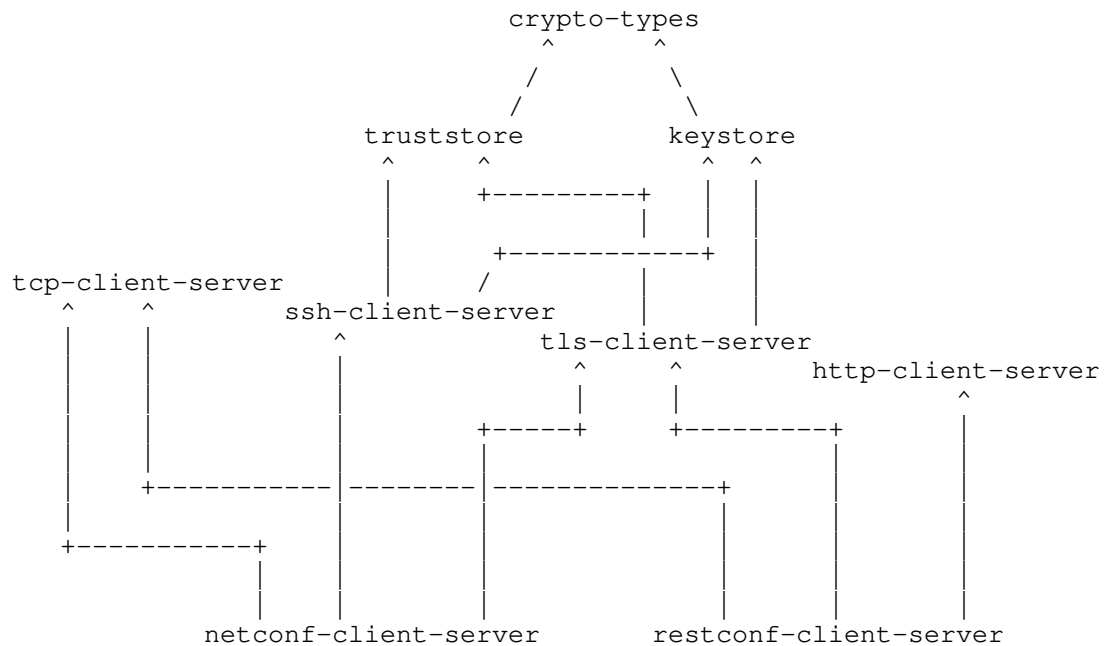
The modules focus on three different types of base TCP parameters that matter for TCP-based applications: First, the modules cover fundamental configuration of a TCP client or TCP server application, such as addresses and port numbers. Second, a reusable grouping enables modification of application-specific parameters for a TCP connections, such as use of TCP keep-alives. And third, client configuration for traversing proxies is included as well. In each case, the modules have a very narrow scope and focus on a minimum set of required parameters.

### 1.1. Relation to other RFCs

This document presents one or more YANG modules [RFC7950] that are part of a collection of RFCs that work together to, ultimately, enable the configuration of the clients and servers of both the NETCONF [RFC6241] and RESTCONF [RFC8040] protocols.

The modules have been defined in a modular fashion to enable their use by other efforts, some of which are known to be in progress at the time of this writing, with many more expected to be defined in time.

The normative dependency relationship between the various RFCs in the collection is presented in the below diagram. The labels in the diagram represent the primary purpose provided by each RFC. Hyperlinks to each RFC are provided below the diagram.



Label in Diagram	Originating RFC
crypto-types	[I-D.ietf-netconf-crypto-types]
truststore	[I-D.ietf-netconf-trust-anchors]
keystore	[I-D.ietf-netconf-keystore]
tcp-client-server	[I-D.ietf-netconf-tcp-client-server]
ssh-client-server	[I-D.ietf-netconf-ssh-client-server]
tls-client-server	[I-D.ietf-netconf-tls-client-server]
http-client-server	[I-D.ietf-netconf-http-client-server]
netconf-client-server	[I-D.ietf-netconf-netconf-client-server]
restconf-client-server	[I-D.ietf-netconf-restconf-client-server]

Table 1: Label to RFC Mapping

## 1.2. Specification Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 1.3. Adherence to the NMDA

This document is compliant with the Network Management Datastore Architecture (NMDA) [RFC8342]. It does not define any protocol accessible nodes that are "config false".

## 1.4. Conventions

Various examples used in this document use a placeholder value for binary data that has been base64 encoded (e.g., "BASE64VALUE="). This placeholder value is used as real base64 encoded structures are often many lines long and hence distracting to the example being presented.

## 2. The "ietf-tcp-common" Module

This section defines a YANG 1.1 module called "ietf-tcp-common". A high-level overview of the module is provided in Section 2.1. Examples illustrating the module's use are provided in Examples (Section 2.2). The YANG module itself is defined in Section 2.3.

### 2.1. Data Model Overview

This section provides an overview of the "ietf-tcp-common" module in terms of its features and groupings.

#### 2.1.1. Model Scope

This document defines a common "grouping" statement for basic TCP connection parameters that matter to applications. In some TCP stacks, such parameters can also directly be set by an application using system calls, such as the sockets API. The base YANG model in this document focuses on modeling TCP keep-alives. This base model can be extended as needed.

#### 2.1.2. Features

The following diagram lists all the "feature" statements defined in the "ietf-tcp-common" module:

Features:

+-- keepalives-supported

| The diagram above uses syntax that is similar to but not  
| defined in [RFC8340].

#### 2.1.3. Groupings

The "ietf-tcp-common" module defines the following "grouping" statement:

\* tcp-common-grouping

This grouping is presented in the following subsection.

##### 2.1.3.1. The "tcp-common-grouping" Grouping

The following tree diagram [RFC8340] illustrates the "tcp-common-grouping" grouping:

```
grouping tcp-common-grouping
  +-- keepalives! {keepalives-supported}?
    +-- idle-time          uint16
    +-- max-probes         uint16
    +-- probe-interval    uint16
```

Comments:

- \* The "keepalives" node is a "presence" node so that the mandatory descendant nodes do not imply that keepalives must be configured.
- \* The "idle-time", "max-probes", and "probe-interval" nodes have the common meanings. Please see the YANG module in Section 2.3 for details.

#### 2.1.4. Protocol-accessible Nodes

The "ietf-tcp-common" module defines only "grouping" statements that are used by other modules to instantiate protocol-accessible nodes.

#### 2.1.5. Guidelines for Configuring TCP Keep-Alives

Network stacks may include "keep-alives" in their TCP implementations, although this practice is not universally accepted. If keep-alives are included, [RFC1122] mandates that the application MUST be able to turn them on or off for each TCP connection, and that they MUST default to off.

Keep-alive mechanisms exist in many protocols. Depending on the protocol stack, TCP keep-alives may only be one out of several alternatives. Which mechanism(s) to use depends on the use case and application requirements. If keep-alives are needed by an application, it is RECOMMENDED that the aliveness check happens only at the protocol layers that are meaningful to the application.

A TCP keep-alive mechanism SHOULD only be invoked in server applications that might otherwise hang indefinitely and consume resources unnecessarily if a client crashes or aborts a connection during a network failure [RFC1122]. TCP keep-alives may consume significant resources both in the network and in endpoints (e.g., battery power). In addition, frequent keep-alives risk network congestion. The higher the frequency of keep-alives, the higher the overhead.

Given the cost of keep-alives, parameters have to be configured carefully:

- \* The default idle interval (leaf "idle-time") MUST default to no less than two hours, i.e., 7200 seconds [RFC1122]. A lower value MAY be configured, but keep-alive messages SHOULD NOT be transmitted more frequently than once every 15 seconds. Longer intervals SHOULD be used when possible.
- \* The maximum number of sequential keep-alive probes that can fail (leaf "max-probes") trades off responsiveness and robustness against packet loss. ACK segments that contain no data are not reliably transmitted by TCP. Consequently, if a keep-alive mechanism is implemented it MUST NOT interpret failure to respond to any specific probe as a dead connection [RFC1122]. Typically, a single-digit number should suffice.
- \* TCP implementations may include a parameter for the number of seconds between TCP keep-alive probes (leaf "probe-interval"). In order to avoid congestion, the time interval between probes MUST NOT be smaller than one second. Significantly longer intervals SHOULD be used. It is important to note that keep-alive probes (or replies) can get dropped due to network congestion. Sending further probe messages into a congested path after a short interval, without backing off timers, could cause harm and result in a congestion collapse. Therefore it is essential to pick a large, conservative value for this interval.

## 2.2. Example Usage

This section presents an example showing the "tcp-common-grouping" populated with some data.

```
<!-- The outermost element below doesn't exist in the data model. -->
<!-- It simulates if the "grouping" were a "container" instead. -->

<tcp-common xmlns="urn:ietf:params:xml:ns:yang:ietf-tcp-common">
  <keepalives>
    <idle-time>15</idle-time>
    <max-probes>3</max-probes>
    <probe-interval>30</probe-interval>
  </keepalives>
</tcp-common>
```

## 2.3. YANG Module

The ietf-tcp-common YANG module references [RFC6991].

```
<CODE BEGINS> file "ietf-tcp-common@2022-03-07.yang"
```



```
module ietf-tcp-common {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-tcp-common";
  prefix tcpcmn;

  organization
    "IETF NETCONF (Network Configuration) Working Group and the
     IETF TCP Maintenance and Minor Extensions (TCPM) Working Group";

  contact
    "WG Web:  https://datatracker.ietf.org/wg/netconf
              https://datatracker.ietf.org/wg/tcpm
    WG List:  NETCONF WG list <mailto:netconf@ietf.org>
              TCPM WG list <mailto:tcpm@ietf.org>
    Authors:  Kent Watsen <mailto:kent+ietf@watsen.net>
              Michael Scharf
              <mailto:michael.scharf@hs-esslingen.de>";

  description
    "This module defines reusable groupings for TCP commons that
     can be used as a basis for specific TCP common instances.

     Copyright (c) 2021 IETF Trust and the persons identified
     as authors of the code. All rights reserved.

     Redistribution and use in source and binary forms, with
     or without modification, is permitted pursuant to, and
     subject to the license terms contained in, the Revised
     BSD License set forth in Section 4.c of the IETF Trust's
     Legal Provisions Relating to IETF Documents
     (https://trustee.ietf.org/license-info).https://www.rfc-editor.org/info/rfcDDDD); see the RFC
     itself for full legal notices.

     The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL',
     'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED',
     'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document
     are to be interpreted as described in BCP 14 (RFC 2119)
     (RFC 8174) when, and only when, they appear in all
     capitals, as shown here."

  revision 2022-03-07 {
    description
      "Initial version";
    reference
      "RFC DDDD: YANG Groupings for TCP Clients and TCP Servers";
```

```
}

// Features

feature keepalives-supported {
  description
    "Indicates that keepalives are supported.";
}

// Groupings

grouping tcp-common-grouping {
  description
    "A reusable grouping for configuring TCP parameters common
    to TCP connections as well as the operating system as a
    whole.";
  container keepalives {
    if-feature "keepalives-supported";
    presence
      "Indicates that keepalives are enabled. This statement is
      present so the mandatory descendant nodes do not imply that
      this node must be configured.";
    description
      "Configures the keep-alive policy, to proactively test the
      aliveness of the TCP peer. An unresponsive TCP peer is
      dropped after approximately (idle-time + max-probes
      * probe-interval) seconds.";
    leaf idle-time {
      type uint16 {
        range "1..max";
      }
      units "seconds";
      mandatory true;
      description
        "Sets the amount of time after which if no data has been
        received from the TCP peer, a TCP-level probe message
        will be sent to test the aliveness of the TCP peer.
        Two hours (7200 seconds) is safe value, per RFC 1122.";
      reference
        "RFC 1122:
        Requirements for Internet Hosts -- Communication Layers";
    }
    leaf max-probes {
      type uint16 {
        range "1..max";
      }
      mandatory true;
      description
```

```
        "Sets the maximum number of sequential keep-alive probes
        that can fail to obtain a response from the TCP peer
        before assuming the TCP peer is no longer alive.";
    }
    leaf probe-interval {
        type uint16 {
            range "1..max";
        }
        units "seconds";
        mandatory true;
        description
            "Sets the time interval between failed probes. The interval
            SHOULD be significantly longer than one second in order to
            avoid harm on a congested link.";
    }
} // container keepalives
} // grouping tcp-common-grouping

}

<CODE ENDS>
```

### 3. The "ietf-tcp-client" Module

This section defines a YANG 1.1 module called "ietf-tcp-client". A high-level overview of the module is provided in Section 3.1. Examples illustrating the module's use are provided in Examples (Section 3.2). The YANG module itself is defined in Section 3.3.

#### 3.1. Data Model Overview

This section provides an overview of the "ietf-tcp-client" module in terms of its features and groupings.

##### 3.1.1. Features

The following diagram lists all the "feature" statements defined in the "ietf-tcp-client" module:

Features:

```
+-- local-binding-supported
+-- tcp-client-keepalives
+-- proxy-connect
+-- socks5-gss-api
+-- socks5-username-password
```

| The diagram above uses syntax that is similar to but not  
| defined in [RFC8340].

### 3.1.2. Groupings

The "ietf-tcp-client" module defines the following "grouping" statement:

```
* tcp-client-grouping
```

This grouping is presented in the following subsection.

#### 3.1.2.1. The "tcp-client-grouping" Grouping

The following tree diagram [RFC8340] illustrates the "tcp-client-grouping" grouping:

```
grouping tcp-client-grouping
+-- remote-address          inet:host
+-- remote-port?           inet:port-number
+-- local-address?         inet:ip-address
|   {local-binding-supported}?
+-- local-port?            inet:port-number
|   {local-binding-supported}?
+-- proxy-server! {proxy-connect}?
|   +-- (proxy-type)
|       +--:(socks4)
|           +-- socks4-parameters
|               +-- remote-address    inet:ip-address
|               +-- remote-port?     inet:port-number
|       +--:(socks4a)
|           +-- socks4a-parameters
|               +-- remote-address    inet:host
|               +-- remote-port?     inet:port-number
|       +--:(socks5)
|           +-- socks5-parameters
|               +-- remote-address    inet:host
|               +-- remote-port?     inet:port-number
|               +-- authentication-parameters!
|                   +-- (auth-type)
|                       +--:(gss-api) {socks5-gss-api}?
|                           |   +-- gss-api
|                       +--:(username-password)
|                           |   {socks5-username-password}?
|                           +-- username-password
|                               +-- username          string
|                               +--u ct:password-grouping
+---u tcpcmn:tcp-common-grouping
```

Comments:

- \* The "remote-address" node, which is mandatory, may be configured as an IPv4 address, an IPv6 address, a hostname.
- \* The "remote-port" node is not mandatory, but its default value is the invalid value '0', thus forcing the consuming data model to refine it in order to provide it an appropriate default value.
- \* The "local-address" node, which is enabled by the "local-binding-supported" feature (Section 2.1.2), may be configured as an IPv4 address, an IPv6 address, or a wildcard value.
- \* The "local-port" node, which is enabled by the "local-binding-supported" feature (Section 2.1.2), is not mandatory. Its default value is '0', indicating that the operating system can pick an arbitrary port number.
- \* The "proxy-server" node is enabled by a "feature" statement and, for servers that enable it, is a "presence" container so that the descendant "mandatory true" choice node does not imply that the proxy-server node must be configured.
- \* This grouping uses the "tcp-common-grouping" grouping discussed in Section 2.1.3.1.

### 3.1.3. Protocol-accessible Nodes

The "ietf-tcp-client" module defines only "grouping" statements that are used by other modules to instantiate protocol-accessible nodes.

### 3.2. Example Usage

This section presents two examples showing the "tcp-client-grouping" populated with some data. This example shows a TCP-client configured to not connect via a proxy:

```
<!-- The outermost element below doesn't exist in the data model. -->
<!-- It simulates if the "grouping" were a "container" instead. -->

<tcp-client xmlns="urn:ietf:params:xml:ns:yang:ietf-tcp-client">
  <remote-address>www.example.com</remote-address>
  <remote-port>443</remote-port>
  <local-address>0.0.0.0</local-address>
  <local-port>0</local-port>
  <keepalives>
    <idle-time>15</idle-time>
    <max-probes>3</max-probes>
    <probe-interval>30</probe-interval>
  </keepalives>
</tcp-client>
```

This example shows a TCP-client configured to connect via a proxy:

```
<!-- The outermost element below doesn't exist in the data model. -->
<!-- It simulates if the "grouping" were a "container" instead. -->

<tcp-client xmlns="urn:ietf:params:xml:ns:yang:ietf-tcp-client">
  <remote-address>www.example.com</remote-address>
  <remote-port>443</remote-port>
  <local-address>0.0.0.0</local-address>
  <local-port>0</local-port>
  <proxy-server>
    <socks5-parameters>
      <remote-address>proxy.my-domain.com</remote-address>
      <remote-port>1080</remote-port>
      <authentication-parameters>
        <username-password>
          <username>foobar</username>
          <cleartext-password>secret</cleartext-password>
        </username-password>
      </authentication-parameters>
    </socks5-parameters>
  </proxy-server>
  <keepalives>
    <idle-time>15</idle-time>
    <max-probes>3</max-probes>
    <probe-interval>30</probe-interval>
  </keepalives>
</tcp-client>
```

### 3.3. YANG Module

The ietf-tcp-client YANG module references [RFC6991].

```
<CODE BEGINS> file "ietf-tcp-client@2022-03-07.yang"

module ietf-tcp-client {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-tcp-client";
  prefix tcpc;

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-crypto-types {
    prefix ct;
    reference
      "RFC AAAA: YANG Data Types and Groupings for Cryptography";
  }

  import ietf-tcp-common {
    prefix tcpcmn;
    reference
      "RFC DDDD: YANG Groupings for TCP Clients and TCP Servers";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group and the
     IETF TCP Maintenance and Minor Extensions (TCPM) Working Group";

  contact
    "WG Web:  https://datatracker.ietf.org/wg/netconf
     https://datatracker.ietf.org/wg/tcpm
    WG List:  NETCONF WG list <mailto:netconf@ietf.org>
              TCPM WG list <mailto:tcpm@ietf.org>
    Authors:  Kent Watsen <mailto:kent+ietf@watsen.net>
              Michael Scharf
              <mailto:michael.scharf@hs-esslingen.de>;

  description
    "This module defines reusable groupings for TCP clients that
     can be used as a basis for specific TCP client instances.

    Copyright (c) 2021 IETF Trust and the persons identified
    as authors of the code. All rights reserved.

    Redistribution and use in source and binary forms, with
    or without modification, is permitted pursuant to, and
    subject to the license terms contained in, the Revised
```

BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC DDDD (<https://www.rfc-editor.org/info/rfcDDDD>); see the RFC itself for full legal notices.

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document are to be interpreted as described in BCP 14 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here.";

```
revision 2022-03-07 {
  description
    "Initial version";
  reference
    "RFC DDDD: YANG Groupings for TCP Clients and TCP Servers";
}
```

```
// Features
```

```
feature local-binding-supported {
  description
    "Indicates that the server supports configuring local
    bindings (i.e., the local address and local port) for
    TCP clients.";
}
```

```
feature tcp-client-keepalives {
  description
    "Per socket TCP keepalive parameters are configurable for
    TCP clients on the server implementing this feature.";
}
```

```
feature proxy-connect {
  description
    "Proxy connection configuration is configurable for
    TCP clients on the server implementing this feature.";
}
```

```
feature socks5-gss-api {
  description
    "Indicates that the server supports authenticating
    using GSSAPI when initiating TCP connections via
    and SOCKS Version 5 proxy server.";
```



```
    reference
      "RFC 1928: SOCKS Protocol Version 5";
  }

  feature socks5-username-password {
    description
      "Indicates that the server supports authenticating using
       username/password when initiating TCP connections via
       and SOCKS Version 5 proxy server.";
    reference
      "RFC 1928: SOCKS Protocol Version 5";
  }

  // Groupings

  grouping tcp-client-grouping {
    description
      "A reusable grouping for configuring a TCP client.

      Note that this grouping uses fairly typical descendant
      node names such that a stack of 'uses' statements will
      have name conflicts. It is intended that the consuming
      data model will resolve the issue (e.g., by wrapping
      the 'uses' statement in a container called
      'tcp-client-parameters'). This model purposely does
      not do this itself so as to provide maximum flexibility
      to consuming models.";

    leaf remote-address {
      type inet:host;
      mandatory true;
      description
        "The IP address or hostname of the remote peer to
         establish a connection with. If a domain name is
         configured, then the DNS resolution should happen on
         each connection attempt. If the DNS resolution
         results in multiple IP addresses, the IP addresses
         are tried according to local preference order until
         a connection has been established or until all IP
         addresses have failed.";
    }
    leaf remote-port {
      type inet:port-number;
      default "0";
      description
        "The IP port number for the remote peer to establish a
         connection with. An invalid default value (0) is used
         (instead of 'mandatory true') so that as application
```

```
        level data model may 'refine' it with an application
        specific default port number value.";
    }
    leaf local-address {
        if-feature "local-binding-supported";
        type inet:ip-address;
        description
            "The local IP address/interface (VRF?) to bind to for when
            connecting to the remote peer.  INADDR_ANY ('0.0.0.0') or
            INADDR6_ANY ('0:0:0:0:0:0:0:0' a.k.a. '::') MAY be used to
            explicitly indicate the implicit default, that the server
            can bind to any IPv4 or IPv6 addresses, respectively.";
    }
    leaf local-port {
        if-feature "local-binding-supported";
        type inet:port-number;
        default "0";
        description
            "The local IP port number to bind to for when connecting
            to the remote peer.  The port number '0', which is the
            default value, indicates that any available local port
            number may be used.";
    }
    container proxy-server {
        if-feature "proxy-connect";
        presence
            "Indicates that a proxy connection has been configured.
            Present so that the mandatory descendant nodes do not
            imply that this node must be configured.";
        choice proxy-type {
            mandatory true;
            description
                "Selects a proxy connection protocol.";
            case socks4 {
                container socks4-parameters {
                    leaf remote-address {
                        type inet:ip-address;
                        mandatory true;
                        description
                            "The IP address of the proxy server.";
                    }
                    leaf remote-port {
                        type inet:port-number;
                        default "1080";
                        description
                            "The IP port number for the proxy server.";
                    }
                }
            }
        }
        description
```

```
        "Parameters for connecting to a TCP-based proxy
        server using the SOCKS4 protocol.";
    reference
        "SOCKS, Proceedings: 1992 Usenix Security Symposium.";
    }
}
case socks4a {
    container socks4a-parameters {
        leaf remote-address {
            type inet:host;
            mandatory true;
            description
                "The IP address or hostname of the proxy server.";
        }
        leaf remote-port {
            type inet:port-number;
            default "1080";
            description
                "The IP port number for the proxy server.";
        }
        description
            "Parameters for connecting to a TCP-based proxy
            server using the SOCKS4a protocol.";
        reference
            "SOCKS Proceedings:
            1992 Usenix Security Symposium.
            OpenSSH message:
            SOCKS 4A: A Simple Extension to SOCKS 4 Protocol
            https://www.openssh.com/txt/socks4a.protocol";
    }
}
case socks5 {
    container socks5-parameters {
        leaf remote-address {
            type inet:host;
            mandatory true;
            description
                "The IP address or hostname of the proxy server.";
        }
        leaf remote-port {
            type inet:port-number;
            default "1080";
            description
                "The IP port number for the proxy server.";
        }
        container authentication-parameters {
            presence
                "Indicates that an authentication mechanism
```

```
has been configured. Present so that the
mandatory descendant nodes do not imply that
this node must be configured.";
description
  "A container for SOCKS Version 5 authentication
  mechanisms.

  A complete list of methods is defined at:
  https://www.iana.org/assignments/socks-methods
  /socks-methods.xhtml.";
reference
  "RFC 1928: SOCKS Protocol Version 5";
choice auth-type {
  mandatory true;
  description
    "A choice amongst supported SOCKS Version 5
    authentication mechanisms.";
  case gss-api {
    if-feature "socks5-gss-api";
    container gss-api {
      description
        "Contains GSS-API configuration. Defines
        as an empty container to enable specific
        GSS-API configuration to be augmented in
        by future modules.";
      reference
        "RFC 1928: SOCKS Protocol Version 5
        RFC 2743: Generic Security Service
        Application Program Interface
        Version 2, Update 1";
    }
  }
  case username-password {
    if-feature "socks5-username-password";
    container username-password {
      leaf username {
        type string;
        mandatory true;
        description
          "The 'username' value to use for client
          identification.";
      }
      uses ct:password-grouping {
        description
          "The password to be used for client
          authentication.";
      }
    }
  }
  description
```

```
        "Contains Username/Password configuration.";
        reference
            "RFC 1929: Username/Password Authentication
              for SOCKS V5";
    }
}
}
description
    "Parameters for connecting to a TCP-based proxy server
      using the SOCKS5 protocol.";
reference
    "RFC 1928: SOCKS Protocol Version 5";
}
}
description
    "Proxy server settings.";
}

uses tcpcmn:tcp-common-grouping {
    augment "keepalives" {
        if-feature "tcp-client-keepalives";
        description
            "Add an if-feature statement so that implementations
              can choose to support TCP client keepalives.";
    }
}
}
```

<CODE ENDS>

#### 4. The "ietf-tcp-server" Module

This section defines a YANG 1.1 module called "ietf-tcp-server". A high-level overview of the module is provided in Section 4.1. Examples illustrating the module's use are provided in Examples (Section 4.2). The YANG module itself is defined in Section 4.3.

##### 4.1. Data Model Overview

This section provides an overview of the "ietf-tcp-server" module in terms of its features and groupings.

#### 4.1.1. Features

The following diagram lists all the "feature" statements defined in the "ietf-tcp-server" module:

Features:

```
+-- tcp-server-keepalives
```

```
| The diagram above uses syntax that is similar to but not
| defined in [RFC8340].
```

#### 4.1.2. Groupings

The "ietf-tcp-server" module defines the following "grouping" statement:

```
* tcp-server-grouping
```

This grouping is presented in the following subsection.

##### 4.1.2.1. The "tcp-server-grouping" Grouping

The following tree diagram [RFC8340] illustrates the "tcp-server-grouping" grouping:

```
grouping tcp-server-grouping
  +-- local-address          inet:ip-address
  +-- local-port?           inet:port-number
  +---u tcpcmn:tcp-common-grouping
```

Comments:

- \* The "local-address" node, which is mandatory, may be configured as an IPv4 address, an IPv6 address, or a wildcard value.
- \* The "local-port" node is not mandatory, but its default value is the invalid value '0', thus forcing the consuming data model to refine it in order to provide it an appropriate default value.
- \* This grouping uses the "tcp-common-grouping" grouping discussed in Section 2.1.3.1.

#### 4.1.3. Protocol-accessible Nodes

The "ietf-tcp-server" module defines only "grouping" statements that are used by other modules to instantiate protocol-accessible nodes.

#### 4.2. Example Usage

This section presents an example showing the "tcp-server-grouping" populated with some data.

```
<!-- The outermost element below doesn't exist in the data model. -->
<!-- It simulates if the "grouping" were a "container" instead. -->
```

```
<tcp-server xmlns="urn:ietf:params:xml:ns:yang:ietf-tcp-server">
  <local-address>10.20.30.40</local-address>
  <local-port>7777</local-port>
  <keepalives>
    <idle-time>15</idle-time>
    <max-probes>3</max-probes>
    <probe-interval>30</probe-interval>
  </keepalives>
</tcp-server>
```

#### 4.3. YANG Module

The ietf-tcp-server YANG module references [RFC6991].

```
<CODE BEGINS> file "ietf-tcp-server@2022-03-07.yang"
```

```
module ietf-tcp-server {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-tcp-server";
  prefix tcps;

  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  import ietf-tcp-common {
    prefix tcpcmn;
    reference
      "RFC DDDD: YANG Groupings for TCP Clients and TCP Servers";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group and the
     IETF TCP Maintenance and Minor Extensions (TCPM) Working Group";

  contact
    "WG Web:  https://datatracker.ietf.org/wg/netconf
     https://datatracker.ietf.org/wg/tcpm
```

WG List: NETCONF WG list <mailto:netconf@ietf.org>  
TCPM WG list <mailto:tcpm@ietf.org>  
Authors: Kent Watsen <mailto:kent+ietf@watsen.net>  
Michael Scharf  
<mailto:michael.scharf@hs-esslingen.de>;

description

"This module defines reusable groupings for TCP servers that can be used as a basis for specific TCP server instances.

Copyright (c) 2021 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Revised BSD License set forth in Section 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>).

This version of this YANG module is part of RFC DDDD (<https://www.rfc-editor.org/info/rfcDDDD>); see the RFC itself for full legal notices.

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document are to be interpreted as described in BCP 14 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here.";

```
revision 2022-03-07 {  
  description  
    "Initial version";  
  reference  
    "RFC DDDD: YANG Groupings for TCP Clients and TCP Servers";  
}
```

// Features

```
feature tcp-server-keepalives {  
  description  
    "Per socket TCP keepalive parameters are configurable for  
    TCP servers on the server implementing this feature.";  
}
```

// Groupings



```
grouping tcp-server-grouping {
  description
    "A reusable grouping for configuring a TCP server.

    Note that this grouping uses fairly typical descendant
    node names such that a stack of 'uses' statements will
    have name conflicts. It is intended that the consuming
    data model will resolve the issue (e.g., by wrapping
    the 'uses' statement in a container called
    'tcp-server-parameters'). This model purposely does
    not do this itself so as to provide maximum flexibility
    to consuming models.";
  leaf local-address {
    type inet:ip-address;
    mandatory true;
    description
      "The local IP address to listen on for incoming
      TCP client connections. INADDR_ANY (0.0.0.0) or
      INADDR6_ANY (0:0:0:0:0:0:0:0 a.k.a. ::) MUST be
      used when the server is to listen on all IPv4 or
      IPv6 addresses, respectively.";
  }
  leaf local-port {
    type inet:port-number;
    default "0";
    description
      "The local port number to listen on for incoming TCP
      client connections. An invalid default value (0)
      is used (instead of 'mandatory true') so that an
      application level data model may 'refine' it with
      an application specific default port number value.";
  }
  uses tcpcmn:tcp-common-grouping {
    augment "keepalives" {
      if-feature "tcp-server-keepalives";
      description
        "Add an if-feature statement so that implementations
        can choose to support TCP server keepalives.";
    }
  }
}
}
```

<CODE ENDS>

## 5. Security Considerations

### 5.1. The "ietf-tcp-common" YANG Module

The "ietf-tcp-common" YANG module defines "grouping" statements that are designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC8341] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

Since the module in this document only define groupings, these considerations are primarily for the designers of other modules that use these groupings.

None of the readable data nodes defined in this YANG module are considered sensitive or vulnerable in network environments. The NACM "default-deny-all" extension has not been set for any data nodes defined in this module.

None of the writable data nodes defined in this YANG module are considered sensitive or vulnerable in network environments. The NACM "default-deny-write" extension has not been set for any data nodes defined in this module.

This module does not define any RPCs, actions, or notifications, and thus the security consideration for such is not provided here.

### 5.2. The "ietf-tcp-client" YANG Module

The "ietf-tcp-client" YANG module defines "grouping" statements that are designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC8341] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

Since the module in this document only define groupings, these considerations are primarily for the designers of other modules that use these groupings.

One readable data node defined in this YANG module may be considered sensitive or vulnerable in some network environments. This node is as follows:

- \* The "proxy-server/socks5-parameters/authentication-parameters/username-password/password" node:

The cleartext "password" node defined in the "tcp-client-grouping" grouping is additionally sensitive to read operations such that, in normal use cases, it should never be returned to a client. For this reason, the NACM extension "default-deny-all" has been applied to it.

None of the writable data nodes defined in this YANG module are considered sensitive or vulnerable in network environments. The NACM "default-deny-write" extension has not been set for any data nodes defined in this module.

This module does not define any RPCs, actions, or notifications, and thus the security consideration for such is not provided here.

Implementations are RECOMMENDED to implement the "local-binding-supported" feature for cryptographically-secure protocols, so as to enable more granular ingress/egress firewall rulebases. It is NOT RECOMMENDED to implement this feature for unsecure protocols, as per [RFC6056].

### 5.3. The "ietf-tcp-server" YANG Module

The "ietf-tcp-server" YANG module defines "grouping" statements that are designed to be accessed via YANG based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. Both of these protocols have mandatory-to-implement secure transport layers (e.g., SSH, TLS) with mutual authentication.

The NETCONF access control model (NACM) [RFC8341] provides the means to restrict access for particular users to a pre-configured subset of all available protocol operations and content.

Since the module in this document only define groupings, these considerations are primarily for the designers of other modules that use these groupings.

None of the readable data nodes defined in this YANG module are considered sensitive or vulnerable in network environments. The NACM "default-deny-all" extension has not been set for any data nodes defined in this module.

None of the writable data nodes defined in this YANG module are considered sensitive or vulnerable in network environments. The NACM "default-deny-write" extension has not been set for any data nodes defined in this module.

This module does not define any RPCs, actions, or notifications, and thus the security consideration for such is not provided here.

## 6. IANA Considerations

### 6.1. The "IETF XML" Registry

This document registers two URIs in the "ns" subregistry of the IETF XML Registry [RFC3688]. Following the format in [RFC3688], the following registrations are requested:

URI: urn:ietf:params:xml:ns:yang:ietf-tcp-common  
Registrant Contact: The IESG  
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-tcp-client  
Registrant Contact: The IESG  
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-tcp-server  
Registrant Contact: The IESG  
XML: N/A, the requested URI is an XML namespace.

### 6.2. The "YANG Module Names" Registry

This document registers two YANG modules in the YANG Module Names registry [RFC6020]. Following the format in [RFC6020], the following registrations are requested:

name: ietf-tcp-common  
namespace: urn:ietf:params:xml:ns:yang:ietf-tcp-common  
prefix: tcpcmn  
reference: RFC DDDD

name: ietf-tcp-client  
namespace: urn:ietf:params:xml:ns:yang:ietf-tcp-client  
prefix: tcpc  
reference: RFC DDDD

name: ietf-tcp-server  
namespace: urn:ietf:params:xml:ns:yang:ietf-tcp-server  
prefix: tcps  
reference: RFC DDDD

## 7. References

### 7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.

### 7.2. Informative References

- [I-D.ietf-netconf-crypto-types]  
Watsen, K., "YANG Data Types and Groupings for Cryptography", Work in Progress, Internet-Draft, draft-ietf-netconf-crypto-types-21, 14 September 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-netconf-crypto-types-21>>.
- [I-D.ietf-netconf-http-client-server]  
Watsen, K., "YANG Groupings for HTTP Clients and HTTP Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-http-client-server-08, 14 December 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-netconf-http-client-server-08>>.

[I-D.ietf-netconf-keystore]

Watsen, K., "A YANG Data Model for a Keystore", Work in Progress, Internet-Draft, draft-ietf-netconf-keystore-23, 14 December 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-netconf-keystore-23>>.

[I-D.ietf-netconf-netconf-client-server]

Watsen, K., "NETCONF Client and Server Models", Work in Progress, Internet-Draft, draft-ietf-netconf-netconf-client-server-24, 14 December 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-netconf-netconf-client-server-24>>.

[I-D.ietf-netconf-restconf-client-server]

Watsen, K., "RESTCONF Client and Server Models", Work in Progress, Internet-Draft, draft-ietf-netconf-restconf-client-server-24, 14 December 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-netconf-restconf-client-server-24>>.

[I-D.ietf-netconf-ssh-client-server]

Watsen, K., "YANG Groupings for SSH Clients and SSH Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-ssh-client-server-26, 14 December 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-netconf-ssh-client-server-26>>.

[I-D.ietf-netconf-tcp-client-server]

Watsen, K. and M. Scharf, "YANG Groupings for TCP Clients and TCP Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-tcp-client-server-11, 14 December 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-netconf-tcp-client-server-11>>.

[I-D.ietf-netconf-tls-client-server]

Watsen, K., "YANG Groupings for TLS Clients and TLS Servers", Work in Progress, Internet-Draft, draft-ietf-netconf-tls-client-server-26, 14 December 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-netconf-tls-client-server-26>>.

[I-D.ietf-netconf-trust-anchors]

Watsen, K., "A YANG Data Model for a Truststore", Work in Progress, Internet-Draft, draft-ietf-netconf-trust-anchors-16, 14 December 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-netconf-trust-anchors-16>>.

- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC6056] Larsen, M. and F. Gont, "Recommendations for Transport-Protocol Port Randomization", BCP 156, RFC 6056, DOI 10.17487/RFC6056, January 2011, <<https://www.rfc-editor.org/info/rfc6056>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.
- [RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <<https://www.rfc-editor.org/info/rfc8342>>.

## Appendix A. Change Log

This section is to be removed before publishing as an RFC.

### A.1. 00 to 01

- \* Added 'local-binding-supported' feature to TCP-client model.
- \* Added 'keepalives-supported' feature to TCP-common model.
- \* Added 'external-endpoint-values' container and 'external-endpoints' feature to TCP-server model.

## A.2. 01 to 02

- \* Removed the 'external-endpoint-values' container and 'external-endpoints' feature from the TCP-server model.

## A.3. 02 to 03

- \* Moved the common model section to be before the client and server specific sections.
- \* Added sections "Model Scope" and "Usage Guidelines for Configuring TCP Keep-Alives" to the common model section.

## A.4. 03 to 04

- \* Fixed a few typos.

## A.5. 04 to 05

- \* Removed commented out "grouping tcp-system-grouping" statement kept for reviewers.
- \* Added a "Note to Reviewers" note to first page.

## A.6. 05 to 06

- \* Added support for TCP proxies.

## A.7. 06 to 07

- \* Expanded "Data Model Overview section(s) [remove "wall" of tree diagrams].
- \* Updated the Security Considerations section.

## A.8. 07 to 08

- \* Added missing IANA registration for "ietf-tcp-common"
- \* Added "mandatory true" for the "username" and "password" leafs
- \* Added an example of a TCP-client configured to connect via a proxy
- \* Fixed issues found by the SecDir review of the "keystore" draft.
- \* Updated the "ietf-tcp-client" module to use the new "password-grouping" grouping from the "crypto-types" module.



## A.9. 08 to 09

- \* Addressed comments raised by YANG Doctor in the ct/ts/ks drafts.

## A.10. 09 to 10

- \* Updated Abstract and Intro to address comments by Tom Petch.
- \* Removed the "tcp-connection-grouping" grouping (now models use the "tcp-common-grouping" directly).
- \* Added XML-comment above examples explaining the reason for the unusual top-most element's presence.
- \* Added Security Considerations section for the "local-binding-supported" feature.
- \* Replaced some hardcoded refs to <xref> elements.
- \* Fixed nits found by YANG Doctor reviews.
- \* Aligned modules with 'pyang -f' formatting.
- \* Added an "Acknowledgements" section.

## A.11. 10 to 11

- \* Replaced "base64encodedvalue==" with "BASE64VALUE=" in examples.
- \* Minor editorial nits

## A.12. 11 to 12

- \* Fixed up the 'WG Web' and 'WG List' lines in YANG module(s)
- \* Fixed up copyright (i.e., s/Simplified/Revised/) in YANG module(s)

## Acknowledgements

The authors would like to thank for following for lively discussions on list and in the halls (ordered by first name): Juergen Schoenwaelder, Ladislav Lhotka, Nick Hancock, and Tom Petch.

## Authors' Addresses

Kent Watsen  
Watsen Networks  
Email: kent+ietf@watsen.net

Michael Scharf  
Hochschule Esslingen - University of Applied Sciences  
Email: michael.scharf@hs-esslingen.de

TCPM WG  
Internet Draft  
Intended status: Informational  
Obsoletes: 2140  
Expires: October 2021

J. Touch  
Independent  
M. Welzl  
S. Islam  
University of Oslo  
April 12, 2021

TCP Control Block Interdependence  
draft-ietf-tcpm-2140bis-11.txt

Status of this Memo

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

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.

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

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

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

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

This Internet-Draft will expire on October 12, 2021.

## Copyright Notice

Copyright (c) 2021 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.

## Abstract

This memo provides guidance to TCP implementers that is intended to help improve connection convergence to steady-state operation without affecting interoperability. It updates and replaces RFC 2140's description of sharing TCP state, as typically represented in TCP Control Blocks, among similar concurrent or consecutive connections.

## Table of Contents

1. Introduction.....	3
2. Conventions Used in This Document.....	4
3. Terminology.....	4
4. The TCP Control Block (TCB).....	5
5. TCB Interdependence.....	7
6. Temporal Sharing.....	7
6.1. Initialization of a new TCB.....	7
6.2. Updates to the TCB cache.....	8
6.3. Discussion.....	10
7. Ensemble Sharing.....	11
7.1. Initialization of a new TCB.....	11
7.2. Updates to the TCB cache.....	12
7.3. Discussion.....	13
8. Issues with TCB information sharing.....	14
8.1. Traversing the same network path.....	15
8.2. State dependence.....	15
8.3. Problems with sharing based on IP address.....	16
9. Implications.....	16
9.1. Layering.....	17
9.2. Other possibilities.....	17

10. Implementation Observations.....	18
11. Changes Compared to RFC 2140.....	19
12. Security Considerations.....	19
13. IANA Considerations.....	20
14. References.....	20
14.1. Normative References.....	20
14.2. Informative References.....	21
15. Acknowledgments.....	24
16. Change log.....	24
Appendix A : TCB Sharing History.....	28
Appendix B : TCP Option Sharing and Caching.....	29
Appendix C : Automating the Initial Window in TCP over Long Timescales.....	31
C.1. Introduction.....	31
C.2. Design Considerations.....	31
C.3. Proposed IW Algorithm.....	32
C.4. Discussion.....	36
C.5. Observations.....	37

## 1. Introduction

TCP is a connection-oriented reliable transport protocol layered over IP [RFC793]. Each TCP connection maintains state, usually in a data structure called the TCP Control Block (TCB). The TCB contains information about the connection state, its associated local process, and feedback parameters about the connection's transmission properties. As originally specified and usually implemented, most TCB information is maintained on a per-connection basis. Some implementations share certain TCB information across connections to the same host [RFC2140]. Such sharing is intended to lead to better overall transient performance, especially for numerous short-lived and simultaneous connections, as can be used in the World-Wide Web and other applications [Be94][Br02]. This sharing of state is intended to help TCP connections converge to long term behavior (assuming stable application load, i.e., so-called "steady-state") more quickly without affecting TCP interoperability.

This document updates RFC 2140's discussion of TCB state sharing and provides a complete replacement for that document. This state sharing affects only TCB initialization [RFC2140] and thus has no effect on the long-term behavior of TCP after a connection has been established nor on interoperability. Path information shared across SYN destination port numbers assumes that TCP segments having the same host-pair experience the same path properties, i.e., that traffic is not routed differently based on port numbers or other connection parameters (also addressed further in Section 8.1). The observations about TCB sharing in this document apply similarly to

any protocol with congestion state, including SCTP [RFC4960] and DCCP [RFC4340], as well as for individual subflows in Multipath TCP [RFC8684].

## 2. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The core of this document describes behavior that is already permitted by TCP standards. As a result, it provides informative guidance but does not use normative language, except when quoting other documents. Normative language is used in Appendix C as examples of requirements for future consideration.

## 3. Terminology

The following terminology is used frequently in this document. Items preceded with a "+" may be part of the state maintained as TCP connection state in the associated connections TCB and are the focus of sharing as described in this document. Note that terms are used as originally introduced where possible; in some cases, direction is indicated with a suffix (\_S for send, \_R for receive) and in other cases spelled out (sendcwnd).

+cwnd - TCP congestion window size [RFC5681]

host - a source or sink of TCP segments associated with a single IP address

host-pair - a pair of hosts and their corresponding IP addresses

+MMS\_R - maximum message size that can be received, the largest received transport payload of an IP datagram [RFC1122]

+MMS\_S - maximum message size that can be sent, the largest transmitted transport payload of an IP datagram [RFC1122]

path - an Internet path between the IP addresses of two hosts

PCB - protocol control block, the data associated with a protocol as maintained by an endpoint; a TCP PCB is called a TCB

PLPMTUD - packetization-layer path MTU discovery, a mechanism that uses transport packets to discover the PMTU [RFC4821]

+PMTU - largest IP datagram that can traverse a path [RFC1191][RFC8201]

PMTUD - path-layer MTU discovery, a mechanism that relies on ICMP error messages to discover the PMTU [RFC1191][RFC8201]

+RTT - round-trip time of a TCP packet exchange [RFC793]

+RTTVAR - variation of round-trip times of a TCP packet exchange [RFC6298]

+rwnd - TCP receive window size [RFC5681]

+sendcwnd - TCP send-side congestion window (cwnd) size [RFC5681]

+sendMSS - TCP maximum segment size, a value transmitted in a TCP option that represents the largest TCP user data payload that can be received [RFC6691]

+sssthresh - TCP slow-start threshold [RFC5681]

TCB - TCP Control Block, the data associated with a TCP connection as maintained by an endpoint

TCP-AO - TCP Authentication Option [RFC5925]

TFO - TCP Fast Open option [RFC7413]

+TFO\_cookie - TCP Fast Open cookie, state that is used as part of the TFO mechanism, when TFO is supported [RFC7413]

+TFO\_failure - an indication of when TFO option negotiation failed, when TFO is supported

+TFOinfo - information cached when a TFO connection is established, which includes the TFO\_cookie [RFC7413]

#### 4. The TCP Control Block (TCB)

A TCB describes the data associated with each connection, i.e., with each association of a pair of applications across the network. The TCB contains at least the following information [RFC793]:

- Local process state
  - pointers to send and receive buffers
  - pointers to retransmission queue and current segment
  - pointers to Internet Protocol (IP) PCB
- Per-connection shared state
  - macro-state
    - connection state
    - timers
    - flags
    - local and remote host numbers and ports
    - TCP option state
  - micro-state
    - send and receive window state (size\*, current number)
    - congestion window size (sendcwnd)\*
    - congestion window size threshold (sssthresh)\*
    - max window size seen\*
    - sendMSS#
    - MMS\_S#
    - MMS\_R#
    - PMTU#
    - round-trip time and its variation#

The per-connection information is shown as split into macro-state and micro-state, terminology borrowed from [Co91]. Macro-state describes the protocol for establishing the initial shared state about the connection; we include the endpoint numbers and components (timers, flags) required upon commencement that are later used to help maintain that state. Micro-state describes the protocol after a connection has been established, to maintain the reliability and congestion control of the data transferred in the connection.

We distinguish two other classes of shared micro-state that are associated more with host-pairs than with application pairs. One class is clearly host-pair dependent (shown above as "#", e.g., sendMSS, MMS\_R, MMS\_S, PMTU, RTT), because these parameters are defined by the endpoint or endpoint pair (sendMSS, MMS\_R, MMS\_S, RTT) or are already cached and shared on that basis (PMTU [RFC1191][RFC4821]). The other is host-pair dependent in its aggregate (shown above as "\*", e.g., congestion window information, current window sizes, etc.) because they depend on the total capacity between the two endpoints.

Not all of the TCB state is necessarily sharable. In particular, some TCP options are negotiated only upon request by the application layer, so their use may not be correlated across connections. Other options negotiate connection-specific parameters, which are similarly not shareable. These are discussed further in Appendix B.



Finally, we exclude `rwnd` from further discussion because its value should depend on the send window size, so it is already addressed by send window sharing and is not independently affected by sharing.

## 5. TCB Interdependence

There are two cases of TCB interdependence. Temporal sharing occurs when the TCB of an earlier (now CLOSED) connection to a host is used to initialize some parameters of a new connection to that same host, i.e., in sequence. Ensemble sharing occurs when a currently active connection to a host is used to initialize another (concurrent) connection to that host.

## 6. Temporal Sharing

The TCB data cache is accessed in two ways: it is read to initialize new TCBs and written when more current per-host state is available.

### 6.1. Initialization of a new TCB

TCBs for new connections can be initialized using cached context from past connections as follows:

## TEMPORAL SHARING - TCB Initialization

Cached TCB	New TCB
old_MMS_S	old_MMS_S or not cached*
old_MMS_R	old_MMS_R or not cached*
old_sendMSS	old_sendMSS
old_PMTU	old_PMTU+
old_RTT	old_RTT
old_RTTVAR	old_RTTVAR
old_option	(option specific)
old_ssthresh	old_ssthresh
old_sendcwnd	old_sendcwnd

+Note that PMTU is cached at the IP layer [RFC1191][RFC4821].

\*Note that some values are not cached when they are computed locally (MMS\_R) or indicated in the connection itself (MMS\_S in the SYN).

The table below gives an overview of option-specific information that can be shared. Additional information on some specific TCP options and sharing is provided in Appendix B.

## TEMPORAL SHARING - Option Info Initialization

Cached	New
old_TFO_cookie	old_TFO_cookie
old_TFO_failure	old_TFO_failure

## 6.2. Updates to the TCB cache

During a connection, the TCB cache can be updated based on events of current connections and their TCBS as they progress over time, as shown below:

## TEMPORAL SHARING - Cache Updates

Cached TCB	Current TCB	when?	New Cached TCB
old_MMS_S	curr_MMS_S	OPEN	curr_MMS_S
old_MMS_R	curr_MMS_R	OPEN	curr_MMS_R
old_sendMSS	curr_sendMSS	MSSopt	curr_sendMSS
old_PMTU	curr_PMTU	PMTUD+ / PLPMTUD+	curr_PMTU
old_RTT	curr_RTT	CLOSE	merge(curr,old)
old_RTTVAR	curr_RTTVAR	CLOSE	merge(curr,old)
old_option	curr_option	ESTAB	(depends on option)
old_ssthresh	curr_ssthresh	CLOSE	merge(curr,old)
old_sendcwnd	curr_sendcwnd	CLOSE	merge(curr,old)

+Note that PMTU is cached at the IP layer [RFC1191][RFC4821].

Merge() is the function that combines the current and previous (old) values and may vary for each parameter of the TCB cache. The particular function is not specified in this document; examples include windowed averages (mean of the past N values, for some N) and exponential decay ( $\text{new} = (1-\alpha) \cdot \text{old} + \alpha \cdot \text{new}$ , where  $\alpha$  is in the range [0..1]).

The table below gives an overview of option-specific information that can be similarly shared. The TFO cookie is maintained until the client explicitly requests it be updated as a separate event.

## TEMPORAL SHARING - Option Info Updates

Cached	Current	when?	New Cached
old_TFO_cookie	old_TFO_cookie	ESTAB	old_TFO_cookie
old_TFO_failure	old_TFO_failure	ESTAB	old_TFO_failure

### 6.3. Discussion

As noted, there is no particular benefit to caching MMS\_S and MMS\_R as these are reported by the local IP stack. Caching sendMSS and PMTU is trivial; reported values are cached (PMTU at the IP layer), and the most recent values are used. The cache is updated when the MSS option is received in a SYN or after PMTUD (i.e., when an ICMPv4 Fragmentation Needed [RFC1191] or ICMPv6 Packet Too Big message is received [RFC8201] or the equivalent is inferred, e.g., as from PLPMTUD [RFC4821]), respectively, so the cache always has the most recent values from any connection. For sendMSS, the cache is consulted only at connection establishment and not otherwise updated, which means that MSS options do not affect current connections. The default sendMSS is never saved; only reported MSS values update the cache, so an explicit override is required to reduce the sendMSS. Cached sendMSS affects only data sent in the SYN segment, i.e., during client connection initiation or during simultaneous open; all other segment MSS are based on the value updated as included in the SYN.

RTT values are updated by formulae that merges the old and new values, as noted in Section 6.2. Dynamic RTT estimation requires a sequence of RTT measurements. As a result, the cached RTT (and its variation) is an average of its previous value with the contents of the currently active TCB for that host, when a TCB is closed. RTT values are updated only when a connection is closed. The method for merging old and current values needs to attempt to reduce the transient effects of the new connections.

The updates for RTT, RTTVAR and ssthresh rely on existing information, i.e., old values. Should no such values exist, the current values are cached instead.

TCP options are copied or merged depending on the details of each option. E.g., TFO state is updated when a connection is established and read before establishing a new connection.

Sections 8 and 9 discuss compatibility issues and implications of sharing the specific information listed above. Section 10 gives an overview of known implementations.

Most cached TCB values are updated when a connection closes. The exceptions are MMS\_R and MMS\_S, which are reported by IP [RFC1122], PMTU which is updated after Path MTU Discovery and also reported by IP [RFC1191][RFC4821][RFC8201], and sendMSS, which is updated if the MSS option is received in the TCP SYN header.

Sharing sendMSS information affects only data in the SYN of the next connection, because sendMSS information is typically included in most TCP SYN segments. Caching PMTU can accelerate the efficiency of PMTUD but can also result in black-holing until corrected if in error. Caching MSS\_R and MSS\_S may be of little direct value as they are reported by the local IP stack anyway.

The way in which other TCP option state can be shared depends on the details of that option. E.g., TFO state includes the TCP Fast Open Cookie [RFC7413] or, in case TFO fails, a negative TCP Fast Open response. RFC 7413 states, "The client MUST cache negative responses from the server in order to avoid potential connection failures. Negative responses include the server not acknowledging the data in the SYN, ICMP error messages, and (most importantly) no response (SYN-ACK) from the server at all, i.e., connection timeout." [RFC 7413]. TFOinfo is cached when a connection is established.

Other TCP option state might not be as readily cached. E.g., TCP-AO [RFC5925] success or failure between a host pair for a single SYN destination port might be usefully cached. TCP-AO success or failure to other SYN destination ports on that host pair is never useful to cache because TCP-AO security parameters can vary per service.

## 7. Ensemble Sharing

Sharing cached TCB data across concurrent connections requires attention to the aggregate nature of some of the shared state. For example, although MSS and RTT values can be shared by copying, it may not be appropriate to simply copy congestion window or ssthresh information; instead, the new values can be a function (f) of the cumulative values and the number of connections (N).

### 7.1. Initialization of a new TCB

TCBs for new connections can be initialized using cached context from concurrent connections as follows:

## ENSEMBLE SHARING - TCB Initialization

Cached TCB	New TCB
old_MMS_S	old_MMS_S
old_MMS_R	old_MMS_R
old_sendMSS	old_sendMSS
old_PMTU	old_PMTU+
old_RTT	old_RTT
old_RTTVAR	old_RTTVAR
sum(old_ssthresh)	f(sum(old_ssthresh), N)
sum(old_sendcwnd)	f(sum(old_sendcwnd), N)
old_option	(option specific)

+Note that PMTU is cached at the IP layer [RFC1191][RFC4821].

In the table, the cached sum() is a total across all active connections because these parameters act in aggregate; similarly f() is a function that updates that sum based on the new connection's values, represented as "N".

The table below gives an overview of option-specific information that can be similarly shared. Again, The TFO\_cookie is updated upon explicit client request, which is a separate event.

## ENSEMBLE SHARING - Option Info Initialization

Cached	New
old_TFO_cookie	old_TFO_cookie
old_TFO_failure	old_TFO_failure

## 7.2. Updates to the TCB cache

During a connection, the TCB cache can be updated based on changes to concurrent connections and their TCBs, as shown below:

## ENSEMBLE SHARING - Cache Updates

Cached TCB	Current TCB	when?	New Cached TCB
old_MMS_S	curr_MMS_S	OPEN	curr_MMS_S
old_MMS_R	curr_MMS_R	OPEN	curr_MMS_R
old_sendMSS	curr_sendMSS	MSSopt	curr_sendMSS
old_PMTU	curr_PMTU	PMTUD+ / PLPMTUD+	curr_PMTU
old_RTT	curr_RTT	update	rtt_update(old, curr)
old_RTTVAR	curr_RTTVAR	update	rtt_update(old, curr)
old_ssthresh	curr_ssthresh	update	adjust sum as appropriate
old_sendcwnd	curr_sendcwnd	update	adjust sum as appropriate
old_option	curr_option	(depends)	(option specific)

+Note that the PMTU is cached at the IP layer [RFC1191][RFC4821].

In the table, rtt\_update() is the function used to combine old and current values, e.g., as a windowed average or exponentially decayed average.

The table below gives an overview of option-specific information that can be similarly shared.

## ENSEMBLE SHARING - Option Info Updates

Cached	Current	when?	New Cached
old_TFO_cookie	old_TFO_cookie	ESTAB	old_TFO_cookie
old_TFO_failure	old_TFO_failure	ESTAB	old_TFO_failure

## 7.3. Discussion

For ensemble sharing, TCB information should be cached as early as possible, sometimes before a connection is closed. Otherwise, opening multiple concurrent connections may not result in TCB data sharing if no connection closes before others open. The amount of work involved in updating the aggregate average should be minimized,

but the resulting value should be equivalent to having all values measured within a single connection. The function "rtt\_update" in the ensemble sharing table indicates this operation, which occurs whenever the RTT would have been updated in the individual TCP connection. As a result, the cache contains the shared RTT variables, which no longer need to reside in the TCB.

Congestion window size and ssthresh aggregation are more complicated in the concurrent case. When there is an ensemble of connections, we need to decide how that ensemble would have shared these variables, in order to derive initial values for new TCBs.

Sections 8 and 9 discuss compatibility issues and implications of sharing the specific information listed above.

There are several ways to initialize the congestion window in a new TCB among an ensemble of current connections to a host. Current TCP implementations initialize it to four segments as standard [RFC3390] and 10 segments experimentally [RFC6928]. These approaches assume that new connections should behave as conservatively as possible. The algorithm described in [Bal2] adjusts the initial cwnd depending on the cwnd values of ongoing connections. It is also possible to use sharing mechanisms over long timescales to adapt TCP's initial window automatically, as described further in Appendix C.

## 8. Issues with TCB information sharing

Here, we discuss various types of problems that may arise with TCB information sharing.

For the congestion and current window information, the initial values computed by TCB interdependence may not be consistent with the long-term aggregate behavior of a set of concurrent connections between the same endpoints. Under conventional TCP congestion control, if the congestion window of a single existing connection has converged to 40 segments, two newly joining concurrent connections assume initial windows of 10 segments [RFC6928], and the current connection's window doesn't decrease to accommodate this additional load and connections can mutually interfere. One example of this is seen on low-bandwidth, high-delay links, where concurrent connections supporting Web traffic can collide because their initial windows were too large, even when set at one segment.

The authors of [Hul2] recommend caching ssthresh for temporal sharing only when flows are long. Some studies suggest that sharing ssthresh between short flows can deteriorate the performance of



individual connections [Hul2, Dul6], although this may benefit aggregate network performance.

#### 8.1. Traversing the same network path

TCP is sometimes used in situations where packets of the same host-pair do not always take the same path, such as when connection-specific parameters are used for routing (e.g., for load balancing). Multipath routing that relies on examining transport headers, such as ECMP and LAG [RFC7424], may not result in repeatable path selection when TCP segments are encapsulated, encrypted, or altered - for example, in some Virtual Private Network (VPN) tunnels that rely on proprietary encapsulation. Similarly, such approaches cannot operate deterministically when the TCP header is encrypted, e.g., when using IPsec ESP (although TCB interdependence among the entire set sharing the same endpoint IP addresses should work without problems when the TCP header is encrypted). Measures to increase the probability that connections use the same path could be applied: e.g., the connections could be given the same IPv6 flow label [RFC6437]. TCB interdependence can also be extended to sets of host IP address pairs that share the same network path conditions, such as when a group of addresses is on the same LAN (see Section 9).

Traversing the same path is not important for host-specific information such as `rwnd` and TCP option state, such as `TFOinfo`, or for information that is already cached per-host, such as path MTU. When TCB information is shared across different SYN destination ports, path-related information can be incorrect; however, the impact of this error is potentially diminished if (as discussed here) TCB sharing affects only the transient event of a connection start or if TCB information is shared only within connections to the same SYN destination port.

In case of Temporal Sharing, TCB information could also become invalid over time, i.e., indicating that although the path remains the same, path properties have changed. Because this is similar to the case when a connection becomes idle, mechanisms that address idle TCP connections (e.g., [RFC7661]) could also be applied to TCB cache management, especially when TCP Fast Open is used [RFC7413].

#### 8.2. State dependence

There may be additional considerations to the way in which TCB interdependence rebalances congestion feedback among the current connections, e.g., it may be appropriate to consider the impact of a connection being in Fast Recovery [RFC5681] or some other similar

unusual feedback state, e.g., as inhibiting or affecting the calculations described herein.

### 8.3. Problems with sharing based on IP address

It can be wrong to share TCB information between TCP connections on the same host as identified by the IP address if an IP address is assigned to a new host (e.g., IP address spinning, as is used by ISPs to inhibit running servers). It can be wrong if Network Address (and Port) Translation (NA(P)T) [RFC2663] or any other IP sharing mechanism is used. Such mechanisms are less likely to be used with IPv6. Other methods to identify a host could also be considered to make correct TCB sharing more likely. Moreover, some TCB information is about dominant path properties rather than the specific host. IP addresses may differ, yet the relevant part of the path may be the same.

## 9. Implications

There are several implications to incorporating TCB interdependence in TCP implementations. First, it may reduce the need for application-layer multiplexing for performance enhancement [RFC7231]. Protocols like HTTP/2 [RFC7540] avoid connection reestablishment costs by serializing or multiplexing a set of per-host connections across a single TCP connection. This avoids TCP's per-connection OPEN handshake and also avoids recomputing the MSS, RTT, and congestion window values. By avoiding the so-called "slow-start restart", performance can be optimized [Hu01]. TCB interdependence can provide the "slow-start restart avoidance" of multiplexing, without requiring a multiplexing mechanism at the application layer.

Like the initial version of this document [RFC2140], this update's approach to TCB interdependence focuses on sharing a set of TCBs by updating the TCB state to reduce the impact of transients when connections begin, end, or otherwise significantly change state. Other mechanisms have since been proposed to continuously share information between all ongoing communication (including connectionless protocols), updating the congestion state during any congestion-related event (e.g., timeout, loss confirmation, etc.) [RFC3124]. By dealing exclusively with transients, the approach in this document is more likely to exhibit the "steady-state" behavior as unmodified, independent TCP connections.

### 9.1. Layering

TCB interdependence pushes some of the TCP implementation from the traditional transport layer (in the ISO model), to the network layer. This acknowledges that some state is in fact per-host-pair or can be per-path as indicated solely by that host-pair. Transport protocols typically manage per-application-pair associations (per stream), and network protocols manage per-host-pair and path associations (routing). Round-trip time, MSS, and congestion information could be more appropriately handled at the network layer, aggregated among concurrent connections, and shared across connection instances [RFC3124].

An earlier version of RTT sharing suggested implementing RTT state at the IP layer, rather than at the TCP layer. Our observations describe sharing state among TCP connections, which avoids some of the difficulties in an IP-layer solution. One such problem of an IP layer solution is determining the correspondence between packet exchanges using IP header information alone, where such correspondence is needed to compute RTT. Because TCB sharing computes RTTs inside the TCP layer using TCP header information, it can be implemented more directly and simply than at the IP layer. This is a case where information should be computed at the transport layer but could be shared at the network layer.

### 9.2. Other possibilities

Per-host-pair associations are not the limit of these techniques. It is possible that TCBs could be similarly shared between hosts on a subnet or within a cluster, because the predominant path can be subnet-subnet, rather than host-host. Additionally, TCB interdependence can be applied to any protocol with congestion state, including SCTP [RFC4960] and DCCP [RFC4340], as well as for individual subflows in Multipath TCP [RFC8684].

There may be other information that can be shared between concurrent connections. For example, knowing that another connection has just tried to expand its window size and failed, a connection may not attempt to do the same for some period. The idea is that existing TCP implementations infer the behavior of all competing connections, including those within the same host or subnet. One possible optimization is to make that implicit feedback explicit, via extended information associated with the endpoint IP address and its TCP implementation, rather than per-connection state in the TCB.

This document focuses on sharing TCB information at connection initialization. Subsequent to RFC 2140, there have been numerous

approaches that attempt to coordinate ongoing state across concurrent connections, both within TCP and other congestion-reactive protocols, which are summarized in [Is18]. These approaches are more complex to implement and their comparison to steady-state TCP equivalence can be more difficult to establish, sometimes intentionally (i.e., they sometimes intend to provide a different kind of "fairness" than emerges from TCP operation).

## 10. Implementation Observations

The observation that some TCB state is host-pair specific rather than application-pair dependent is not new and is a common engineering decision in layered protocol implementations. Although now deprecated, T/TCP [RFC1644] was the first to propose using caches in order to maintain TCB states (see Appendix A).

The table below describes the current implementation status for TCB temporal sharing in Windows as of December 2020, Apple variants (macOS, iOS, iPadOS, tvOS, watchOS) as of January 2021, Linux kernel version 5.10.3, and FreeBSD 12. Ensemble sharing is not yet implemented.

KNOWN IMPLEMENTATION STATUS

TCB data	Status
old_MMS_S	Not shared
old_MMS_R	Not shared
old_sendMSS	Cached and shared in Apple, Linux (MSS)
old_PMTU	Cached and shared in Apple, FreeBSD, Windows (PMTU)
old_RTT	Cached and shared in Apple, FreeBSD, Linux, Windows
old_RTTVAR	Cached and shared in Apple, FreeBSD, Windows
old_TFOinfo	Cached and shared in Apple, Linux, Windows
old_sendcwnd	Not shared
old_ssthresh	Cached and shared in Apple, FreeBSD*, Linux*
TFO failure	Cached and shared in Apple

In the table above, "Apple" refers to all Apple OSes, i.e., desktop/laptop macOS, phone iOS, pad iPadOS, video player tvOS, and watch watchOS, which all share the same Internet protocol stack.

\*Note: In FreeBSD, new ssthresh is the mean of curr\_ssthresh and previous value if a previous value exists; in Linux, the calculation depends on state and is  $\max(\text{curr\_cwnd}/2, \text{old\_ssthresh})$  in most cases.

## 11. Changes Compared to RFC 2140

This document updates the description of TCB sharing in RFC 2140 and its associated impact on existing and new connection state, providing a complete replacement for that document [RFC2140]. It clarifies the previous description and terminology and extends the mechanism to its impact on new protocols and mechanisms, including multipath TCP, fast open, PLPMTUD, NAT, and the TCP Authentication Option.

The detailed impact on TCB state addresses TCB parameters in greater detail, addressing MSS in both the send and receive direction, MSS and sendMSS separately, adds path MTU and ssthresh, and addresses the impact on TCP option state.

New sections have been added to address compatibility issues and implementation observations. The relation of this work to T/TCP has been moved to 0 on history, partly to reflect the deprecation of that protocol.

Appendix C has been added to discuss the potential to use temporal sharing over long timescales to adapt TCP's initial window automatically, avoiding the need to periodically revise a single global constant value.

Finally, this document updates and significantly expands the referenced literature.

## 12. Security Considerations

These presented implementation methods do not have additional ramifications for direct (connection-aborting or information injecting) attacks on individual connections. Individual connections, whether using sharing or not, also may be susceptible to denial-of-service attacks that reduce performance or completely deny connections and transfers if not otherwise secured.

TCB sharing may create additional denial-of-service attacks that affect the performance of other connections by polluting the cached information. This can occur across whatever set of connections where the TCB is shared, between connections in a single host, or between hosts if TCB sharing is implemented within a subnet (see Implications section). Some shared TCB parameters are used only to create new TCBs, others are shared among the TCBs of ongoing connections. New connections can join the ongoing set, e.g., to optimize send window size among a set of connections to the same host. PMTU is defined as shared at the IP layer, and is already susceptible in this way.

Options in client SYNs can be easier to forge than complete, two-way connections. As a result, their values may not be safely incorporated in shared values until after the three-way handshake completes.

Attacks on parameters used only for initialization affect only the transient performance of a TCP connection. For short connections, the performance ramification can approach that of a denial-of-service attack. E.g., if an application changes its TCB to have a false and small window size, subsequent connections will experience performance degradation until their window grew appropriately.

TCB sharing reuses and mixes information from past and current connections. Although reusing information could create a potential for fingerprinting to identify hosts, the mixing reduces that potential. There has been no evidence of fingerprinting based on this technique and it is currently considered safe in that regard. Further, information about the performance of a TCP connection has not been considered as private.

### 13. IANA Considerations

There are no IANA implications or requests in this document.

This section should be removed upon final publication as an RFC.

### 14. References

#### 14.1. Normative References

- [RFC793] Postel, J., "Transmission Control Protocol," Network Working Group RFC-793/STD-7, ISI, Sept. 1981.
- [RFC1122] Braden, R. (ed), "Requirements for Internet Hosts -- Communication Layers", RFC-1122, Oct. 1989.

- [RFC1191] Mogul, J., Deering, S., "Path MTU Discovery," RFC 1191, Nov. 1990.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4821] Mathis, M., Heffner, J., "Packetization Layer Path MTU Discovery," RFC 4821, Mar. 2007.
- [RFC5681] Allman, M., Paxson, V., Blanton, E., "TCP Congestion Control," RFC 5681 (Standards Track), Sep. 2009.
- [RFC6298] Paxson, V., Allman, M., Chu, J., Sargent, M., "Computing TCP's Retransmission Timer," RFC 6298, June 2011.
- [RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., Jain, A., "TCP Fast Open", RFC 7413, Dec. 2014.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", RFC 8174, May 2017.
- [RFC8201] McCann, J., Deering, S., Mogul, J., Hinden, R. (Ed.), "Path MTU Discovery for IP version 6," RFC 8201, Jul. 2017.

#### 14.2. Informative References

- [Al10] Allman, M., "Initial Congestion Window Specification", (work in progress), draft-allman-tcpm-bump-initcwnd-00, Nov. 2010.
- [Ba12] Barik, R., Welzl, M., Ferlin, S., Alay, O., " LISA: A Linked Slow-Start Algorithm for MPTCP", IEEE ICC, Kuala Lumpur, Malaysia, May 23-27 2016.
- [Ba20] Bagnulo, M., Briscoe, B., "ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control Packets", draft-ietf-tcpm-generalized-ecn-07, Feb. 2021.
- [Be94] Berners-Lee, T., et al., "The World-Wide Web," Communications of the ACM, V37, Aug. 1994, pp. 76-82.
- [Br94] Braden, B., "T/TCP -- Transaction TCP: Source Changes for Sun OS 4.1.3," Release 1.0, USC/ISI, September 14, 1994.

- [Br02] Brownlee, N., Claffy, K., "Understanding Internet Traffic Streams: Dragonflies and Tortoises", IEEE Communications Magazine p110-117, 2002.
- [Co91] Comer, D., Stevens, D., Internetworking with TCP/IP, V2, Prentice-Hall, NJ, 1991.
- [Du16] Dukkkipati, N., Yuchung C., Amin V., "Research Impacting the Practice of Congestion Control." ACM SIGCOMM CCR (editorial), on-line post, July 2016.
- [FreeBSD] FreeBSD source code, Release 2.10, <http://www.freebsd.org/>
- [Hu01] Hughes, A., Touch, J., Heidemann, J., "Issues in Slow-Start Restart After Idle", draft-hughes-restart-00 (expired), Dec. 2001.
- [Hul2] Hurtig, P., Brunstrom, A., "Enhanced metric caching for short TCP flows," 2012 IEEE International Conference on Communications (ICC), Ottawa, ON, 2012, pp. 1209-1213.
- [IANA] IANA TCP Parameters (options) registry, <https://www.iana.org/assignments/tcp-parameters>
- [Is18] Islam, S., Welzl, M., Hiorth, K., Hayes, D., Armitage, G., Gjessing, S., "ctrlTCP: Reducing Latency through Coupled, Heterogeneous Multi-Flow TCP Congestion Control," Proc. IEEE INFOCOM Global Internet Symposium (GI) workshop (GI 2018), Honolulu, HI, April 2018.
- [Ja88] Jacobson, V., Karels, M., "Congestion Avoidance and Control", Proc. Sigcomm 1988.
- [RFC1644] Braden, R., "T/TCP -- TCP Extensions for Transactions Functional Specification," RFC-1644, July 1994.
- [RFC1379] Braden, R., "Transaction TCP -- Concepts," RFC-1379, September 1992.
- [RFC2001] Stevens, W., "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC2001 (Standards Track), Jan. 1997.
- [RFC2140] Touch, J., "TCP Control Block Interdependence", RFC 2140, April 1997.



- [RFC2414] Allman, M., Floyd, S., Partridge, C., "Increasing TCP's Initial Window", RFC 2414 (Experimental), Sept. 1998.
- [RFC2663] Srisuresh, P., Holdrege, M., "IP Network Address Translator (NAT) Terminology and Considerations", RFC-2663, August 1999.
- [RFC3390] Allman, M., Floyd, S., Partridge, C., "Increasing TCP's Initial Window," RFC 3390, Oct. 2002.
- [RFC3124] Balakrishnan, H., Seshan, S., "The Congestion Manager," RFC 3124, June 2001.
- [RFC4340] Kohler, E., Handley, M., Floyd, S., "Datagram Congestion Control Protocol (DCCP)," RFC 4340, Mar. 2006.
- [RFC4960] Stewart, R., (Ed.), "Stream Control Transmission Protocol," RFC4960, Sept. 2007.
- [RFC5925] Touch, J., Mankin, A., Bonica, R., "The TCP Authentication Option," RFC 5925, June 2010.
- [RFC6437] Amante, S., Carpenter, B., Jiang, S., Rajajalme, J., "IPv6 Flow Label Specification," RFC 6437, Nov. 2011.
- [RFC6691] Borman, D., "TCP Options and Maximum Segment Size (MSS)," RFC 6691, July 2012.
- [RFC6928] Chu, J., Dukkkipati, N., Cheng, Y., Mathis, M., "Increasing TCP's Initial Window," RFC 6928, Apr. 2013.
- [RFC7231] Fielding, R., Reshke, J., Eds., "HTTP/1.1 Semantics and Content," RFC-7231, June 2014.
- [RFC7323] Borman, D., Braden, B., Jacobson, V., Scheffenegger, R., (Ed.), "TCP Extensions for High Performance," RFC 7323, Sept. 2014.
- [RFC7424] Krishnan, R., Yong, L., Ghanwani, A., So, N., Khasnabish, B., "Mechanisms for Optimizing Link Aggregation Group (LAG) and Equal-Cost Multipath (ECMP) Component Link Utilization in Networks", RFC 7424, Jan. 2015
- [RFC7540] Belshé, M., Peon, R., Thomson, M., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, May 2015.

[RFC7661] Fairhurst, G., Sathiaselalan, A., Secchi, R., "Updating TCP to Support Rate-Limited Traffic", RFC 7661, Oct. 2015.

[RFC8684] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., Paasch, C., "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 8684, Mar. 2020.

## 15. Acknowledgments

The authors would like to thank for Praveen Balasubramanian for information regarding TCB sharing in Windows, Christoph Paasch for information regarding TCB sharing in Apple OSes, and Yuchung Cheng, Lars Eggert, Ilpo Jarvinen and Michael Scharf for comments on earlier versions of the draft, as well as members of the TCPM WG. Earlier revisions of this work received funding from a collaborative research project between the University of Oslo and Huawei Technologies Co., Ltd. and were partly supported by USC/ISI's Postel Center.

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

## 16. Change log

This section should be removed upon final publication as an RFC.

ietf-11:

- Addressed gen-art review and IESG feedback

ietf-10:

- Addressed IETF last call feedback

ietf-09:

- Correction of typographic errors

ietf-08:

- Address TSV AD comments, add Apple OS implementation status

ietf-07:

- Update per id-nits and normative language for consistency

ietf-06:

- Address WGLC comments

ietf-05:

- Correction of typographic errors, expansion of terminology

ietf-04:

- Fix internal cross-reference errors that appeared in ietf-02
- Updated tables to re-center; clarified text

ietf-03:

- Correction of typographic errors, minor rewording in appendices

ietf-02:

- Minor reorganization and correction of typographic errors
- Added text to address fingerprinting in Security section
- Now retains Appendix B and body option tables upon publication

ietf-01:

- Added Appendix C to address long-timescale temporal adaptation

ietf-00:

- Re-issued as draft-ietf-tcpm-2140bis due to WG adoption.
- Cleaned orphan references to T/TCP, removed incomplete refs
- Moved references to informative section and updated Sec 2
- Updated to clarify no impact to interoperability
- Updated appendix B to avoid 2119 language

06:

- Changed to update 2140, cite it normatively, and summarize the updates in a separate section

05:

- Fixed some TBDs

04:

- Removed BCP-style recommendations and fixed some TBDs

03:

- Updated Touch's affiliation and address information

02:

- Stated that our OS implementation overview table only covers temporal sharing.
- Correctly reflected sharing of old\_RTT in Linux in the implementation overview table.
- Marked entries that are considered safe to share with an asterisk (suggestion was to split the table)
- Discussed correct host identification: NATs may make IP addresses the wrong input, could e.g., use HTTP cookie.
- Included MMS\_S and MMS\_R from RFC1122; fixed the use of MSS and MTU
- Added information about option sharing, listed options in Appendix B

#### Authors' Addresses

Joe Touch  
Manhattan Beach, CA 90266  
USA

Phone: +1 (310) 560-0334  
Email: touch@strayalpha.com

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

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

Safiqul Islam  
University of Oslo  
PO Box 1080 Blindern  
Oslo N-0316  
Norway

Phone: +47 22 84 08 37  
Email: safiquli@ifi.uio.no

## Appendix A: TCB Sharing History

T/TCP proposed using caches to maintain TCB information across instances (temporal sharing), e.g., smoothed RTT, RTT variation, congestion avoidance threshold, and MSS [RFC1644]. These values were in addition to connection counts used by T/TCP to accelerate data delivery prior to the full three-way handshake during an OPEN. The goal was to aggregate TCB components where they reflect one association - that of the host-pair, rather than artificially separating those components by connection.

At least one T/TCP implementation saved the MSS and aggregated the RTT parameters across multiple connections but omitted caching the congestion window information [Br94], as originally specified in [RFC1379]. Some T/TCP implementations immediately updated MSS when the TCP MSS header option was received [Br94], although this was not addressed specifically in the concepts or functional specification [RFC1379][RFC1644]. In later T/TCP implementations, RTT values were updated only after a CLOSE, which does not benefit concurrent sessions.

Temporal sharing of cached TCB data was originally implemented in the SunOS 4.1.3 T/TCP extensions [Br94] and the FreeBSD port of same [FreeBSD]. As mentioned before, only the MSS and RTT parameters were cached, as originally specified in [RFC1379]. Later discussion of T/TCP suggested including congestion control parameters in this cache; for example, [RFC1644] (Section 3.1) hints at initializing the congestion window to the old window size.

## Appendix B: TCP Option Sharing and Caching

In addition to the options that can be cached and shared, this memo also lists known TCP options [IANA] for which state is unsafe to be kept. This list is not intended to be authoritative or exhaustive.

Obsolete (unsafe to keep state):

ECHO

ECHO REPLY

PO Conn permitted

PO service profile

CC

CC.NEW

CC.ECHO

Alt CS req

Alt CS data

No state to keep:

EOL

NOP

WS

SACK

TS

MD5

TCP-AO

EXP1

EXP2

Unsafe to keep state:

Skeeter (DH exchange, known to be vulnerable)

Bubba (DH exchange, known to be vulnerable)

Trailer CS

SCPS capabilities

S-NACK

Records boundaries

Corruption experienced

SNAP

TCP Compression

Quickstart response

UTO

MPTCP negotiation success (see below for negotiation failure)

TFO negotiation success (see below for negotiation failure)

Safe but optional to keep state:

MPTCP negotiation failure (to avoid negotiation retries)

MSS

TFO negotiation failure (to avoid negotiation retries)

Safe and necessary to keep state:

TFO cookie (if TFO succeeded in the past)



## Appendix C: Automating the Initial Window in TCP over Long Timescales

## C.1. Introduction

Temporal sharing, as described earlier in this document, builds on the assumption that multiple consecutive connections between the same host pair are somewhat likely to be exposed to similar environment characteristics. The stored information can become less accurate over time and suitable precautions should take this ageing into consideration (this is discussed further in section 8.1). However, there are also cases where it can make sense to track these values over longer periods, observing properties of TCP connections to gradually influence evolving trends in TCP parameters. This appendix describes an example of such a case.

TCP's congestion control algorithm uses an initial window value (IW), both as a starting point for new connections and as an upper limit for restarting after an idle period [RFC5681][RFC7661]. This value has evolved over time, originally one maximum segment size (MSS), and increased to the lesser of four MSS or 4,380 bytes [RFC3390][RFC5681]. For a typical Internet connection with a maximum transmission unit (MTU) of 1500 bytes, this permits three segments of 1,460 bytes each.

The IW value was originally implied in the original TCP congestion control description and documented as a standard in 1997 [RFC2001][Ja88]. The value was updated in 1998 experimentally and moved to the standards track in 2002 [RFC2414][RFC3390]. In 2013, it was experimentally increased to 10 [RFC6928].

This appendix discusses how TCP can objectively measure when an IW is too large, and that such feedback should be used over long timescales to adjust the IW automatically. The result should be safer to deploy and might avoid the need to repeatedly revisit IW over time.

Note that this mechanism attempts to make the IW more adaptive over time. It can increase the IW beyond that which is currently recommended for widescale deployment, and so its use should be carefully monitored.

## C.2. Design Considerations

TCP's IW value has existed statically for over two decades, so any solution to adjusting the IW dynamically should have similarly stable, non-invasive effects on the performance and complexity of TCP. In order to be fair, the IW should be similar for most machines

on the public Internet. Finally, a desirable goal is to develop a self-correcting algorithm, so that IW values that cause network problems can be avoided. To that end, we propose the following design goals:

- o Impart little to no impact to TCP in the absence of loss, i.e., it should not increase the complexity of default packet processing in the normal case.
- o Adapt to network feedback over long timescales, avoiding values that persistently cause network problems.
- o Decrease the IW in the presence of sustained loss of IW segments, as determined over a number of different connections.
- o Increase the IW in the absence of sustained loss of IW segments, as determined over a number of different connections.
- o Operate conservatively, i.e., tend towards leaving the IW the same in the absence of sufficient information, and give greater consideration to IW segment loss than IW segment success.

We expect that, without other context, a good IW algorithm will converge to a single value, but this is not required. An endpoint with additional context or information, or deployed in a constrained environment, can always use a different value. In particular, information from previous connections, or sets of connections with a similar path, can already be used as context for such decisions (as noted in the core of this document).

However, if a given IW value persistently causes packet loss during the initial burst of packets, it is clearly inappropriate and could be inducing unnecessary loss in other competing connections. This might happen for sites behind very slow boxes with small buffers, which may or may not be the first hop.

### C.3. Proposed IW Algorithm

Below is a simple description of the proposed IW algorithm. It relies on the following parameters:

- o MinIW = 3 MSS or 4,380 bytes (as per [RFC3390])
- o MaxIW = 10 MSS (as per [RFC6928])
- o MulDecr = 0.5

- o AddIncr = 2 MSS
- o Threshold = 0.05

We assume that the minimum IW (MinIW) should be as currently specified as standard [RFC3390]. The maximum IW can be set to a fixed value (we suggest using the experimental and now somewhat de-facto standard in [RFC6928]) or set based on a schedule if trusted time references are available [Al10]; here we prefer a fixed value. We also propose to use an AIMD algorithm, with increase and decreases as noted.

Although these parameters are somewhat arbitrary, their initial values are not important except that the algorithm is AIMD and the MaxIW should not exceed that recommended for other systems on the Internet (here we selected the current de-facto standard rather than the actual standard). Current proposals, including default current operation, are degenerate cases of the algorithm below for given parameters - notably MulDec = 1.0 and AddIncr = 0 MSS, thus disabling the automatic part of the algorithm.

The proposed algorithm is as follows:

1. On boot:

```
IW = MaxIW; # assume this is in bytes, and indicates an integer
multiple of 2 MSS (an even number to support ACK compression)
```

2. Upon starting a new connection:

```
CWND = IW;
conncount++;
IWnotchecked = 1; # true
```

3. During a connection's SYN-ACK processing, if SYN-ACK includes ECN (as similarly addressed in Sec 5 of ECN++ for TCP [Ba20]), treat as if the IW is too large:

```
if (IWnotchecked && (synackecn == 1)) {
    losscount++;
    IWnotchecked = 0; # never check again
}
```

4. During a connection, if retransmission occurs, check the seqno of the outgoing packet (in bytes) to see if the resent segment fixes an IW loss:

```
if (Retransmitting && IWnotchecked && ((seqno - ISN) < IW)) {  
    losscount++;  
    IWnotchecked = 0; # never do this entire "if" again  
} else {  
    IWnotchecked = 0; # you're beyond the IW so stop checking  
}
```

5. Once every 1000 connections, as a separate process (i.e., not as part of processing a given connection):

```
if (conncount > 1000) {  
    if (losscount/conncount > threshold) {  
        # the number of connections with errors is too high  
        IW = IW * MulDecr;  
    } else {  
        IW = IW + AddIncr;  
    }  
}
```

As presented, this algorithm can yield a false positive when the sequence number wraps around, e.g., the code might increment losscount in step 4 when no loss occurred or fail to increment losscount when a loss did occur. This can be avoided using either PAWS [RFC7323] context or internal extended sequence number representations (as in TCP-AO [RFC5925]). Alternately, false positives can be tolerated because they are expected to be infrequent and thus will not significantly impact the algorithm.

A number of additional constraints need to be imposed if this mechanism is implemented to ensure that it defaults to values that comply with current Internet standards, is conservative in how it extends those values, and returns to those values in the absence of positive feedback (i.e., success). To that end, we recommend the following list of example constraints:

>> The automatic IW algorithm MUST initialize MaxIW a value no larger than the currently recommended Internet default, in the absence of other context information.

Thus, if there are too few connections to make a decision or if there is otherwise insufficient information to increase the IW, then the MaxIW defaults to the current recommended value.

>> An implementation MAY allow the MaxIW to grow beyond the currently recommended Internet default, but not more than 2 segments per calendar year.

Thus, if an endpoint has a persistent history of successfully transmitting IW segments without loss, then it is allowed to probe the Internet to determine if larger IW values have similar success. This probing is limited and requires a trusted time source, otherwise the MaxIW remains constant.

>> An implementation MUST adjust the IW based on loss statistics at least once every 1000 connections.

An endpoint needs to be sufficiently reactive to IW loss.

>> An implementation MUST decrease the IW by at least one MSS when indicated during an evaluation interval.

An endpoint that detects loss needs to decrease its IW by at least one MSS, otherwise it is not participating in an automatic reactive algorithm.

>> An implementation MUST increase by no more than 2 MSS per evaluation interval.

An endpoint that does not experience IW loss needs to probe the network incrementally.

>> An implementation SHOULD use an IW that is an integer multiple of 2 MSS.

The IW should remain a multiple of 2 MSS segments, to enable efficient ACK compression without incurring unnecessary timeouts.

>> An implementation MUST decrease the IW if more than 95% of connections have IW losses.

Again, this is to ensure an implementation is sufficiently reactive.

>> An implementation MAY group IW values and statistics within subsets of connections. Such grouping MAY use any information about connections to form groups except loss statistics.

There are some TCP connections which might not be counted at all, such as those to/from loopback addresses, or those within the same subnet as that of a local interface (for which congestion control is sometimes disabled anyway). This may also include connections that

terminate before the IW is full, i.e., as a separate check at the time of the connection closing.

The period over which the IW is updated is intended to be a long timescale, e.g., a month or so, or 1,000 connections, whichever is longer. An implementation might check the IW once a month, and simply not update the IW or clear the connection counts in months where the number of connections is too small.

#### C.4. Discussion

There are numerous parameters to the above algorithm that are compliant with the given requirements; this is intended to allow variation in configuration and implementation while ensuring that all such algorithms are reactive and safe.

This algorithm continues to assume segments because that is the basis of most TCP implementations. It might be useful to consider revising the specifications to allow byte-based congestion given sufficient experience.

The algorithm checks for IW losses only during the first IW after a connection start; it does not check for IW losses elsewhere the IW is used, e.g., during slow-start restarts.

>> An implementation MAY detect IW losses during slow-start restarts in addition to losses during the first IW of a connection. In this case, the implementation MUST count each restart as a "connection" for the purposes of connection counts and periodic rechecking of the IW value.

False positives can occur during some kinds of segment reordering, e.g., that might trigger spurious retransmissions even without a true segment loss. These are not expected to be sufficiently common to dominate the algorithm and its conclusions.

This mechanism does require additional per-connection state, which is currently common in some implementations, and is useful for other reasons (e.g., the ISN is used in TCP-AO [RFC5925]). The mechanism also benefits from persistent state kept across reboots, as would be other state sharing mechanisms (e.g., TCP Control Block Sharing per the main body of this document).

The receive window (rwnd) is not involved in this calculation. The size of rwnd is determined by receiver resources and provides space to accommodate segment reordering. It is not involved with

congestion control, which is the focus of this document and its management of the IW.

#### C.5. Observations

The IW may not converge to a single, global value. It also may not converge at all, but rather may oscillate by a few MSS as it repeatedly probes the Internet for larger IWs and fails. Both properties are consistent with TCP behavior during each individual connection.

This mechanism assumes that losses during the IW are due to IW size. Persistent errors that drop packets for other reasons - e.g., OS bugs, can cause false positives. Again, this is consistent with TCP's basic assumption that loss is caused by congestion and requires backoff. This algorithm treats the IW of new connections as a long-timescale backoff system.





Internet Engineering Task Force  
Internet-Draft  
Obsoletes: 793, 879, 2873, 6093, 6429, 6528,  
6691 (if approved)  
Updates: 5961, 1011, 1122 (if approved)  
Intended status: Standards Track  
Expires: 8 September 2022

W. Eddy, Ed.  
MTI Systems  
7 March 2022

Transmission Control Protocol (TCP) Specification  
draft-ietf-tcpm-rfc793bis-28

Abstract

This document specifies the Transmission Control Protocol (TCP). TCP is an important transport layer protocol in the Internet protocol 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, as well as RFCs 879, 2873, 6093, 6429, 6528, and 6691 that updated parts of RFC 793. It updates RFCs 1011 and 1122, and should be considered as a replacement for the portions of those document dealing with TCP requirements. It also updates RFC 5961 by adding a small clarification in reset handling while in the SYN-RECEIVED state. The TCP header control bits from RFC 793 have also been updated based on RFC 3168.

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.

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 8 September 2022.

## Copyright Notice

Copyright (c) 2022 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised 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 . . . . .	4
2. Introduction . . . . .	5
2.1. Requirements Language . . . . .	5
2.2. Key TCP Concepts . . . . .	6
3. Functional Specification . . . . .	6
3.1. Header Format . . . . .	6
3.2. Specific Option Definitions . . . . .	12
3.2.1. Other Common Options . . . . .	13
3.2.2. Experimental TCP Options . . . . .	13
3.3. TCP Terminology Overview . . . . .	13
3.3.1. Key Connection State Variables . . . . .	13
3.3.2. State Machine Overview . . . . .	15
3.4. Sequence Numbers . . . . .	18
3.4.1. Initial Sequence Number Selection . . . . .	21
3.4.2. Knowing When to Keep Quiet . . . . .	23
3.4.3. The TCP Quiet Time Concept . . . . .	23
3.5. Establishing a connection . . . . .	25
3.5.1. Half-Open Connections and Other Anomalies . . . . .	28
3.5.2. Reset Generation . . . . .	31
3.5.3. Reset Processing . . . . .	32

3.6.	Closing a Connection . . . . .	32
3.6.1.	Half-Closed Connections . . . . .	35
3.7.	Segmentation . . . . .	35
3.7.1.	Maximum Segment Size Option . . . . .	37
3.7.2.	Path MTU Discovery . . . . .	38
3.7.3.	Interfaces with Variable MTU Values . . . . .	39
3.7.4.	Nagle Algorithm . . . . .	39
3.7.5.	IPv6 Jumbograms . . . . .	40
3.8.	Data Communication . . . . .	40
3.8.1.	Retransmission Timeout . . . . .	41
3.8.2.	TCP Congestion Control . . . . .	41
3.8.3.	TCP Connection Failures . . . . .	42
3.8.4.	TCP Keep-Alives . . . . .	43
3.8.5.	The Communication of Urgent Information . . . . .	44
3.8.6.	Managing the Window . . . . .	45
3.9.	Interfaces . . . . .	50
3.9.1.	User/TCP Interface . . . . .	50
3.9.2.	TCP/Lower-Level Interface . . . . .	59
3.10.	Event Processing . . . . .	61
3.10.1.	OPEN Call . . . . .	63
3.10.2.	SEND Call . . . . .	64
3.10.3.	RECEIVE Call . . . . .	65
3.10.4.	CLOSE Call . . . . .	67
3.10.5.	ABORT Call . . . . .	68
3.10.6.	STATUS Call . . . . .	69
3.10.7.	SEGMENT ARRIVES . . . . .	70
3.10.8.	Timeouts . . . . .	84
4.	Glossary . . . . .	84
5.	Changes from RFC 793 . . . . .	89
6.	IANA Considerations . . . . .	96
7.	Security and Privacy Considerations . . . . .	97
8.	Acknowledgements . . . . .	99
9.	References . . . . .	100
9.1.	Normative References . . . . .	100
9.2.	Informative References . . . . .	102
Appendix A.	Other Implementation Notes . . . . .	107
A.1.	IP Security Compartment and Precedence . . . . .	108
A.1.1.	Precedence . . . . .	108
A.1.2.	MLS Systems . . . . .	109
A.2.	Sequence Number Validation . . . . .	109
A.3.	Nagle Modification . . . . .	109
A.4.	Low Watermark Settings . . . . .	110
Appendix B.	TCP Requirement Summary . . . . .	110
Author's Address	. . . . .	114

## 1. Purpose and Scope

In 1981, RFC 793 [16] 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 widely implemented, 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 core specification for TCP [50]. Over time, a number of errata have been filed against RFC 793. There have also been deficiencies found and resolved in security, performance, and many other aspects. The number of enhancements has grown over time across many separate documents. These were never accumulated together into a comprehensive update to the base specification.

The purpose of this document is to bring together all of the IETF Standards Track changes and other clarifications that have been made to the base TCP functional specification and unify them into an updated version of RFC 793.

Some companion documents are referenced for important algorithms that are used by TCP (e.g. for congestion control), but have not been completely included 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. This document focuses on the common basis all TCP implementations must support in order to interoperate. Since 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 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. This document preserves 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 purposes, 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, connection termination, and packet retransmission to repair losses.

This document describes the basic functionality expected in modern TCP implementations, and replaces the protocol specification in RFC 793. It does not replicate or attempt to update the introduction and philosophy content in Sections 1 and 2 of 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.

The "TCP Roadmap" [50] provides a more extensive guide to the RFCs that define TCP and describe various important algorithms. The TCP Roadmap contains sections on strongly encouraged enhancements that improve performance and other aspects of TCP beyond the basic operation specified in this document. As one example, implementing congestion control (e.g. [8]) is a TCP requirement, but is a complex topic on its own, and not described in detail in this document, as there are many options and possibilities that do not impact basic interoperability. Similarly, most TCP implementations today include the high-performance extensions in [48], but these are not strictly required or discussed in this document. Multipath considerations for TCP are also specified separately in [59].

A list of changes from RFC 793 is contained in Section 5.

### 2.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [3][12] when, and only when, they appear in all capitals, as shown here.

Each use of RFC 2119 keywords in the document is individually labeled and referenced in Appendix B that summarizes implementation requirements.

Sentences using "MUST" are labeled as "MUST-X" with X being a numeric identifier enabling the requirement to be located easily when referenced from Appendix B.

Similarly, sentences using "SHOULD" are labeled with "SHLD-X", "MAY" with "MAY-X", and "RECOMMENDED" with "REC-X".

For the purposes of this labeling, "SHOULD NOT" and "MUST NOT" are labeled the same as "SHOULD" and "MUST" instances.

## 2.2. Key TCP Concepts

TCP provides a reliable, in-order, byte-stream service to applications.

The application byte-stream is conveyed over the network via TCP segments, with each TCP segment sent as an Internet Protocol (IP) datagram.

TCP reliability consists of detecting packet losses (via sequence numbers) and errors (via per-segment checksums), as well as correction via retransmission.

TCP supports unicast delivery of data. Anycast applications exist that successfully use TCP without modifications, though there is some risk of instability due to changes of lower-layer forwarding behavior [47].

TCP is connection-oriented, though does not inherently include a liveness detection capability.

Data flow is supported bidirectionally over TCP connections, though applications are free to send data only unidirectionally, if they so choose.

TCP uses port numbers to identify application services and to multiplex distinct flows between hosts.

A more detailed description of TCP features compared to other transport protocols can be found in Section 3.1 of [53]. Further description of the motivations for developing TCP and its role in the Internet protocol stack can be found in Section 2 of [16] and earlier versions of the TCP specification.

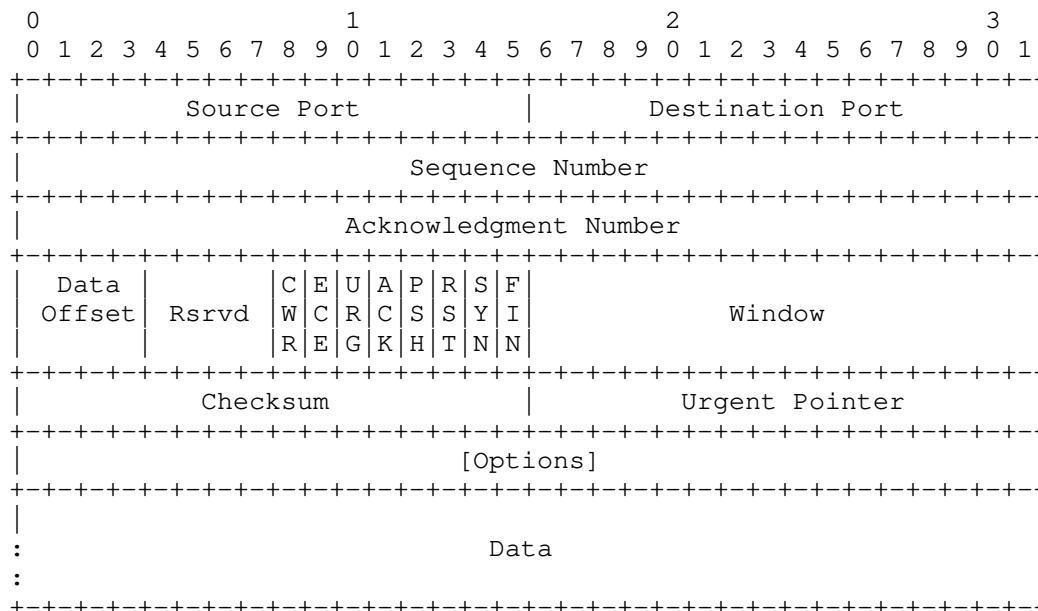
## 3. Functional Specification

### 3.1. Header Format

TCP segments are sent as internet datagrams. The Internet Protocol (IP) header carries several information fields, including the source and destination host addresses [1] [13]. A TCP header follows the IP headers, supplying information specific to the TCP protocol. This division allows for the existence of host level protocols other than TCP. In early development of the Internet suite of protocols, the IP header fields had been a part of TCP.

This document describes the TCP protocol. The TCP protocol uses TCP Headers.

A TCP Header, followed by any user data in the segment, is formatted as follows, using the style from [67]:



Note that one tick mark represents one bit position.

Figure 1: TCP Header Format

where:

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 the SYN flag is set). If SYN is set 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 (DOffset): 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 integer multiple of 32 bits long.

Reserved (Rsrvd): 4 bits.

A set of control bits reserved for future use. Must be zero in generated segments and must be ignored in received segments, if corresponding future features are unimplemented by the sending or receiving host.

The control bits are also known as "flags". Assignment is managed by IANA from the "TCP Header Flags" registry [63]. The currently assigned control bits are CWR, ECE, URG, ACK, PSH, RST, SYN, and FIN.

CWR: 1 bit.

Congestion Window Reduced (see [6]).

ECE: 1 bit.

ECN-Echo (see [6]).

URG: 1 bit.

Urgent Pointer field is significant.

ACK: 1 bit.

Acknowledgment field is significant.

PSH: 1 bit.

Push Function (see the Send Call description in Section 3.9.1).

RST: 1 bit.

Reset the connection.

SYN: 1 bit.

Synchronize sequence numbers.

FIN: 1 bit.

No more data from sender.

Window: 16 bits.



The number of data octets beginning with the one indicated in the acknowledgment field that the sender of this segment is willing to accept. The value is shifted when the Window Scaling extension is used [48].

The window size MUST be treated as an unsigned number, or else large window sizes will appear like negative windows and TCP will not work (MUST-1). It is RECOMMENDED that implementations will reserve 32-bit fields for the send and receive window sizes in the connection record and do all window computations with 32 bits (RECOMMENDED-1).

Checksum: 16 bits.

The checksum field is the 16 bit ones' complement of the ones' complement sum of all 16 bit words in the header and text. The checksum computation needs to ensure the 16-bit alignment of the data being summed. If a segment contains an odd number of header and text octets, alignment can be achieved by padding the last octet with zeros on its right 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 pseudo header (Figure 2) conceptually prefixed to the TCP header. The pseudo header is 96 bits for IPv4 and 320 bits for IPv6. Including the pseudo header in the checksum gives the TCP connection protection against misrouted segments. This information is carried in IP headers and is transferred across the TCP/Network interface in the arguments or results of calls by the TCP implementation on the IP layer.

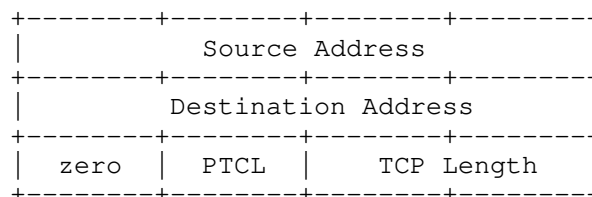


Figure 2: IPv4 Pseudo Header

Pseudo header components for IPv4:

Source Address: the IPv4 source address in network byte order

Destination Address: the IPv4 destination address in network byte order

zero: bits set to zero

PTCL: the protocol number from the IP header

TCP Length: 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.

For IPv6, the pseudo header is defined in Section 8.1 of RFC 8200 [13], and contains the IPv6 Source Address and Destination Address, an Upper Layer Packet Length (a 32-bit value otherwise equivalent to TCP Length in the IPv4 pseudo header), three bytes of zero-padding, and a Next Header value (differing from the IPv6 header value in the case of extension headers present in between IPv6 and TCP).

The TCP checksum is never optional. The sender MUST generate it (MUST-2) and the receiver MUST check it (MUST-3).

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 to be interpreted in segments with the URG control bit set.

Options: [TCP Option];  $\text{size}(\text{Options}) == (\text{DOffset} - 5) * 32$ ; present only when DOffset > 5. Note that this size expression also includes any padding trailing the actual options present.

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 (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 of zeros (MUST-69).

The list of all currently defined options is managed by IANA [62], and each option is defined in other RFCs, as indicated there. That set includes experimental options that can be extended to support multiple concurrent usages [46].

A given TCP implementation can support any currently defined options, but the following options MUST be supported (MUST-4 - note Maximum Segment Size option support is also part of MUST-19 in Section 3.7.2):

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

These options are specified in detail in Section 3.2.

A TCP implementation MUST be able to receive a TCP option in any segment (MUST-5).

A TCP implementation MUST (MUST-6) 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 MUST have length fields, including all future options (MUST-68). TCP implementations MUST be prepared to handle an illegal option length (e.g., zero); a suggested procedure is to reset the connection and log the error cause (MUST-7).

Note: There is ongoing work to extend the space available for TCP options, such as [66].

Data: variable length.

User data carried by the TCP segment.

### 3.2. Specific Option Definitions

A TCP Option, in the mandatory option set, is one of: an End of Option List Option, a No-Operation Option, or a Maximum Segment Size Option.

An End of Option List Option is formatted as follows:

```

0
0 1 2 3 4 5 6 7
+---+---+---+---+---+---+
|           0           |
+---+---+---+---+---+---+

```

where:

Kind: 1 byte; 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.

A No-Operation Option is formatted as follows:

```

0
0 1 2 3 4 5 6 7
+---+---+---+---+---+---+
|           1           |
+---+---+---+---+---+---+

```

where:

Kind: 1 byte; Kind == 1.

This option code can 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 (MUST-64).

A Maximum Segment Size Option is formatted as follows:

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           2           |      Length      | Maximum Segment Size (MSS) |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

where:

Kind: 1 byte; Kind == 2.

If this option is present, then it communicates the maximum receive segment size at the TCP endpoint that sends this segment. This value is limited by the IP reassembly limit. 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 (MUST-65). If this option is not used, any segment size is allowed. A more complete description of this option is provided in Section 3.7.1.

Length: 1 byte; Length == 4.

Length of the option in bytes.

Maximum Segment Size (MSS): 2 bytes.

The maximum receive segment size at the TCP endpoint that sends this segment.

### 3.2.1. Other Common Options

Additional RFCs define some other commonly used options that are recommended to implement for high performance, but not necessary for basic TCP interoperability. These are the TCP Selective Acknowledgement (SACK) option [23][27], TCP Timestamp (TS) option [48], and TCP Window Scaling (WS) option [48].

### 3.2.2. Experimental TCP Options

Experimental TCP option values are defined in [31], and [46] describes the current recommended usage for these experimental values.

## 3.3. TCP Terminology Overview

This section includes an overview of key terms needed to understand the detailed protocol operation in the rest of the document. There is a glossary of terms in Section 4.

### 3.3.1. Key Connection State Variables

Before we can discuss very much about the operation of the TCP implementation we need to introduce some detailed terminology. The maintenance of a TCP connection requires maintaining state for 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 IP addresses and port numbers, the IP security level and compartment of the

connection (see Appendix A.1), 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.

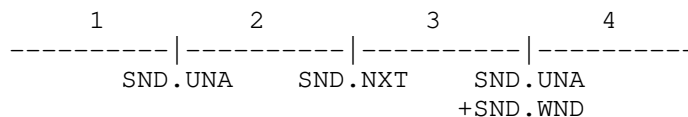
#### Send Sequence Variables:

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

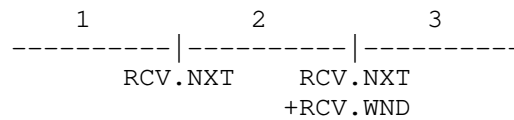
The following diagrams may help to relate some of these variables to the sequence space.



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

Figure 3: Send Sequence Space

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



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

Figure 4: Receive Sequence Space

The receive window is the portion of the sequence space labeled 2 in Figure 4.

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

### 3.3.2. State Machine Overview

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 peer 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 peer, or an acknowledgment of the connection termination request previously sent.

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

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 peer.

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

TIME-WAIT - represents waiting for enough time to pass to be sure the remote TCP peer received the acknowledgment of its connection termination request, and to avoid new connections being impacted by delayed segments from previous connections.

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 OPEN call specifies whether connection establishment is to be actively pursued, or to be passively waited for.

A passive OPEN request means that the process wants to accept incoming connection requests, in contrast to an active OPEN attempting to initiate a connection.

The state diagram in Figure 5 illustrates only state changes, together with the causing events and resulting actions, but addresses neither error conditions nor actions that are not connected with state changes. In a later section, more detail is offered with respect to the reaction of the TCP implementation to events. Some state names are abbreviated or hyphenated differently in the diagram from how they appear elsewhere in the document.

NOTA BENE: This diagram is only a summary and must not be taken as the total specification. Many details are not included.



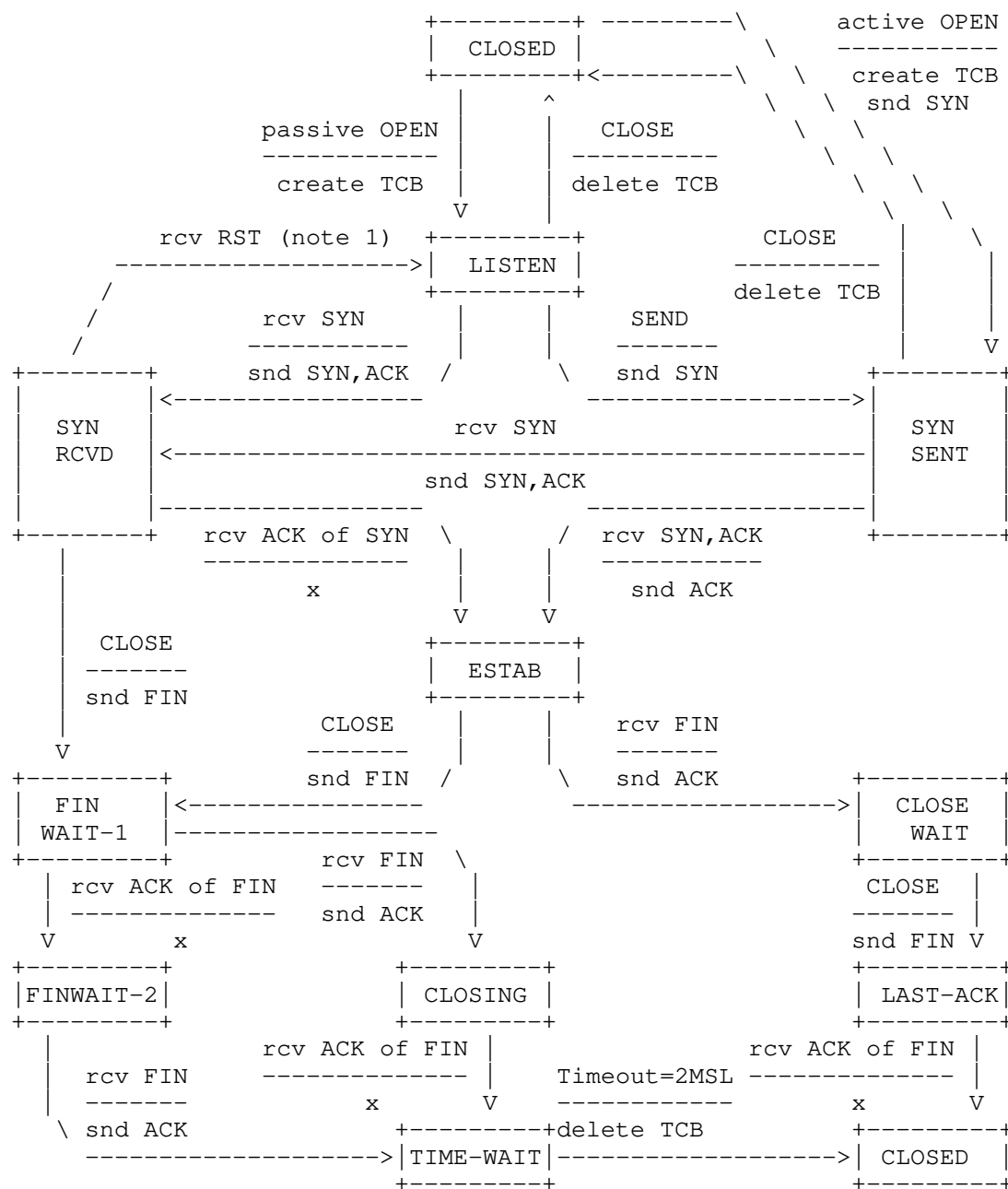


Figure 5: TCP Connection State Diagram

The following notes apply to Figure 5:

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

Note 2: The figure omits a transition from FIN-WAIT-1 to TIME-WAIT if a FIN is received and the local FIN is also acknowledged.

Note 3: A RST can be sent from any state with a corresponding transition to TIME-WAIT (see [71] for rationale). These transitions are not explicitly shown, otherwise the diagram would become very difficult to read. Similarly, receipt of a RST from any state results in a transition to LISTEN or CLOSED, though this is also omitted from the diagram for legibility.

### 3.4. 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 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 that the TCP implementation 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 that are expected (i.e., that the segment "overlaps" the receive window).

In response to sending data the TCP endpoint 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 peer (next sequence number expected by the receiving TCP peer)

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 segment, 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 implementation to maintain a zero receive window while transmitting data and receiving ACKs. A TCP receiver MUST process the RST and URG fields of all incoming segments, even when the receive window is zero (MUST-66).

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.

### 3.4.1. Initial Sequence Number Selection

A connection is defined by a pair of sockets. Connections can be reused. 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 implementation 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 support this, the TIME-WAIT state limits the rate of connection reuse, while the initial sequence number selection described below further protects against ambiguity about what incarnation of a connection an incoming packet corresponds to.

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 endpoint loses all knowledge of the sequence numbers it has been using. When new connections are created, an initial sequence number (ISN) generator is employed that 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 [43].

TCP Initial Sequence Numbers are generated from a number sequence that monotonically increases until it wraps, known loosely as a "clock". This clock is a 32-bit counter that typically increments at least once every roughly 4 microseconds, although it is neither assumed to be realtime nor precise, and need not persist across reboots. The clock component is intended to ensure 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.

A TCP implementation **MUST** use the above type of "clock" for clock-driven selection of initial sequence numbers (**MUST-8**), and **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") (SHLD-1). F() MUST NOT be computable from the outside (MUST-9), 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 [43].

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

For a connection to be established or initialized, the two TCP peers 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 ISNs.

The synchronization requires each side to send its own initial sequence number and to receive a confirmation of it in acknowledgment from the remote TCP peer. Each side must also receive the remote peer'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 (3WHS).

A 3WHS is necessary because sequence numbers are not tied to a global clock in the network, and TCP implementations may have different mechanisms for picking the ISNs. The receiver of the first SYN has no way of knowing whether the segment was an old 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 for ISN selection are discussed in [70].

### 3.4.2. Knowing When to Keep Quiet

A theoretical problem exists where data could be corrupted due to confusion between old segments in the network and new ones after a host reboots, if the same port numbers and sequence space are reused. The "Quiet Time" concept discussed below addresses this and the discussion of it is included for situations where it might be relevant, although it is not felt to be necessary in most current implementations. The problem was more relevant earlier in the history of TCP. In practical use on the Internet today, the error-prone conditions are sufficiently unlikely that it is felt safe to ignore. Reasons why it is now negligible include: (a) ISS and ephemeral port randomization have reduced likelihood of reuse of port numbers and sequence numbers after reboots, (b) the effective MSL of the Internet has declined as links have become faster, and (c) reboots often taking longer than an MSL anyways.

To be sure that a TCP implementation does not create a segment carrying a sequence number that may be duplicated by an old segment remaining in the network, the TCP endpoint must keep quiet for an MSL before assigning any sequence numbers upon starting up or recovering from a situation where 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 endpoint 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.

### 3.4.3. The TCP Quiet Time Concept

Hosts that for any reason lose 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 MSL in the internet system that the host is a part of. 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 data by some receivers in the internet system.

TCP endpoints 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, TCP implementations 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 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 10s of megabits/sec. At 100 megabits/sec, the cycle time is 5.4 minutes, which may be a little short, but still within reason. Much higher data rates are possible today, with implications described in the final paragraph of this subsection.

The basic duplicate detection and sequencing algorithm in TCP can be defeated, however, if a source TCP endpoint does not have any memory of the sequence numbers it last used on a given connection. For example, if the TCP implementation were to start all connections with sequence number 0, then upon the host rebooting, a TCP peer might reform 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 that 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 endpoint on a particular connection. Now suppose, at this instant, the host reboots 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 S1 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 was down between rebooting 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 reboot - this is the "quiet time" specification. Hosts that prefer to avoid waiting and are willing to risk possible confusion of old and new packets at a given destination may choose not to wait for the "quiet time". Implementors may provide TCP users with the ability to select on a connection by connection basis whether to wait after a reboot, or may informally implement the "quiet 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 rebooting a block of space-time is occupied by the octets and SYN or FIN flags of any potentially still in-flight segments, and if a new connection is started too soon and uses any of the sequence numbers in the space-time footprint of those potentially still in-flight segments of the previous connection incarnation, there is a potential sequence number overlap area that could cause confusion at the receiver.

High performance cases will have shorter cycle times than those in the megabits per second that the base TCP design described above considers. At 1 Gbps, the cycle time is 34 seconds, only 3 seconds at 10 Gbps, and around a third of a second at 100 Gbps. In these higher performance cases, TCP Timestamp options and Protection Against Wrapped Sequences (PAWS) [48] provide the needed capability to detect and discard old duplicates.

### 3.5. Establishing a connection

The "three-way handshake" is the procedure used to establish a connection. This procedure normally is initiated by one TCP peer and responded to by another TCP peer. The procedure also works if two TCP peers simultaneously initiate the procedure. When simultaneous open occurs, each TCP peer receives a "SYN" segment that 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 endpoint doesn't deliver the data to the user until it is clear the data is valid (e.g., the data is buffered at the receiver until the connection reaches the ESTABLISHED state, given that the three-way handshake reduces the possibility of false connections). It is a trade-off between memory and messages to provide information for this checking.

The simplest 3WHS is shown in Figure 6. 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 peer A to TCP peer B, or arrival of a segment at B from A. Left arrows (<--), indicate the reverse. Ellipsis (...) indicates a segment that is still in the network (delayed). Comments appear in parentheses. TCP connection 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 Peer A		TCP Peer 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

Figure 6: Basic 3-Way Handshake for Connection Synchronization

In line 2 of Figure 6, TCP Peer A begins by sending a SYN segment indicating that it will use sequence numbers starting with sequence number 100. In line 3, TCP Peer B sends a SYN and acknowledges the SYN it received from TCP Peer A. Note that the acknowledgment field indicates TCP Peer B is now expecting to hear sequence 101, acknowledging the SYN that occupied sequence 100.

At line 4, TCP Peer A responds with an empty segment containing an ACK for TCP Peer B's SYN; and in line 5, TCP Peer 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 ACKs!).

Simultaneous initiation is only slightly more complex, as is shown in Figure 7. Each TCP peer's connection state cycles from CLOSED to SYN-SENT to SYN-RECEIVED to ESTABLISHED.

TCP Peer A		TCP Peer 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

Figure 7: Simultaneous Connection Synchronization

A TCP implementation MUST support simultaneous open attempts (MUST-10).

Note that a TCP implementation MUST keep track of whether a connection has reached SYN-RECEIVED state as the result of a passive OPEN or an active OPEN (MUST-11).

The principal 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, is specified. If the receiving TCP peer is in a non-synchronized state (i.e., SYN-SENT, SYN-RECEIVED), it returns to LISTEN on receiving an acceptable reset. If the TCP peer 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 Peer A		TCP Peer 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. ESTABLISHED	<-- <SEQ=400><ACK=101><CTL=SYN,ACK>	<-- SYN-RECEIVED
8. ESTABLISHED	--> <SEQ=101><ACK=401><CTL=ACK>	--> ESTABLISHED

Figure 8: Recovery from Old Duplicate SYN

As a simple example of recovery from old duplicates, consider Figure 8. At line 3, an old duplicate SYN arrives at TCP Peer B. TCP Peer B cannot tell that this is an old duplicate, so it responds normally (line 4). TCP Peer A detects that the ACK field is incorrect and returns a RST (reset) with its SEQ field selected to make the segment believable. TCP Peer B, on receiving the RST, returns to the LISTEN state. When the original SYN 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.

### 3.5.1. Half-Open Connections and Other Anomalies

An established connection is said to be "half-open" if one of the TCP peers 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 failure or reboot 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.

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 endpoint receiving a reset control message. Such a message indicates to the site B TCP endpoint 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 failure or reboot occurs causing loss of memory to A's TCP implementation. Depending on the operating system supporting A's TCP implementation, it is likely that some error recovery mechanism exists. When the TCP endpoint 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 implementation. In an attempt to establish the connection, A's TCP implementation will send a segment containing SYN. This scenario leads to the example shown in Figure 9. After TCP Peer A reboots, the user attempts to re-open the connection. TCP Peer B, in the meantime, thinks the connection is open.

TCP Peer A	TCP Peer B
1. (REBOOT)	(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>	-->

Figure 9: Half-Open Connection Discovery

When the SYN arrives at line 3, TCP Peer 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 Peer 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 Peer B aborts at line 5. TCP Peer A will continue to try to establish the connection; the problem is now reduced to the basic 3-way handshake of Figure 6.

An interesting alternative case occurs when TCP Peer A reboots and TCP Peer B tries to send data on what it thinks is a synchronized connection. This is illustrated in Figure 10. In this case, the data arriving at TCP Peer A from TCP Peer B (line 2) is unacceptable because no such connection exists, so TCP Peer A sends a RST. The RST is acceptable so TCP Peer B processes it and aborts the connection.

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

Figure 10: Active Side Causes Half-Open Connection Discovery

In Figure 11, two TCP Peers A and B with passive connections waiting for SYN are depicted. An old duplicate arriving at TCP Peer 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 Peer B accepts the reset and returns to its passive LISTEN state.

TCP Peer A	TCP Peer 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

Figure 11: Old Duplicate SYN Initiates a Reset on two Passive Sockets

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

### 3.5.2. Reset Generation

A TCP user or application can issue a reset on a connection at any time, though reset events are also generated by the protocol itself when various error conditions occur, as described below. The side of a connection issuing a reset should enter the TIME-WAIT state, as this generally helps to reduce the load on busy servers for reasons described in [71].

As a general rule, reset (RST) is sent whenever a segment arrives that 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. A SYN segment that does not match an existing connection is 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 Appendix A.1 that does not exactly match the level and compartment requested for the connection, a reset is 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 be responded to with an empty acknowledgment segment (without any user data) 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 that does not exactly match the level and compartment 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.

### 3.5.3. 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.

TCP implementations SHOULD allow a received RST segment to include data (SHLD-2). It has been suggested that a RST segment could contain diagnostic data that explains the cause of the RST. No standard has yet been established for such data.

### 3.6. 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 the TCP receiver is told that the remote peer 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 remote peer has CLOSED. The TCP implementation will signal a user, even if no RECEIVES are outstanding, that the remote peer has closed, so the user can terminate their side gracefully. A TCP implementation 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 their data was received at the destination TCP endpoint. Users must keep reading connections they close for sending until the TCP implementation indicates there is no more data.

There are essentially three cases:



- 1) The user initiates by telling the TCP implementation to CLOSE the connection (TCP Peer A in Figure 12).
- 2) The remote TCP endpoint initiates by sending a FIN control signal (TCP Peer B in Figure 12).
- 3) Both users CLOSE simultaneously (Figure 13).

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 implementation, 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 peer has both acknowledged the FIN and sent a FIN of its own, the first TCP peer can ACK this FIN. Note that a TCP endpoint receiving a FIN will ACK but not send its own FIN until its user has CLOSED the connection also.

Case 2: TCP endpoint receives a FIN from the network

If an unsolicited FIN arrives from the network, the receiving TCP endpoint can ACK it and tell the user that the connection is closing. The user will respond with a CLOSE, upon which the TCP endpoint can send a FIN to the other TCP peer after sending any remaining data. The TCP endpoint 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 (Figure 13). When all segments preceding the FINs have been processed and acknowledged, each TCP peer can ACK the FIN it has received. Both will, upon receiving these ACKs, delete the connection.

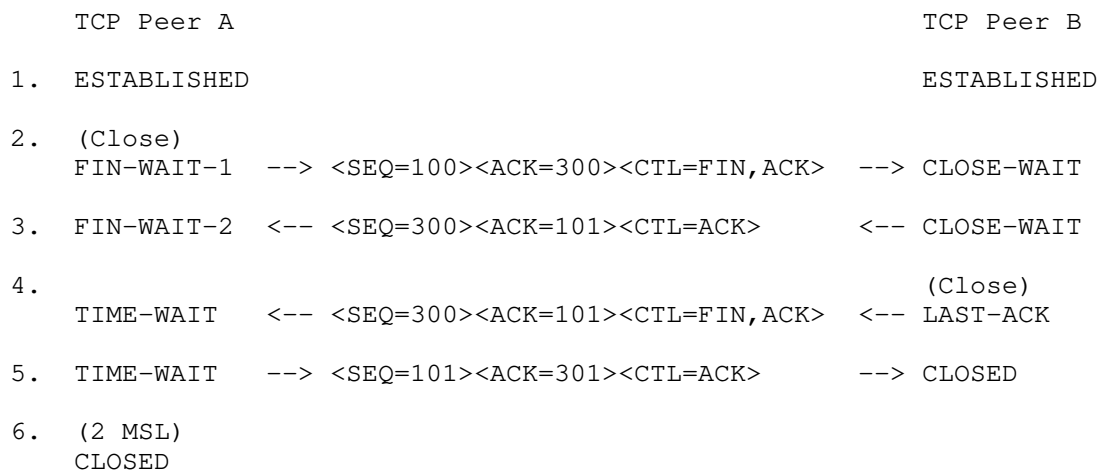


Figure 12: Normal Close Sequence

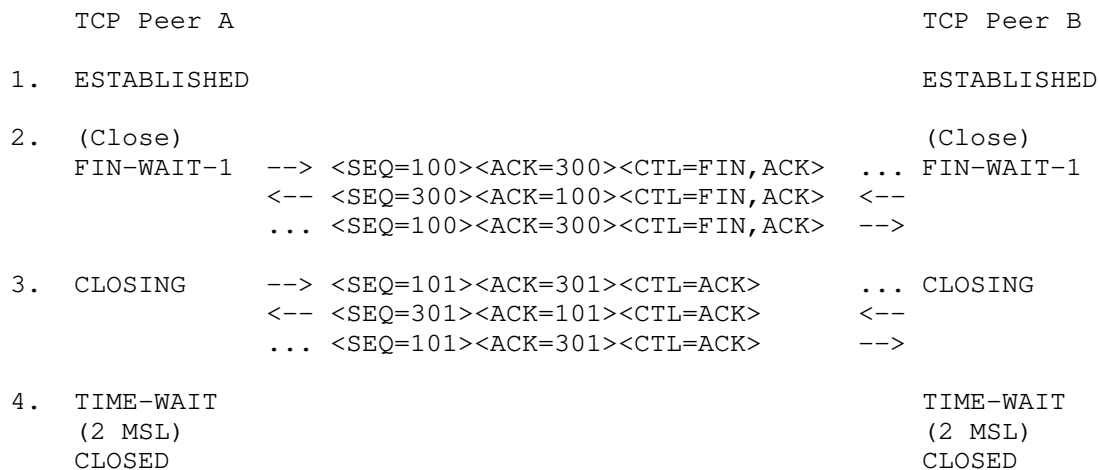


Figure 13: Simultaneous Close Sequence

A TCP connection may terminate in two ways: (1) the normal TCP close sequence using a FIN handshake (Figure 12), and (2) an "abort" in which one or more RST segments are sent and the connection state is immediately discarded. If the local TCP connection is closed by the remote side due to a FIN or RST received from the remote side, then the local application MUST be informed whether it closed normally or was aborted (MUST-12).

### 3.6.1. Half-Closed Connections

The normal TCP close sequence delivers buffered data reliably in both directions. Since the two directions of a TCP connection are closed independently, it is possible for a connection to be "half closed," i.e., closed in only one direction, and a host is permitted to continue sending data in the open direction on a half-closed connection.

A host MAY implement a "half-duplex" TCP close sequence, so that an application that has called CLOSE cannot continue to read data from the connection (MAY-1). If such a host issues a CLOSE call while received data is still pending in the TCP connection, or if new data is received after CLOSE is called, its TCP implementation SHOULD send a RST to show that data was lost (SHLD-3). See [24] section 2.17 for discussion.

When a connection is closed actively, it MUST linger in the TIME-WAIT state for a time  $2 \times \text{MSL}$  (Maximum Segment Lifetime) (MUST-13). However, it MAY accept a new SYN from the remote TCP endpoint to reopen the connection directly from TIME-WAIT state (MAY-2), if it:

- (1) assigns its initial sequence number for the new connection to be larger than the largest sequence number it used on the previous connection incarnation, and
- (2) returns to TIME-WAIT state if the SYN turns out to be an old duplicate.

When the TCP Timestamp options are available, an improved algorithm is described in [41] in order to support higher connection establishment rates. This algorithm for reducing TIME-WAIT is a Best Current Practice that SHOULD be implemented, since timestamp options are commonly used, and using them to reduce TIME-WAIT provides benefits for busy Internet servers (SHLD-4).

### 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. Individual TCP segments often do not correspond one-for-one to individual send (or socket write) calls from the application. Applications may perform writes at the granularity of messages in the upper layer protocol, but TCP guarantees no boundary coherence between the TCP segments sent and received versus user application data read or write buffer boundaries. In some specific protocols, such as Remote Direct Memory Access (RDMA) using Direct Data Placement (DDP) and Marker PDU

Aligned Framing (MPA) [35], there are performance optimizations possible when the relation between TCP segments and application data units can be controlled, and MPA includes a specific mechanism for detecting and verifying this relationship between TCP segments and application message data structures, but this is specific to applications like RDMA. In general, multiple goals influence the sizing of TCP segments created by a TCP implementation.

Goals driving the sending of larger segments include:

- \* Reducing the number of packets in flight within the network.
- \* Increasing processing efficiency and potential performance by enabling a smaller number of interrupts and inter-layer interactions.
- \* Limiting the overhead of TCP headers.

Note that the performance benefits of sending larger segments may decrease as the size increases, and there may be boundaries where advantages are reversed. For instance, on some implementation architectures, 1025 bytes within a segment could lead to worse performance than 1024 bytes, due purely to data alignment on copy operations.

Goals driving the sending of smaller segments include:

- \* Avoiding sending a TCP segment that would result in an IP datagram larger than the smallest MTU along an IP network path, because this results in either packet loss or packet fragmentation. Making matters worse, some firewalls or middleboxes may drop fragmented packets or ICMP messages related to fragmentation.
- \* Preventing delays to the application data stream, especially when TCP is waiting on the application to generate more data, or when the application is waiting on an event or input from its peer in order to generate more data.
- \* Enabling "fate sharing" between TCP segments and lower-layer data units (e.g. below IP, for links with cell or frame sizes smaller than the IP MTU).

Towards meeting these competing sets of goals, TCP includes several mechanisms, including the Maximum Segment Size option, Path MTU Discovery, the Nagle algorithm, and support for IPv6 Jumbograms, as discussed in the following subsections.

### 3.7.1. Maximum Segment Size Option

TCP endpoints **MUST** implement both sending and receiving the MSS option (MUST-14).

TCP implementations **SHOULD** send an MSS option in every SYN segment when its receive MSS differs from the default 536 for IPv4 or 1220 for IPv6 (SHLD-5), and **MAY** send it always (MAY-3).

If an MSS option is not received at connection setup, TCP implementations **MUST** assume a default send MSS of 536 (576 - 40) for IPv4 or 1220 (1280 - 60) for IPv6 (MUST-15).

The maximum size of a segment that TCP endpoint really sends, the "effective send MSS," **MUST** be the smaller (MUST-16) of the send MSS (that reflects the available reassembly buffer size at the remote host, the EMTU\_R [20]) and the largest transmission size permitted by the IP layer (EMTU\_S [20]):

$$\text{Eff.snd.MSS} =$$
$$\min(\text{SendMSS}+20, \text{MMS\_S}) - \text{TCPhdrsize} - \text{IPOptionsize}$$

where:

- \* SendMSS is the MSS value received from the remote host, or the default 536 for IPv4 or 1220 for IPv6, if no MSS option is received.
- \* MMS\_S is the maximum size for a transport-layer message that TCP may send.
- \* TCPhdrsize is the size of the fixed TCP header and any options. This is 20 in the (rare) case that no options are present, but may be larger if TCP options are to be sent. Note that some options might not be included on all segments, but that for each segment sent, the sender should adjust the data length accordingly, within the Eff.snd.MSS.
- \* IPOptionsize is the size of any IPv4 options or IPv6 extension headers associated with a TCP connection. Note that some options or extension headers might not be included on all packets, but that for each segment sent, the sender should adjust the data length accordingly, within the Eff.snd.MSS.

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 [44] discusses this in greater detail.

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

$MMS\_R - 20$

where  $MMS\_R$  is the maximum size for a transport-layer message that can be received (and reassembled at the IP layer) (MUST-67). TCP obtains  $MMS\_R$  and  $MMS\_S$  from the IP layer; see the generic call `GET_MAXSIZES` in Section 3.4 of RFC 1122. These are defined in terms of their IP MTU equivalents,  $EMTU\_R$  and  $EMTU\_S$  [20].

When TCP is used in a situation where either the IP or TCP headers are not fixed, 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. This has been a point of confusion historically, as explained in RFC 6691, Section 3.1.

### 3.7.2. Path MTU Discovery

A TCP implementation may be aware of the MTU on directly connected links, but will rarely have insight about MTUs across an entire network path. For IPv4, RFC 1122 recommends an IP-layer default effective MTU of less than or equal to 576 for destinations not directly connected, and for IPv6 this would be 1280. Using these fixed values limits TCP connection performance and efficiency. Instead, implementation of Path MTU Discovery (PMTUD) and Packetization Layer Path MTU Discovery (PLPMTUD) is strongly recommended in order for TCP to improve segmentation decisions. Both PMTUD and PLPMTUD help TCP choose segment sizes that avoid both on-path (for IPv4) and source fragmentation (IPv4 and IPv6).

PMTUD for IPv4 [2] or IPv6 [14] is implemented in conjunction between TCP, IP, and ICMP protocols. It relies both on avoiding source fragmentation and setting the IPv4 DF (don't fragment) flag, the latter to inhibit on-path fragmentation. It relies on ICMP errors from routers along the path, whenever a segment is too large to traverse a link. Several adjustments to a TCP implementation with PMTUD are described in RFC 2923 in order to deal with problems experienced in practice [28]. PLPMTUD [32] is a Standards Track improvement to PMTUD that relaxes the requirement for ICMP support across a path, and improves performance in cases where ICMP is not consistently conveyed, but still tries to avoid source fragmentation. The mechanisms in all four of these RFCs are recommended to be included in TCP implementations.

The TCP MSS option specifies an upper bound for the size of packets that can be received (see [44]). Hence, setting the value in the MSS option too small can impact the ability for PMTUD or PLPMTUD to find a larger path MTU. RFC 1191 discusses this implication of many older TCP implementations setting the TCP MSS to 536 (corresponding to the IPv4 576 byte default MTU) for non-local destinations, rather than deriving it from the MTUs of connected interfaces as recommended.

### 3.7.3. Interfaces with Variable MTU Values

The effective MTU can sometimes vary, as when used with variable compression, e.g., RObust Header Compression (ROHC) [38]. It is tempting for a TCP implementation 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 packet-to-packet, TCP implementations SHOULD use the smallest effective MTU of the interface to calculate the value to advertise in the MSS option (SHLD-6).

### 3.7.4. Nagle Algorithm

The "Nagle algorithm" was described in RFC 896 [18] and was recommended in RFC 1122 [20] for mitigation of an early problem of too many small packets being generated. It has been implemented in most current TCP code bases, sometimes with minor variations (see Appendix A.3).

If there is unacknowledged data (i.e.,  $SND.NXT > SND.UNA$ ), then the sending TCP endpoint buffers all user data (regardless of the PSH bit), until the outstanding data has been acknowledged or until the TCP endpoint can send a full-sized segment ( $Eff.snd.MSS$  bytes).

A TCP implementation SHOULD implement the Nagle Algorithm to coalesce short segments (SHLD-7). However, there MUST be a way for an application to disable the Nagle algorithm on an individual connection (MUST-17). In all cases, sending data is also subject to the limitation imposed by the Slow Start algorithm [8].

Since there can be problematic interactions between the Nagle Algorithm and delayed acknowledgements, some implementations use minor variations of the Nagle algorithm, such as the one described in Appendix A.3.

### 3.7.5. IPv6 Jumbograms

In order to support TCP over IPv6 Jumbograms, implementations need to be able to send TCP segments larger than the 64KB limit that the MSS option can convey. RFC 2675 [25] defines that an MSS value of 65,535 bytes is to be treated as infinity, and Path MTU Discovery [14] is used to determine the actual MSS.

The Jumbo Payload option need not be implemented or understood by IPv6 nodes that do not support attachment to links with a MTU greater than 65,575 [25], and the present IPv6 Node Requirements does not include support for Jumbograms [55].

### 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 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 implementation 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 (see Section 3.9.1), as does the FIN control flag in an incoming segment.



### 3.8.1. Retransmission Timeout

Because of the variability of the networks that compose an internetwork system and the wide range of uses of TCP connections the retransmission timeout (RTO) must be dynamically determined.

The RTO MUST be computed according to the algorithm in [10], including Karn's algorithm for taking RTT samples (MUST-18).

RFC 793 contains an early example procedure for computing the RTO, based on work mentioned in IEN 177 [72]. This was then replaced by the algorithm described in RFC 1122, and subsequently updated in RFC 2988, and then again in RFC 6298.

RFC 1122 allows that if a retransmitted packet is identical to the original packet (which implies not only that the data boundaries have not changed, but also that none of the headers have changed), then the same IPv4 Identification field MAY be used (see Section 3.2.1.5 of RFC 1122) (MAY-4). The same IP identification field may be reused anyways, since it is only meaningful when a datagram is fragmented [45]. TCP implementations should not rely on or typically interact with this IPv4 header field in any way. It is not a reasonable way to either indicate duplicate sent segments, nor to identify duplicate received segments.

### 3.8.2. TCP Congestion Control

RFC 2914 [5] explains the importance of congestion control for the Internet.

RFC 1122 required implementation of Van Jacobson's congestion control algorithms slow start and congestion avoidance together with exponential back-off for successive RTO values for the same segment. RFC 2581 provided IETF Standards Track description of slow start and congestion avoidance, along with fast retransmit and fast recovery. RFC 5681 is the current description of these algorithms and is the current Standards Track specification providing guidelines for TCP congestion control. RFC 6298 describes exponential back-off of RTO values, including keeping the backed-off value until a subsequent segment with new data has been sent and acknowledged without retransmission.

A TCP endpoint MUST implement the basic congestion control algorithms slow start, congestion avoidance, and exponential back-off of RTO to avoid creating congestion collapse conditions (MUST-19). RFC 5681 and RFC 6298 describe the basic algorithms on the IETF Standards Track that are broadly applicable. Multiple other suitable algorithms exist and have been widely used. Many TCP implementations

support a set of alternative algorithms that can be configured for use on the endpoint. An endpoint MAY implement such alternative algorithms provided that the algorithms are conformant with the TCP specifications from the IETF Standards Track as described in RFC 2914, RFC 5033 [7], and RFC 8961 [15] (MAY-18).

Explicit Congestion Notification (ECN) was defined in RFC 3168 and is an IETF Standards Track enhancement that has many benefits [52].

A TCP endpoint SHOULD implement ECN as described in RFC 3168 (SHLD-8).

### 3.8.3. TCP Connection Failures

Excessive retransmission of the same segment by a TCP endpoint indicates some failure of the remote host or the Internet path. This failure may be of short or long duration. The following procedure MUST be used to handle excessive retransmissions of data segments (MUST-20):

- (a) There are two thresholds R1 and R2 measuring the amount of retransmission that has occurred for the same segment. R1 and R2 might be measured in time units or as a count of retransmissions (with the current RTO and corresponding backoffs as a conversion factor, if needed).
- (b) When the number of transmissions of the same segment reaches or exceeds threshold R1, pass negative advice (see Section 3.3.1.4 of [20]) to the IP layer, to trigger dead-gateway diagnosis.
- (c) When the number of transmissions of the same segment reaches a threshold R2 greater than R1, close the connection.
- (d) An application MUST (MUST-21) be able to set the value for R2 for a particular connection. For example, an interactive application might set R2 to "infinity," giving the user control over when to disconnect.
- (e) TCP implementations SHOULD inform the application of the delivery problem (unless such information has been disabled by the application; see Asynchronous Reports section), when R1 is reached and before R2 (SHLD-9). This will allow a remote login application program to inform the user, for example.

The value of R1 SHOULD correspond to at least 3 retransmissions, at the current RTO (SHLD-10). The value of R2 SHOULD correspond to at least 100 seconds (SHLD-11).

An attempt to open a TCP connection could fail with excessive retransmissions of the SYN segment or by receipt of a RST segment or an ICMP Port Unreachable. SYN retransmissions MUST be handled in the general way just described for data retransmissions, including notification of the application layer.

However, the values of R1 and R2 may be different for SYN and data segments. In particular, R2 for a SYN segment MUST be set large enough to provide retransmission of the segment for at least 3 minutes (MUST-23). The application can close the connection (i.e., give up on the open attempt) sooner, of course.

#### 3.8.4. TCP Keep-Alives

A TCP connection is said to be "idle" if for some long amount of time there have been no incoming segments received and there is no new or unacknowledged data to be sent.

Implementors MAY include "keep-alives" in their TCP implementations (MAY-5), although this practice is not universally accepted. Some TCP implementations, however, have included a keep-alive mechanism. To confirm that an idle connection is still active, these implementations send a probe segment designed to elicit a response from the TCP peer. Such a segment generally contains SEG.SEQ = SND.NXT-1 and may or may not contain one garbage octet of data. If keep-alives are included, the application MUST be able to turn them on or off for each TCP connection (MUST-24), and they MUST default to off (MUST-25).

Keep-alive packets MUST only be sent when no sent data is outstanding, and no data or acknowledgement packets have been received for the connection within an interval (MUST-26). This interval MUST be configurable (MUST-27) and MUST default to no less than two hours (MUST-28).

It is extremely important to remember that ACK segments that contain no data are not reliably transmitted by TCP. Consequently, if a keep-alive mechanism is implemented it MUST NOT interpret failure to respond to any specific probe as a dead connection (MUST-29).

An implementation SHOULD send a keep-alive segment with no data (SHLD-12); however, it MAY be configurable to send a keep-alive segment containing one garbage octet (MAY-6), for compatibility with erroneous TCP implementations.

### 3.8.5. The Communication of Urgent Information

As a result of implementation differences and middlebox interactions, new applications SHOULD NOT employ the TCP urgent mechanism (SHLD-13). However, TCP implementations MUST still include support for the urgent mechanism (MUST-30). Information on how some TCP implementations interpret the urgent pointer can be found in RFC 6093 [40].

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 endpoint 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 endpoint, that TCP must tell the user to go into "urgent mode"; when the receive sequence number catches up to the urgent pointer, the TCP implementation 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 an urgent field that 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. Note that because changes in the urgent pointer correspond to data being written by a sending application, the urgent pointer can not "recede" in the sequence space, but a TCP receiver should be robust to invalid urgent pointer values.

A TCP implementation MUST support a sequence of urgent data of any length (MUST-31). [20]

The urgent pointer MUST point to the sequence number of the octet following the urgent data (MUST-62).

A TCP implementation MUST (MUST-32) 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. The TCP implementation MUST (MUST-33)

provide a way for the application to learn how much urgent data remains to be read from the connection, or at least to determine whether more urgent data remains to be read [20].

#### 3.8.6. 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.

The sending TCP endpoint packages the data to be transmitted into segments that 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).

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 TCP endpoints. 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 endpoint 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 [20] dictates that TCP peers will not shrink the window themselves, but will be prepared for such behavior on the part of other TCP peers.

A TCP receiver SHOULD NOT shrink the window, i.e., move the right window edge to the left (SHLD-14). However, a sending TCP peer MUST be robust against window shrinking, which may cause the "usable window" (see Section 3.8.6.2.1) to become negative (MUST-34).

If this happens, the sender SHOULD NOT send new data (SHLD-15), but SHOULD retransmit normally the old unacknowledged data between SND.UNA and SND.UNA+SND.WND (SHLD-16). The sender MAY also retransmit old data beyond SND.UNA+SND.WND (MAY-7), but SHOULD NOT time out the connection if data beyond the right window edge is not acknowledged (SHLD-17). If the window shrinks to zero, the TCP implementation MUST probe it in the standard way (described below) (MUST-35).

#### 3.8.6.1. Zero Window Probing

The sending TCP peer must regularly transmit at least one octet of new data (if available) or retransmit to the receiving TCP peer even if the send window is zero, in order to "probe" the window. This retransmission is essential to guarantee that when either TCP peer has a zero window the re-opening of the window will be reliably reported to the other. This is referred to as Zero-Window Probing (ZWP) in other documents.

Probing of zero (offered) windows MUST be supported (MUST-36).

A TCP implementation MAY keep its offered receive window closed indefinitely (MAY-8). As long as the receiving TCP peer continues to send acknowledgments in response to the probe segments, the sending TCP peer MUST allow the connection to stay open (MUST-37). This enables TCP to function in scenarios such as the "printer ran out of paper" situation described in Section 4.2.2.17 of [20]. The behavior is subject to the implementation's resource management concerns, as noted in [42].

When the receiving TCP peer 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 transmitting host SHOULD send the first zero-window probe when a zero window has existed for the retransmission timeout period (SHLD-29) (Section 3.8.1), and SHOULD increase exponentially the interval between successive probes (SHLD-30).

#### 3.8.6.2. Silly Window Syndrome Avoidance

The "Silly Window Syndrome" (SWS) is a stable pattern of small incremental window movements resulting in extremely poor TCP performance. Algorithms to avoid SWS are described below for both the sending side and the receiving side. RFC 1122 contains more detailed discussion of the SWS problem. Note that the Nagle algorithm and the sender SWS avoidance algorithm play complementary roles in improving performance. The Nagle algorithm discourages

sending tiny segments when the data to be sent increases in small increments, while the SWS avoidance algorithm discourages small segments resulting from the right window edge advancing in small increments.

#### 3.8.6.2.1. Sender's Algorithm - When to Send Data

A TCP implementation MUST include a SWS avoidance algorithm in the sender (MUST-38).

The Nagle algorithm from Section 3.7.4 additionally describes how to coalesce short segments.

The sender's SWS avoidance algorithm is more difficult than the receiver's, because the sender does not know (directly) the receiver's total buffer space RCV.BUFF. An approach that has been found to work well is for the sender to calculate  $\text{Max}(\text{SND.WND})$ , the maximum send window it has seen so far on the connection, and to use this value as an estimate of RCV.BUFF. Unfortunately, this can only be an estimate; the receiver may at any time reduce the size of RCV.BUFF. To avoid a resulting deadlock, it is necessary to have a timeout to force transmission of data, overriding the SWS avoidance algorithm. In practice, this timeout should seldom occur.

The "usable window" is:

$$U = \text{SND.UNA} + \text{SND.WND} - \text{SND.NXT}$$

i.e., the offered window less the amount of data sent but not acknowledged. If D is the amount of data queued in the sending TCP endpoint but not yet sent, then the following set of rules is recommended.

Send data:

- (1) if a maximum-sized segment can be sent, i.e., if:

$$\min(D, U) \geq \text{Eff.snd.MSS};$$

- (2) or if the data is pushed and all queued data can be sent now, i.e., if:

$$[\text{SND.NXT} = \text{SND.UNA} \text{ and}] \text{ PUSHED and } D \leq U$$

(the bracketed condition is imposed by the Nagle algorithm);

- (3) or if at least a fraction  $F_s$  of the maximum window can be sent, i.e., if:

[SND.NXT = SND.UNA and]

$\min(D, U) \geq F_s * \text{Max}(\text{SND.WND});$

(4) or if the override timeout occurs.

Here  $F_s$  is a fraction whose recommended value is  $1/2$ . The override timeout should be in the range 0.1 – 1.0 seconds. It may be convenient to combine this timer with the timer used to probe zero windows (Section 3.8.6.1).

#### 3.8.6.2.2. Receiver's Algorithm - When to Send a Window Update

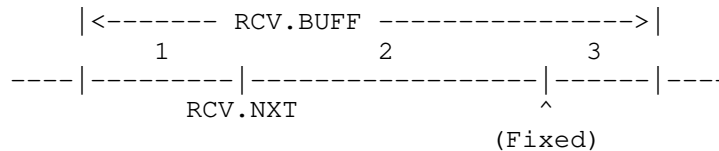
A TCP implementation MUST include a SWS avoidance algorithm in the receiver (MUST-39).

The receiver's SWS avoidance algorithm determines when the right window edge may be advanced; this is customarily known as "updating the window". This algorithm combines with the delayed ACK algorithm (Section 3.8.6.3) to determine when an ACK segment containing the current window will really be sent to the receiver.

The solution to receiver SWS is to avoid advancing the right window edge  $\text{RCV.NXT} + \text{RCV.WND}$  in small increments, even if data is received from the network in small segments.

Suppose the total receive buffer space is  $\text{RCV.BUFF}$ . At any given moment,  $\text{RCV.USER}$  octets of this total may be tied up with data that has been received and acknowledged but that the user process has not yet consumed. When the connection is quiescent,  $\text{RCV.WND} = \text{RCV.BUFF}$  and  $\text{RCV.USER} = 0$ .

Keeping the right window edge fixed as data arrives and is acknowledged requires that the receiver offer less than its full buffer space, i.e., the receiver must specify a  $\text{RCV.WND}$  that keeps  $\text{RCV.NXT} + \text{RCV.WND}$  constant as  $\text{RCV.NXT}$  increases. Thus, the total buffer space  $\text{RCV.BUFF}$  is generally divided into three parts:



- 1 -  $\text{RCV.USER}$  = data received but not yet consumed;
- 2 -  $\text{RCV.WND}$  = space advertised to sender;
- 3 - Reduction = space available but not yet advertised.



The suggested SWS avoidance algorithm for the receiver is to keep RCV.NXT+RCV.WND fixed until the reduction satisfies:

$$\begin{aligned} \text{RCV.BUFF} - \text{RCV.USER} - \text{RCV.WND} &\geq \\ \min( \text{Fr} * \text{RCV.BUFF}, \text{Eff.snd.MSS} ) \end{aligned}$$

where Fr is a fraction whose recommended value is 1/2, and Eff.snd.MSS is the effective send MSS for the connection (see Section 3.7.1). When the inequality is satisfied, RCV.WND is set to RCV.BUFF-RCV.USER.

Note that the general effect of this algorithm is to advance RCV.WND in increments of Eff.snd.MSS (for realistic receive buffers: Eff.snd.MSS < RCV.BUFF/2). Note also that the receiver must use its own Eff.snd.MSS, making the assumption that it is the same as the sender's.

#### 3.8.6.3. Delayed Acknowledgements - When to Send an ACK Segment

A host that is receiving a stream of TCP data segments can increase efficiency in both the Internet and the hosts by sending fewer than one ACK (acknowledgment) segment per data segment received; this is known as a "delayed ACK".

A TCP endpoint SHOULD implement a delayed ACK (SHLD-18), but an ACK should not be excessively delayed; in particular, the delay MUST be less than 0.5 seconds (MUST-40). An ACK SHOULD be generated for at least every second full-sized segment or 2\*RMSS bytes of new data (where RMSS is the MSS specified by the TCP endpoint receiving the segments to be acknowledged, or the default value if not specified) (SHLD-19). Excessive delays on ACKs can disturb the round-trip timing and packet "clocking" algorithms. More complete discussion of delayed ACK behavior is in Section 4.2 of RFC 5681 [8], including recommendations to immediately acknowledge out-of-order segments, segments above a gap in sequence space, or segments that fill all or part of a gap, in order to accelerate loss recovery.

Note that there are several current practices that further lead to a reduced number of ACKs, including generic receive offload (GRO) [73], ACK compression, and ACK decimation [29].

### 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 TCP implementations might use.

#### 3.9.1. User/TCP Interface

The following functional description of user commands to the TCP implementation 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 TCP implementations 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.

Section 3.1 of [54] also identifies primitives provided by TCP, and could be used as an additional reference for implementers.

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.

The user commands described below specify the basic functions the TCP implementation 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 implementation 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 remote socket).
- (b) replies to specific user commands indicating success or various types of failure.

## 3.9.1.1. Open

Format: OPEN (local port, remote socket, active/passive [, timeout] [, DiffServ field] [, security/compartments] [local IP address,] [, options]) -> local connection name

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 remote socket to wait for a particular connection or an unspecified remote 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.

Every passive OPEN call either creates a new connection record in LISTEN state, or it returns an error; it MUST NOT affect any previously created connection record (MUST-41).

A TCP implementation that supports multiple concurrent connections MUST provide an OPEN call that will functionally allow an application to LISTEN on a port while a connection block with the same local port is in SYN-SENT or SYN-RECEIVED state (MUST-42).

On an active OPEN command, the TCP endpoint 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 endpoint will abort the connection. The present global default is five minutes.

The TCP implementation or some component of the operating system will verify the user's authority to open a connection with the specified DiffServ field value or security/compartments. The absence of a DiffServ field value or security/compartments specification in the OPEN call indicates the default values must be used.

TCP will accept incoming requests as matching only if the security/compartments information is exactly the same as that requested in the OPEN call.

The DiffServ field value indicated by the user only impacts outgoing packets, may be altered en route through the network, and has no direct bearing or relation to received packets.

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

The optional "local IP address" parameter MUST be supported to allow the specification of the local IP address (MUST-43). This enables applications that need to select the local IP address used when multihoming is present.

A passive OPEN call with a specified "local IP address" parameter will await an incoming connection request to that address. If the parameter is unspecified, a passive OPEN will await an incoming connection request to any local IP address, and then bind the local IP address of the connection to the particular address that is used.

For an active OPEN call, a specified "local IP address" parameter will be used for opening the connection. If the parameter is unspecified, the host will choose an appropriate local IP address (see RFC 1122 section 3.3.4.2).

If an application on a multihomed host does not specify the local IP address when actively opening a TCP connection, then the TCP implementation MUST ask the IP layer to select a local IP address before sending the (first) SYN (MUST-44). See the function GET\_SRCADDR() in Section 3.4 of RFC 1122.

At all other times, a previous segment has either been sent or received on this connection, and TCP implementations MUST use the same local address that was used in those previous segments (MUST-45).

A TCP implementation MUST reject as an error a local OPEN call for an invalid remote IP address (e.g., a broadcast or multicast address) (MUST-46).

#### 3.9.1.2. Send

Format: SEND (local connection name, buffer address, byte count, PUSH flag (optional), 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. For example, this might be one way for application data to be included in SYN segments. If the calling process is not authorized to use this connection, an error is returned.

A TCP endpoint MAY implement PUSH flags on SEND calls (MAY-15). If PUSH flags are not implemented, then the sending TCP peer: (1) MUST NOT buffer data indefinitely (MUST-60), and (2) MUST set the PSH bit in the last buffered segment (i.e., when there is no more queued data to be sent) (MUST-61). The remaining description below assumes the PUSH flag is supported on SEND calls.

If the PUSH flag is set, the application intends the data to be transmitted promptly to the receiver, and the PUSH bit will be set in the last TCP segment created from the buffer.

The PSH bit is not a record marker and is independent of segment boundaries. The transmitter SHOULD collapse successive bits when it packetizes data, to send the largest possible segment (SHLD-27).

If the PUSH flag is not set, the data may be combined with data from subsequent SENDs for transmission efficiency. When an application issues a series of SEND calls without setting the PUSH flag, the TCP implementation MAY aggregate the data internally without sending it (MAY-16). Note that when the Nagle algorithm is in use, TCP implementations may buffer the data before sending, without regard to the PUSH flag (see Section 3.7.4).

An application program is logically required to set the PUSH flag in a SEND call whenever it needs to force delivery of the data to avoid a communication deadlock. However, a TCP implementation SHOULD send a maximum-sized segment whenever possible (SHLD-28), to improve performance (see Section 3.8.6.2.1).

New applications SHOULD NOT set the URGENT flag [40] due to implementation differences and middlebox issues (SHLD-13).

If the URGENT flag is set, segments sent to the destination TCP peer will have the urgent pointer set. The receiving TCP peer 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 implementation 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 remote socket was specified in the OPEN, but the connection is established (e.g., because a LISTENing connection has become specific due to a remote segment arriving for the local socket), then the designated buffer is sent to the implied remote socket. Users who make use of OPEN with an unspecified remote socket can make use of SEND without ever explicitly knowing the remote socket address.

However, if a SEND is attempted before the remote socket becomes specified, an error will be returned. Users can use the STATUS call to determine the status of the connection. Some TCP implementations 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 endpoint 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 endpoint. 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 endpoint. 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 that should be returned to the calling process.

#### 3.9.1.3. Receive

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

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.

A TCP receiver MAY pass a received PSH flag to the application layer via the PUSH flag in the interface (MAY-17), but it is not required (this was clarified in RFC 1122 section 4.2.2.2). The remainder of text describing the RECEIVE call below assumes that passing the PUSH indication is supported.

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 endpoint allocate buffer storage, or the TCP endpoint might share a ring buffer with the user.

#### 3.9.1.4. 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 remote peer 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 peer gives up.

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

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

Close also implies push function.

#### 3.9.1.5. 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,
- remote socket,
- local connection name,
- receive window,
- send window,
- connection state,
- number of buffers awaiting acknowledgment,
- number of buffers pending receipt,
- urgent state,
- DiffServ field value,
- 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.

#### 3.9.1.6. 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 remote TCP peer 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.

#### 3.9.1.7. Flush

Some TCP implementations have included a FLUSH call, which will empty the TCP send queue of any data that the user has issued SEND calls for but is still to the right of the current send window. That is, it flushes as much queued send data as possible without losing sequence number synchronization. The FLUSH call MAY be implemented (MAY-14).

#### 3.9.1.8. Asynchronous Reports

There MUST be a mechanism for reporting soft TCP error conditions to the application (MUST-47). Generically, we assume this takes the form of an application-supplied ERROR\_REPORT routine that may be upcalled asynchronously from the transport layer:

- ERROR\_REPORT(local connection name, reason, subreason)

The precise encoding of the reason and subreason parameters is not specified here. However, the conditions that are reported asynchronously to the application MUST include:

- \* ICMP error message arrived (see Section 3.9.2.2 for description of handling each ICMP message type, since some message types need to be suppressed from generating reports to the application)
- \* Excessive retransmissions (see Section 3.8.3)
- \* Urgent pointer advance (see Section 3.8.5)

However, an application program that does not want to receive such ERROR\_REPORT calls SHOULD be able to effectively disable these calls (SHLD-20).

#### 3.9.1.9. Set Differentiated Services Field (IPv4 TOS or IPv6 Traffic Class)

The application layer MUST be able to specify the Differentiated Services field for segments that are sent on a connection (MUST-48). The Differentiated Services field includes the 6-bit Differentiated Services Code Point (DSCP) value. It is not required, but the application SHOULD be able to change the Differentiated Services field during the connection lifetime (SHLD-21). TCP implementations SHOULD pass the current Differentiated Services field value without change to the IP layer, when it sends segments on the connection (SHLD-22).

The Differentiated Services field will be specified independently in each direction on the connection, so that the receiver application will specify the Differentiated Services field used for ACK segments.

TCP implementations MAY pass the most recently received Differentiated Services field up to the application (MAY-9).

### 3.9.2. TCP/Lower-Level Interface

The TCP endpoint calls on a lower level protocol module to actually send and receive information over a network. The two current standard Internet Protocol (IP) versions layered below TCP are IPv4 [1] and IPv6 [13].

If the lower level protocol is IPv4 it provides arguments for a type of service (used within the Differentiated Services field) and for a time to live. TCP uses the following settings for these parameters:

DiffServ field: The IP header value for the DiffServ field is given by the user. This includes the bits of the DiffServ Code Point (DSCP).

Time to Live (TTL): The TTL value used to send TCP segments MUST be configurable (MUST-49).

- Note that RFC 793 specified one minute (60 seconds) as a constant for the TTL, because the assumed maximum segment lifetime was two minutes. This was intended to explicitly ask that a segment be destroyed if it cannot be delivered by the internet system within one minute. RFC 1122 changed this specification to require that the TTL be configurable.
- Note that the DiffServ field is permitted to change during a connection (Section 4.2.4.2 of RFC 1122). However, the application interface might not support this ability, and the application does not have knowledge about individual TCP segments, so this can only be done on a coarse granularity, at best. This limitation is further discussed in RFC 7657 (sec 5.1, 5.3, and 6) [51]. Generally, an application SHOULD NOT change the DiffServ field value during the course of a connection (SHLD-23).

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.

When received options are passed up to TCP from the IP layer, a TCP implementation MUST ignore options that it does not understand (MUST-50).

A TCP implementation MAY support the Time Stamp (MAY-10) and Record Route (MAY-11) options.

#### 3.9.2.1. Source Routing

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.

An application MUST be able to specify a source route when it actively opens a TCP connection (MUST-51), and this MUST take precedence over a source route received in a datagram (MUST-52).

When a TCP connection is OPENed passively and a packet arrives with a completed IP Source Route option (containing a return route), TCP implementations MUST save the return route and use it for all segments sent on this connection (MUST-53). If a different source route arrives in a later segment, the later definition SHOULD override the earlier one (SHLD-24).

#### 3.9.2.2. ICMP Messages

TCP implementations MUST act on an ICMP error message passed up from the IP layer, directing it to the connection that created the error (MUST-54). The necessary demultiplexing information can be found in the IP header contained within the ICMP message.

This applies to ICMPv6 in addition to IPv4 ICMP.

[36] contains discussion of specific ICMP and ICMPv6 messages classified as either "soft" or "hard" errors that may bear different responses. Treatment for classes of ICMP messages is described below:

##### Source Quench

TCP implementations MUST silently discard any received ICMP Source Quench messages (MUST-55). See [11] for discussion.

##### Soft Errors

For IPv4 ICMP these include: Destination Unreachable -- codes 0, 1, 5; Time Exceeded -- codes 0, 1; and Parameter Problem.

For ICMPv6 these include: Destination Unreachable -- codes 0, 3; Time Exceeded -- codes 0, 1; and Parameter Problem -- codes 0, 1, 2.

Since these Unreachable messages indicate soft error conditions, TCP implementations **MUST NOT** abort the connection (MUST-56), and it **SHOULD** make the information available to the application (SHLD-25).

#### Hard Errors

For ICMP these include Destination Unreachable -- codes 2-4.

These are hard error conditions, so TCP implementations **SHOULD** abort the connection (SHLD-26). [36] notes that some implementations do not abort connections when an ICMP hard error is received for a connection that is in any of the synchronized states.

Note that [36] section 4 describes widespread implementation behavior that treats soft errors as hard errors during connection establishment.

#### 3.9.2.3. Source Address Validation

RFC 1122 requires addresses to be validated in incoming SYN packets:

An incoming SYN with an invalid source address **MUST** be ignored either by TCP or by the IP layer (MUST-63) (Section 3.2.1.3 of [20]).

A TCP implementation **MUST** silently discard an incoming SYN segment that is addressed to a broadcast or multicast address (MUST-57).

This prevents connection state and replies from being erroneously generated, and implementers should note that this guidance is applicable to all incoming segments, not just SYNs, as specifically indicated in RFC 1122.

#### 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 endpoint 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 endpoint 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 in this document are identified by 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 " $=<$ " 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 connection stays in the same state.

### 3.10.1. 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, remote socket, DiffServ field, security/compartments, and user timeout information. Note that some parts of the remote 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 DiffServ value requested are allowed for this user, if not return "error: DiffServ value not allowed" or "error: security/compartments not allowed." If passive enter the LISTEN state and return. If active and the remote socket is unspecified, return "error: remote socket unspecified"; if active and the remote 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 the OPEN call is active and the remote 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 the remote socket was not specified, then return "error: remote 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".

### 3.10.2. 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 remote 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 the remote socket was not specified, then return "error: remote 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.

### 3.10.3. 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 endpoint 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 data already on hand, but not yet delivered to the user. If no text is awaiting delivery, the RECEIVE will get an "error: connection closing" response. Otherwise, any remaining data can be used to satisfy the RECEIVE.

## CLOSING STATE

## LAST-ACK STATE

## TIME-WAIT STATE

- Return "error: connection closing".

## 3.10.4. 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 an "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".

### 3.10.5. 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:
  - o <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.

#### 3.10.6. 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.

## 3.10.7. SEGMENT ARRIVES

## 3.10.7.1. CLOSED State

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 endpoint 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.

## 3.10.7.2. LISTEN State

If the state is LISTEN then

first check for an RST

- 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. 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:

- o <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.
  - o <SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>
- 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:
  - o <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 remote socket was not fully specified), then the unspecified fields should be filled in now.

fourth other data or control

- This should not be reached. Drop the segment and return. Any other control or data-bearing segment (not containing SYN) must have an ACK and thus would have been discarded by the ACK processing in the second step, unless it was first discarded by RST checking in the first step.

### 3.10.7.3. SYN-SENT State

If the state is SYN-SENT then

first check the ACK bit

- If the ACK bit is set
  - o 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>
  - o and discard the segment. Return.

- o If `SND.UNA < SEG.ACK =< SND.NXT` then the ACK is acceptable. Some deployed TCP code has used the check `SEG.ACK == SND.NXT` (using `"=="` rather than `"=<"`, but this is not appropriate when the stack is capable of sending data on the SYN, because the TCP peer may not accept and acknowledge all of the data on the SYN.

second check the RST bit

- If the RST bit is set
  - o A potential blind reset attack is described in RFC 5961 [9]. The mitigation described in that document has specific applicability explained therein, and is not a substitute for cryptographic protection (e.g. IPsec or TCP-AO). A TCP implementation that supports the RFC 5961 mitigation SHOULD first check that the sequence number exactly matches `RCV.NXT` prior to executing the action in the next paragraph.
  - o 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

- If the security/compartments in the segment does not exactly match the security/compartments in the TCB, send a reset
  - o If there is an ACK
    - + `<SEQ=SEG.ACK><CTL=RST>`
  - o Otherwise
    - + `<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>`
- 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 the segment did not contain a RST.



- If the SYN bit is on and the security/compartiment is 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 that 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
    - o <SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>
  - and send it. Data or controls that were queued for transmission MAY be included. Some TCP implementations suppress sending this segment when the received segment contains data that will anyways generate an acknowledgement in the later processing steps, saving this extra acknowledgement of the SYN from being sent. If there are other controls or text in the segment then continue processing at the sixth step under Section 3.10.7.4 where the URG bit is checked, otherwise return.
  - Otherwise enter SYN-RECEIVED, form a SYN,ACK segment
    - o <SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>
  - and send it. Set the variables:
    - o 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.
- Note that it is legal to send and receive application data on SYN segments (this is the "text in the segment" mentioned above. There has been significant misinformation and misunderstanding of this topic historically. Some firewalls and security devices consider this suspicious. However, the capability was used in T/TCP [22] and is used in TCP Fast Open (TFO) [49], so is important for implementations and network devices to permit.

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

## 3.10.7.4. Other States

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

- o 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 are processed.
- o In general, the processing of received segments MUST be implemented to aggregate ACK segments whenever possible (MUST-58). For example, if the TCP endpoint is processing a series of queued segments, it MUST process them all before sending any ACK segments (MUST-59).
- o 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

- o In implementing sequence number validation as described here, please note Appendix A.2.
- o If the RCV.WND is zero, no segments will be acceptable, but special allowance should be made to accept valid ACKs, URGs and RSTs.
- o 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>
- o After sending the acknowledgment, drop the unacceptable segment and return.
- o Note that for the TIME-WAIT state, there is an improved algorithm described in [41] for handling incoming SYN segments, that utilizes timestamps rather than relying on the sequence number check described here. When the improved algorithm is implemented, the logic above is not applicable for incoming SYN segments with timestamp options, received on a connection in the TIME-WAIT state.
- o 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 (SHLD-31).
- second check the RST bit,

- o RFC 5961 [9] section 3 describes a potential blind reset attack and optional mitigation approach. This does not provide a cryptographic protection (e.g. as in IPsec or TCP-AO), but can be applicable in situations described in RFC 5961. For stacks implementing the RFC 5961 protection, the three checks below apply, otherwise processing for these states is indicated further below.

- + 1) If the RST bit is set and the sequence number is outside the current receive window, silently drop the segment.
- + 2) If the RST bit is set and the sequence number exactly matches the next expected sequence number (RCV.NXT), then TCP endpoints MUST reset the connection in the manner prescribed below according to the connection state.
- + 3) If the RST bit is set and the sequence number does not exactly match the next expected sequence value, yet is within the current receive window, TCP endpoints MUST send an acknowledgement (challenge ACK):

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

After sending the challenge ACK, TCP endpoints MUST drop the unacceptable segment and stop processing the incoming packet further. Note that RFC 5961 and Errata ID 4772 contain additional considerations for ACK throttling in an implementation.

- o 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, the retransmission queue should be flushed. And in the active OPEN case, enter the CLOSED state and delete the TCB, and return.

- o 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.

o 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

o SYN-RECEIVED

- + If the security/compartments in the segment does not exactly match the security/compartments in the TCB then send a reset, and return.

o ESTABLISHED

FIN-WAIT-1

FIN-WAIT-2

CLOSE-WAIT

CLOSING

LAST-ACK

TIME-WAIT

- + If the security/compartments in the segment does not exactly match the security/compartments 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.

- o Note this check is placed following the sequence check to prevent a segment from an old connection between these port numbers with a different security from causing an abort of the current connection.
- fourth, check the SYN bit,
  - o SYN-RECEIVED
    - + If the connection was initiated with a passive OPEN, then return this connection to the LISTEN state and return. Otherwise, handle per the directions for synchronized states below.

ESTABLISHED STATE

FIN-WAIT STATE-1

FIN-WAIT STATE-2

CLOSE-WAIT STATE

CLOSING STATE

LAST-ACK STATE

TIME-WAIT STATE

- + If the SYN bit is set in these synchronized states, it may be either a legitimate new connection attempt (e.g. in the case of TIME-WAIT), an error where the connection should be reset, or the result of an attack attempt, as described in RFC 5961 [9]. For the TIME-WAIT state, new connections can be accepted if the timestamp option is used and meets expectations (per [41]). For all other cases, RFC 5961 provides a mitigation with applicability to some situations, though there are also alternatives that offer cryptographic protection (see Section 7). RFC 5961 recommends that in these synchronized states, if the SYN bit is set, irrespective of the sequence number, TCP endpoints MUST send a "challenge ACK" to the remote peer:
  - + <SEQ=SEND.NXT><ACK=RCV.NXT><CTL=ACK>
- + After sending the acknowledgement, TCP implementations MUST drop the unacceptable segment and stop processing further. Note that RFC 5961 and Errata ID 4772 contain additional ACK throttling notes for an implementation.

- + For implementations that do not follow RFC 5961, the original RFC 793 behavior follows in this paragraph. 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,
  - o if the ACK bit is off drop the segment and return
  - o if the ACK bit is on
- + RFC 5961 [9] section 5 describes a potential blind data injection attack, and mitigation that implementations MAY choose to include (MAY-12). TCP stacks that implement RFC 5961 MUST add an input check that the ACK value is acceptable only if it is in the range of ((SND.UNA - MAX.SND.WND) =< SEG.ACK =< SND.NXT). All incoming segments whose ACK value doesn't satisfy the above condition MUST be discarded and an ACK sent back. The new state variable MAX.SND.WND is defined as the largest window that the local sender has ever received from its peer (subject to window scaling) or may be hard-coded to a maximum permissible window value. When the ACK value is acceptable, the processing per-state below applies:
- + 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:
    - $\text{SND.WND} \leftarrow \text{SEG.WND}$
    - $\text{SND.WL1} \leftarrow \text{SEG.SEQ}$
    - $\text{SND.WL2} \leftarrow \text{SEG.ACK}$
  - \* If the segment acknowledgment is not acceptable, form a reset segment,

- <SEQ=SEG.ACK><CTL=RST>
- \* and send it.
- + ESTABLISHED STATE
  - \* If `SND.UNA < SEG.ACK =< SND.NXT` then, set `SND.UNA <- SEG.ACK`. Any segments on the retransmission queue that are thereby entirely acknowledged are removed. Users should receive positive acknowledgments for buffers that have been SENT and fully acknowledged (i.e., SEND buffer should be returned with "ok" response). If the ACK is a duplicate (`SEG.ACK =< SND.UNA`), it can be ignored. If the ACK acks something not yet sent (`SEG.ACK > SND.NXT`) then send an ACK, drop the segment, and return.
  - \* If `SND.UNA =< SEG.ACK =< SND.NXT`, the send window should be updated. If (`SND.WL1 < SEG.SEQ` or (`SND.WL1 = SEG.SEQ` and `SND.WL2 =< SEG.ACK`)), set `SND.WND <- SEG.WND`, set `SND.WL1 <- SEG.SEQ`, and set `SND.WL2 <- 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 the FIN segment 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,
  - o 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.
  - o 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,
  - o ESTABLISHED STATE

## FIN-WAIT-1 STATE

## FIN-WAIT-2 STATE

- + Once in the ESTABLISHED state, it is possible to deliver segment data to user RECEIVE buffers. Data from segments can be moved into buffers until either the buffer is full or the segment is empty. If the segment empties and carries a PUSH flag, then the user is informed, when the buffer is returned, that a PUSH has been received.
- + When the TCP endpoint takes responsibility for delivering the data to the user it must also acknowledge the receipt of the data.
- + Once the TCP endpoint 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.
- + A TCP implementation MAY send an ACK segment acknowledging RCV.NXT when a valid segment arrives that is in the window but not at the left window edge (MAY-13).
- + Please note the window management suggestions in Section 3.8.
- + 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.

## o 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,
  - o 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.
  - o 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.

## 3.10.8. Timeouts

## 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.

## 4. Glossary

## 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.

## 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 network layer address of the endpoint intended to receive a segment.

## FIN

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

- flush**  
To remove all of the contents (data or segments) from a store (buffer or queue).
- fragment**  
A portion of a logical unit of data, in particular an internet fragment is a portion of an internet datagram.
- 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.
- internet address**  
A network layer address.
- internet datagram**  
A unit of data exchanged between internet hosts, together with the internet header that allows the datagram to be routed from source to destination.
- internet fragment**  
A portion of the data of an internet datagram with an internet header.
- IP**  
Internet Protocol. See [1] and [13].
- 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.

**left sequence**

This is the next sequence number to be acknowledged by the data receiving TCP endpoint (or the lowest currently unacknowledged sequence number) and is sometimes referred to as the left edge of the send window.

**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.

**packet**

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

**port**

The portion of a connection identifier used for demultiplexing connections at an endpoint.

**process**

A program in execution. A source or destination of data from the point of view of the TCP endpoint 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 endpoint is expecting to receive.

**receive window**

This represents the sequence numbers the local (receiving) TCP endpoint is willing to receive. Thus, the local TCP endpoint 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 this range are considered duplicates or injection attacks 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.

**SEG.ACK**

segment acknowledgment

**SEG.LEN**

segment length

**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 that 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 endpoint 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 that the remote (receiving) TCP endpoint is willing to receive. It is the value of the window field specified in segments from the remote (data receiving) TCP endpoint. The range of new sequence numbers that may be emitted by a TCP implementation 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 (or socket number, or socket address, or socket identifier)**

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

**Source Address**

The network layer address of the sending endpoint.



## 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.

## TCP

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

## TOS

Type of Service, an obsoleted IPv4 field. The same header bits currently are used for the Differentiated Services field [4] containing the Differentiated Services Code Point (DSCP) value and the 2-bit ECN codepoint [6].

## Type of Service

See "TOS".

## 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 by 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 that indicates the data octet associated with the sending user's urgent call.

## 5. 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 some informational text with background and rationale may not have 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, 1571, 1572, 2297, 2298, 2748, 2749, 2934, 3213, 3300, 3301, 6222). Some errata were not applicable due to other changes (Errata IDs: 572, 575, 1565, 1569, 2296, 3305, 3602).

Changes to the specification of the Urgent Pointer described in RFCs 1011, 1122, and 6093 were incorporated. See RFC 6093 for detailed discussion of why these changes were necessary.

The discussion of the RTO from RFC 793 was updated to refer to RFC 6298. The RFC 1122 text on the RTO originally replaced the 793 text, however, RFC 2988 should have updated 1122, and has subsequently been obsoleted by 6298.

RFC 1011 [19] contains a number of comments about RFC 793, including some needed changes to the TCP specification. These are expanded in RFC 1122, which contains a collection of other changes and clarifications to RFC 793. The normative items impacting the protocol have been incorporated here, though some historically useful implementation advice and informative discussion from RFC 1122 is not included here. The present document updates RFC 1011, since this is now the TCP specification rather than RFC 793, and the comments noted in 1011 have been incorporated.

RFC 1122 contains more than just TCP requirements, so this document can't obsolete RFC 1122 entirely. It is only marked as "updating" 1122, however, it should be understood to effectively obsolete all of the RFC 1122 material on TCP.

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.

A note based on RFC 6429 was added to explicitly clarify that system resource management concerns allow connection resources to be reclaimed. RFC 6429 is obsoleted in the sense that this clarification has been reflected in this update to the base TCP specification now.

The description of congestion control implementation was added, based on the set of documents that are IETF BCP or Standards Track on the topic, and the current state of common implementations.

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.

This document started as draft-eddy-rfc793bis-00, that was merely a proposal and rough plan for updating RFC 793.

The -01 revision of this draft-eddy-rfc793bis 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 draft-eddy-rfc793bis incorporates errata that have been verified:

Errata ID 573: Reported by Bob Braden (note: This errata report 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

Errata ID 3305: 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.

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

The -03 revision of draft-eddy-rfc793bis 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 draft-eddy-rfc793bis includes the ISN generation changes from RFC 6528.

The -05 revision of draft-eddy-rfc793bis incorporates MSS requirements and definitions from RFC 879 [17], 1122, and 6691, as well as option-handling requirements from RFC 1122.

The -00 revision of draft-ietf-tcpm-rfc793bis incorporates several additional clarifications and updates to the section on segmentation, many of which are based on feedback from Joe Touch improving from the initial text on this in the previous revision.

The -01 revision incorporates the change to Reserved bits due to ECN, as well as many other changes that come from RFC 1122.

The -02 revision has small formatting modifications in order to address xml2rfc warnings about long lines. It was a quick update to avoid document expiration. TCPM working group discussion in 2015 also indicated that we should not try to add sections on implementation advice or similar non-normative information.

The -03 revision incorporates more content from RFC 1122: Passive OPEN Calls, Time-To-Live, Multihoming, IP Options, ICMP messages, Data Communications, When to Send Data, When to Send a Window Update, Managing the Window, Probing Zero Windows, When to Send an ACK Segment. The section on data communications was re-organized into clearer subsections (previously headings were embedded in the 793 text), and windows management advice from 793 was removed (as reviewed by TCPM working group) in favor of the 1122 additions on SWS, ZWP, and related topics.

The -04 revision includes reference to RFC 6429 on the ZWP condition, RFC1122 material on TCP Connection Failures, TCP Keep-Alives, Acknowledging Queued Segments, and Remote Address Validation. RTO computation is referenced from RFC 6298 rather than RFC 1122.

The -05 revision includes the requirement to implement TCP congestion control with recommendation to implement ECN, the RFC 6633 update to 1122, which changed the requirement on responding to source quench ICMP messages, and discussion of ICMP (and ICMPv6) soft and hard errors per RFC 5461 (ICMPv6 handling for TCP doesn't seem to be mentioned elsewhere in standards track).

The -06 revision includes an appendix on "Other Implementation Notes" to capture widely-deployed fundamental features that are not contained in the RFC series yet. It also added mention of RFC 6994 and the IANA TCP parameters registry as a reference. It includes references to RFC 5961 in appropriate places. The references to TOS were changed to DiffServ field, based on reflecting RFC 2474 as well as the IPv6 presence of traffic class (carrying DiffServ field) rather than TOS.

The -07 revision includes reference to RFC 6191, updated security considerations, discussion of additional implementation considerations, and clarification of data on the SYN.

The -08 revision includes changes based on:

- describing treatment of reserved bits (following TCPM mailing list thread from July 2014 on "793bis item - reserved bit behavior"
- addition a brief TCP key concepts section to make up for not including the outdated section 2 of RFC 793
- changed "TCP" to "host" to resolve conflict between 1122 wording on whether TCP or the network layer chooses an address when multihomed
- fixed/updated definition of options in glossary
- moved note on aggregating ACKs from 1122 to a more appropriate location
- resolved notes on IP precedence and security/compartments

added implementation note on sequence number validation  
added note that PUSH does not apply when Nagle is active  
added 1122 content on asynchronous reports to replace 793 section  
on TCP to user messages

The -09 revision fixes section numbering problems.

The -10 revision includes additions to the security considerations based on comments from Joe Touch, and suggested edits on RST/FIN notification, RFC 2525 reference, and other edits suggested by Yuchung Cheng, as well as modifications to DiffServ text from Yuchung Cheng and Gorry Fairhurst.

The -11 revision includes a start at identifying all of the requirements text and referencing each instance in the common table at the end of the document.

The -12 revision completes the requirement language indexing started in -11 and adds necessary description of the PUSH functionality that was missing.

The -13 revision contains only changes in the inline editor notes.

The -14 revision includes updates with regard to several comments from the mailing list, including editorial fixes, adding IANA considerations for the header flags, improving figure title placement, and breaking up the "Terminology" section into more appropriately titled subsections.

The -15 revision has many technical and editorial corrections from Gorry Fairhurst's review, and subsequent discussion on the TCPM list, as well as some other collected clarifications and improvements from mailing list discussion.

The -16 revision addresses several discussions that rose from additional reviews and follow-up on some of Gorry Fairhurst's comments from revision 14.

The -17 revision includes errata 6222 from Charles Deng, update to the key words boilerplate, updated description of the header flags registry changes, and clarification about connections rather than users in the discussion of OPEN calls.

The -18 revision includes editorial changes to the IANA considerations, based on comments from Richard Scheffenegger at the IETF 108 TCPM virtual meeting.

The -19 revision includes editorial changes from Errata 6281 and 6282 reported by Merlin Buge. It also includes WGLC changes noted by Mohamed Boucadair, Rahul Jadhav, Praveen Balasubramanian, Matt Olson, Yi Huang, Joe Touch, and Juhamatti Kuusisaari.

The -20 revision includes text on congestion control based on mailing list and meeting discussion, put together in its final form by Markku Kojo. It also clarifies that SACK, WS, and TS options are recommended for high performance, but not needed for basic interoperability. It also clarifies that the length field is required for new TCP options.

The -21 revision includes slight changes to the header diagram for compatibility with tooling, from Stephen McQuistin, clarification on the meaning of idle connections from Yuchung Cheng, Neal Cardwell, Michael Scharf, and Richard Scheffenegger, editorial improvements from Markku Kojo, notes that some stacks suppress extra acknowledgments of the SYN when SYN-ACK carries data from Richard Scheffenegger, and adds MAY-18 numbering based on note from Jonathan Morton.

The -22 revision includes small clarifications on terminology (might versus may) and IPv6 extension headers versus IPv4 options, based on comments from Gorry Fairhurst.

The -23 revision has a fix to indentation from Michael Tuexen and idnits issues addressed from Michael Scharf.

The -24 revision incorporates changes after Martin Duke's AD review, including further feedback on those comments from Yuchung Cheng and Joe Touch. Important changes for review include (1) removal of the need to check for the PUSH flag when evaluating the SWS override timer expiration, (2) clarification about receding urgent pointer, and (3) de-duplicating handling of the RST checking between step 4 and step 1.

The -25 revision incorporates changes based on the GENART review from Francis Dupont, SECDIR review from Kyle Rose, and OPSDIR review from Sarah Banks.

The -26 revision incorporates changes stemming from the IESG reviews, and INTDIR review from Bernie Volz.

The -27 revision fixes a few small editorial incompatibilities that Stephen McQuistin found related to automated code generation.

The -28 revision addresses some COMMENTS from Ben Kaduk's IESG review.

Some other suggested changes that will not be incorporated in this 793 update unless TCPM consensus changes with regard to scope are:

1. Tony Sabatini's suggestion for describing DO field
2. Per discussion with Joe Touch (TAPS list, 6/20/2015), the description of the API could be revisited
3. Reducing the R2 value for SYNs has been suggested as a possible topic for future consideration.

Early in the process of updating RFC 793, Scott Brim mentioned that this should include a PERPASS/privacy review. This may be something for the chairs or AD to request during WGLC or IETF LC.

## 6. IANA Considerations

In the "Transmission Control Protocol (TCP) Header Flags" registry, IANA is asked to make several changes described in this section.

RFC 3168 originally created this registry, but only populated it with the new bits defined in RFC 3168, neglecting the other bits that had previously been described in RFC 793 and other documents. Bit 7 has since also been updated by RFC 8311.

The "Bit" column is renamed below as the "Bit Offset" column, since it references each header flag's offset within the 16-bit aligned view of the TCP header in Figure 1. The bits in offsets 0 through 4 are the TCP segment Data Offset field, and not header flags.

IANA should add a column for "Assignment Notes".

IANA should assign values indicated below.



## TCP Header Flags

Bit Notes Offset	Name	Reference	Assignment
---	----	-----	-----
4	Reserved for future use	(this document)	
5	Reserved for future use	(this document)	
6	Reserved for future use	(this document)	
7	Reserved for future use	[RFC8311]	[1]
8	CWR (Congestion Window Reduced)	[RFC3168]	
9	ECE (ECN-Echo)	[RFC3168]	
10	Urgent Pointer field is significant (URG)	(this document)	
11	Acknowledgment field is significant (ACK)	(this document)	
12	Push Function (PSH)	(this document)	
13	Reset the connection (RST)	(this document)	
14	Synchronize sequence numbers (SYN)	(this document)	
15	No more data from sender (FIN)	(this document)	

## FOOTNOTES:

[1] Previously used by Historic [RFC3540] as NS (Nonce Sum).

This TCP Header Flags registry should also be moved to a sub-registry under the global "Transmission Control Protocol (TCP) Parameters registry (<https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml>).

The registry's Registration Procedure should remain Standards Action, but the Reference can be updated to this document, and the Note removed.

## 7. Security and Privacy Considerations

The TCP design includes only rudimentary security features that improve the robustness and reliability of connections and application data transfer, but there are no built-in cryptographic capabilities to support any form of confidentiality, authentication, or other typical security functions. Non-cryptographic enhancements (e.g. [9]) have been developed to improve robustness of TCP connections to particular types of attacks, but the applicability and protections of non-cryptographic enhancements are limited (e.g. see section 1.1 of [9]). Applications typically utilize lower-layer (e.g. IPsec) and upper-layer (e.g. TLS) protocols to provide security and privacy for TCP connections and application data carried in TCP. Methods based on TCP options have been developed as well, to support some security capabilities.

In order to fully provide confidentiality, integrity protection, and authentication for TCP connections (including their control flags) IPsec is the only current effective method. For integrity protection and authentication, the TCP Authentication Option (TCP-AO) [39] is available, with a proposed extension to also provide confidentiality for the segment payload. Other methods discussed in this section may provide confidentiality or integrity protection for the payload, but for the TCP header only cover either a subset of the fields (e.g. tcpcrypt [57]) or none at all (e.g. TLS). Other security features that have been added to TCP (e.g. ISN generation, sequence number checks, and others) are only capable of partially hindering attacks.

Applications using long-lived TCP flows have been vulnerable to attacks that exploit the processing of control flags described in earlier TCP specifications [34]. TCP-MD5 was a commonly implemented TCP option to support authentication for some of these connections, but had flaws and is now deprecated. TCP-AO provides a capability to protect long-lived TCP connections from attacks, and has superior properties to TCP-MD5. It does not provide any privacy for application data, nor for the TCP headers.

The "tcpcrypt" [57] Experimental extension to TCP provides the ability to cryptographically protect connection data. Metadata aspects of the TCP flow are still visible, but the application stream is well-protected. Within the TCP header, only the urgent pointer and FIN flag are protected through tcpcrypt.

The TCP Roadmap [50] includes notes about several RFCs related to TCP security. Many of the enhancements provided by these RFCs have been integrated into the present document, including ISN generation, mitigating blind in-window attacks, and improving handling of soft errors and ICMP packets. These are all discussed in greater detail in the referenced RFCs that originally described the changes needed to earlier TCP specifications. Additionally, see RFC 6093 [40] for discussion of security considerations related to the urgent pointer field, that has been deprecated.

Since TCP is often used for bulk transfer flows, some attacks are possible that abuse the TCP congestion control logic. An example is "ACK-division" attacks. Updates that have been made to the TCP congestion control specifications include mechanisms like Appropriate Byte Counting (ABC) [30] that act as mitigations to these attacks.

Other attacks are focused on exhausting the resources of a TCP server. Examples include SYN flooding [33] or wasting resources on non-progressing connections [42]. Operating systems commonly implement mitigations for these attacks. Some common defenses also utilize proxies, stateful firewalls, and other technologies outside the end-host TCP implementation.

The concept of a protocol's "wire image" is described in RFC 8546 [56], which describes how TCP's cleartext headers expose more metadata to nodes on the path than is strictly required to route the packets to their destination. On-path adversaries may be able to leverage this metadata. Lessons learned in this respect from TCP have been applied in the design of newer transports like QUIC [60]. Additionally, based partly on experiences with TCP and its extensions, there are considerations that might be applicable for future TCP extensions and other transports that the IETF has documented in RFC 9065 [61], along with IAB recommendations in RFC 8558 [58] and [68].

There are also methods of "fingerprinting" that can be used to infer the host TCP implementation (operating system) version or platform information. These collect observations of several aspects such as the options present in segments, the ordering of options, the specific behaviors in the case of various conditions, packet timing, packet sizing, and other aspects of the protocol that are left to be determined by an implementer, and can use those observations to identify information about the host and implementation.

## 8. 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, over the course of work on this document:

Michael Scharf

Yoshifumi Nishida

Pasi Sarolahti

Michael Tuexen

During the discussions of this work on the TCPM mailing list, in working group meetings, and via area reviews, helpful comments, critiques, and reviews were received from (listed alphabetically by last name): Praveen Balasubramanian, David Borman, Mohamed Boucadair, Bob Briscoe, Neal Cardwell, Yuchung Cheng, Martin Duke, Francis Dupont, Ted Faber, Gorrry Fairhurst, Fernando Gont, Rodney Grimes, Yi Huang, Rahul Jadhav, Markku Kojo, Mike Kosek, Juhamatti Kuusisaari, Kevin Lahey, Kevin Mason, Matt Mathis, Stephen McQuistin, Jonathan Morton, Matt Olson, Tommy Pauly, Tom Petch, Hagen Paul Pfeifer, Kyle Rose, Anthony Sabatini, Michael Scharf, Greg Skinner, Joe Touch, Michael Tuexen, Reji Varghese, Bernie Volz, Tim Wicinski, Lloyd Wood, and Alex Zimmermann.

Joe Touch provided additional help in clarifying the description of segment size parameters and PMTUD/PLPMTUD recommendations. Markku Kojo helped put together the text in the section on TCP Congestion Control.

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, Charles Deng, Merlin Buge.

## 9. References

### 9.1. Normative References

- [1] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <<https://www.rfc-editor.org/info/rfc791>>.
- [2] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990, <<https://www.rfc-editor.org/info/rfc1191>>.
- [3] 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>>.
- [4] 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, DOI 10.17487/RFC2474, December 1998, <<https://www.rfc-editor.org/info/rfc2474>>.

- [5] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, DOI 10.17487/RFC2914, September 2000, <<https://www.rfc-editor.org/info/rfc2914>>.
- [6] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<https://www.rfc-editor.org/info/rfc3168>>.
- [7] Floyd, S. and M. Allman, "Specifying New Congestion Control Algorithms", BCP 133, RFC 5033, DOI 10.17487/RFC5033, August 2007, <<https://www.rfc-editor.org/info/rfc5033>>.
- [8] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<https://www.rfc-editor.org/info/rfc5681>>.
- [9] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", RFC 5961, DOI 10.17487/RFC5961, August 2010, <<https://www.rfc-editor.org/info/rfc5961>>.
- [10] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <<https://www.rfc-editor.org/info/rfc6298>>.
- [11] Gont, F., "Deprecation of ICMP Source Quench Messages", RFC 6633, DOI 10.17487/RFC6633, May 2012, <<https://www.rfc-editor.org/info/rfc6633>>.
- [12] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [13] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <<https://www.rfc-editor.org/info/rfc8200>>.
- [14] McCann, J., Deering, S., Mogul, J., and R. Hinden, Ed., "Path MTU Discovery for IP version 6", STD 87, RFC 8201, DOI 10.17487/RFC8201, July 2017, <<https://www.rfc-editor.org/info/rfc8201>>.

- [15] Allman, M., "Requirements for Time-Based Loss Detection", BCP 233, RFC 8961, DOI 10.17487/RFC8961, November 2020, <<https://www.rfc-editor.org/info/rfc8961>>.

## 9.2. Informative References

- [16] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [17] Postel, J., "The TCP Maximum Segment Size and Related Topics", RFC 879, DOI 10.17487/RFC0879, November 1983, <<https://www.rfc-editor.org/info/rfc879>>.
- [18] Nagle, J., "Congestion Control in IP/TCP Internetworks", RFC 896, DOI 10.17487/RFC0896, January 1984, <<https://www.rfc-editor.org/info/rfc896>>.
- [19] Reynolds, J. and J. Postel, "Official Internet protocols", RFC 1011, DOI 10.17487/RFC1011, May 1987, <<https://www.rfc-editor.org/info/rfc1011>>.
- [20] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.
- [21] Almquist, P., "Type of Service in the Internet Protocol Suite", RFC 1349, DOI 10.17487/RFC1349, July 1992, <<https://www.rfc-editor.org/info/rfc1349>>.
- [22] Braden, R., "T/TCP -- TCP Extensions for Transactions Functional Specification", RFC 1644, DOI 10.17487/RFC1644, July 1994, <<https://www.rfc-editor.org/info/rfc1644>>.
- [23] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, DOI 10.17487/RFC2018, October 1996, <<https://www.rfc-editor.org/info/rfc2018>>.
- [24] Paxson, V., Allman, M., Dawson, S., Fenner, W., Griner, J., Heavens, I., Lahey, K., Semke, J., and B. Volz, "Known TCP Implementation Problems", RFC 2525, DOI 10.17487/RFC2525, March 1999, <<https://www.rfc-editor.org/info/rfc2525>>.

- [25] Borman, D., Deering, S., and R. Hinden, "IPv6 Jumbograms", RFC 2675, DOI 10.17487/RFC2675, August 1999, <<https://www.rfc-editor.org/info/rfc2675>>.
- [26] Xiao, X., Hannan, A., Paxson, V., and E. Crabbe, "TCP Processing of the IPv4 Precedence Field", RFC 2873, DOI 10.17487/RFC2873, June 2000, <<https://www.rfc-editor.org/info/rfc2873>>.
- [27] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, DOI 10.17487/RFC2883, July 2000, <<https://www.rfc-editor.org/info/rfc2883>>.
- [28] Lahey, K., "TCP Problems with Path MTU Discovery", RFC 2923, DOI 10.17487/RFC2923, September 2000, <<https://www.rfc-editor.org/info/rfc2923>>.
- [29] Balakrishnan, H., Padmanabhan, V., Fairhurst, G., and M. Sooriyabandara, "TCP Performance Implications of Network Path Asymmetry", BCP 69, RFC 3449, DOI 10.17487/RFC3449, December 2002, <<https://www.rfc-editor.org/info/rfc3449>>.
- [30] Allman, M., "TCP Congestion Control with Appropriate Byte Counting (ABC)", RFC 3465, DOI 10.17487/RFC3465, February 2003, <<https://www.rfc-editor.org/info/rfc3465>>.
- [31] Fenner, B., "Experimental Values In IPv4, IPv6, ICMPv4, ICMPv6, UDP, and TCP Headers", RFC 4727, DOI 10.17487/RFC4727, November 2006, <<https://www.rfc-editor.org/info/rfc4727>>.
- [32] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, DOI 10.17487/RFC4821, March 2007, <<https://www.rfc-editor.org/info/rfc4821>>.
- [33] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <<https://www.rfc-editor.org/info/rfc4987>>.
- [34] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, DOI 10.17487/RFC4953, July 2007, <<https://www.rfc-editor.org/info/rfc4953>>.
- [35] Culley, P., Elzur, U., Recio, R., Bailey, S., and J. Carrier, "Marker PDU Aligned Framing for TCP Specification", RFC 5044, DOI 10.17487/RFC5044, October 2007, <<https://www.rfc-editor.org/info/rfc5044>>.

- [36] Gont, F., "TCP's Reaction to Soft Errors", RFC 5461, DOI 10.17487/RFC5461, February 2009, <<https://www.rfc-editor.org/info/rfc5461>>.
- [37] StJohns, M., Atkinson, R., and G. Thomas, "Common Architecture Label IPv6 Security Option (CALIPSO)", RFC 5570, DOI 10.17487/RFC5570, July 2009, <<https://www.rfc-editor.org/info/rfc5570>>.
- [38] Sandlund, K., Pelletier, G., and L-E. Jonsson, "The RObust Header Compression (ROHC) Framework", RFC 5795, DOI 10.17487/RFC5795, March 2010, <<https://www.rfc-editor.org/info/rfc5795>>.
- [39] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <<https://www.rfc-editor.org/info/rfc5925>>.
- [40] Gont, F. and A. Yourtchenko, "On the Implementation of the TCP Urgent Mechanism", RFC 6093, DOI 10.17487/RFC6093, January 2011, <<https://www.rfc-editor.org/info/rfc6093>>.
- [41] Gont, F., "Reducing the TIME-WAIT State Using TCP Timestamps", BCP 159, RFC 6191, DOI 10.17487/RFC6191, April 2011, <<https://www.rfc-editor.org/info/rfc6191>>.
- [42] Bashyam, M., Jethanandani, M., and A. Ramaiah, "TCP Sender Clarification for Persist Condition", RFC 6429, DOI 10.17487/RFC6429, December 2011, <<https://www.rfc-editor.org/info/rfc6429>>.
- [43] Gont, F. and S. Bellovin, "Defending against Sequence Number Attacks", RFC 6528, DOI 10.17487/RFC6528, February 2012, <<https://www.rfc-editor.org/info/rfc6528>>.
- [44] Borman, D., "TCP Options and Maximum Segment Size (MSS)", RFC 6691, DOI 10.17487/RFC6691, July 2012, <<https://www.rfc-editor.org/info/rfc6691>>.
- [45] Touch, J., "Updated Specification of the IPv4 ID Field", RFC 6864, DOI 10.17487/RFC6864, February 2013, <<https://www.rfc-editor.org/info/rfc6864>>.
- [46] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, DOI 10.17487/RFC6994, August 2013, <<https://www.rfc-editor.org/info/rfc6994>>.



- [47] McPherson, D., Oran, D., Thaler, D., and E. Osterweil, "Architectural Considerations of IP Anycast", RFC 7094, DOI 10.17487/RFC7094, January 2014, <<https://www.rfc-editor.org/info/rfc7094>>.
- [48] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, DOI 10.17487/RFC7323, September 2014, <<https://www.rfc-editor.org/info/rfc7323>>.
- [49] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <<https://www.rfc-editor.org/info/rfc7413>>.
- [50] Duke, M., Braden, R., Eddy, W., Blanton, E., and A. Zimmermann, "A Roadmap for Transmission Control Protocol (TCP) Specification Documents", RFC 7414, DOI 10.17487/RFC7414, February 2015, <<https://www.rfc-editor.org/info/rfc7414>>.
- [51] Black, D., Ed. and P. Jones, "Differentiated Services (Diffserv) and Real-Time Communication", RFC 7657, DOI 10.17487/RFC7657, November 2015, <<https://www.rfc-editor.org/info/rfc7657>>.
- [52] Fairhurst, G. and M. Welzl, "The Benefits of Using Explicit Congestion Notification (ECN)", RFC 8087, DOI 10.17487/RFC8087, March 2017, <<https://www.rfc-editor.org/info/rfc8087>>.
- [53] Fairhurst, G., Ed., Trammell, B., Ed., and M. Kuehlewind, Ed., "Services Provided by IETF Transport Protocols and Congestion Control Mechanisms", RFC 8095, DOI 10.17487/RFC8095, March 2017, <<https://www.rfc-editor.org/info/rfc8095>>.
- [54] Welzl, M., Tuexen, M., and N. Khademi, "On the Usage of Transport Features Provided by IETF Transport Protocols", RFC 8303, DOI 10.17487/RFC8303, February 2018, <<https://www.rfc-editor.org/info/rfc8303>>.
- [55] Chown, T., Loughney, J., and T. Winters, "IPv6 Node Requirements", BCP 220, RFC 8504, DOI 10.17487/RFC8504, January 2019, <<https://www.rfc-editor.org/info/rfc8504>>.
- [56] Trammell, B. and M. Kuehlewind, "The Wire Image of a Network Protocol", RFC 8546, DOI 10.17487/RFC8546, April 2019, <<https://www.rfc-editor.org/info/rfc8546>>.

- [57] Bittau, A., Giffin, D., Handley, M., Mazieres, D., Slack, Q., and E. Smith, "Cryptographic Protection of TCP Streams (tcpcrypt)", RFC 8548, DOI 10.17487/RFC8548, May 2019, <<https://www.rfc-editor.org/info/rfc8548>>.
- [58] Hardie, T., Ed., "Transport Protocol Path Signals", RFC 8558, DOI 10.17487/RFC8558, April 2019, <<https://www.rfc-editor.org/info/rfc8558>>.
- [59] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 8684, DOI 10.17487/RFC8684, March 2020, <<https://www.rfc-editor.org/info/rfc8684>>.
- [60] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <<https://www.rfc-editor.org/info/rfc9000>>.
- [61] Fairhurst, G. and C. Perkins, "Considerations around Transport Header Confidentiality, Network Operations, and the Evolution of Internet Transport Protocols", RFC 9065, DOI 10.17487/RFC9065, July 2021, <<https://www.rfc-editor.org/info/rfc9065>>.
- [62] IANA, "Transmission Control Protocol (TCP) Parameters, <https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml>", 2019.
- [63] IANA, "Transmission Control Protocol (TCP) Header Flags, <https://www.iana.org/assignments/tcp-header-flags/tcp-header-flags.xhtml>", 2019.
- [64] Gont, F., "Processing of IP Security/Compartment and Precedence Information by TCP", Work in Progress, Internet-Draft, draft-gont-tcpm-tcp-seccomp-prec-00, 29 March 2012, <<http://www.ietf.org/internet-drafts/draft-gont-tcpm-tcp-seccomp-prec-00.txt>>.
- [65] Gont, F. and D. Borman, "On the Validation of TCP Sequence Numbers", Work in Progress, Internet-Draft, draft-gont-tcpm-tcp-seq-validation-04, 11 March 2019, <<http://www.ietf.org/internet-drafts/draft-gont-tcpm-tcp-seq-validation-04.txt>>.

- [66] Touch, J. and W. Eddy, "TCP Extended Data Offset Option", Work in Progress, Internet-Draft, draft-ietf-tcpm-tcp-edo-10, 19 July 2018, <<http://www.ietf.org/internet-drafts/draft-ietf-tcpm-tcp-edo-10.txt>>.
- [67] McQuistin, S., Band, V., Jacob, D., and C. Perkins, "Describing Protocol Data Units with Augmented Packet Header Diagrams", Work in Progress, Internet-Draft, draft-mcquistin-augmented-ascii-diagrams-08, 5 May 2021, <<https://www.ietf.org/archive/id/draft-mcquistin-augmented-ascii-diagrams-08.txt>>.
- [68] Thomson, M. and T. Pauly, "Long-term Viability of Protocol Extension Mechanisms", Work in Progress, Internet-Draft, draft-iab-use-it-or-lose-it-02, 23 August 2021, <<https://www.ietf.org/archive/id/draft-iab-use-it-or-lose-it-02.txt>>.
- [69] Minshall, G., "A Proposed Modification to Nagle's Algorithm", Work in Progress, Internet-Draft, draft-minshall-nagle-01, June 1999, <<https://datatracker.ietf.org/doc/html/draft-minshall-nagle-01>>.
- [70] Dalal, Y. and C. Sunshine, "Connection Management in Transport Protocols", Computer Networks Vol. 2, No. 6, pp. 454-473, December 1978.
- [71] Faber, T., Touch, J., and W. Yui, "The TIME-WAIT state in TCP and Its Effect on Busy Servers", Proceedings of IEEE INFOCOM pp. 1573-1583, March 1999.
- [72] Postel, J., "Comments on Action Items from the January Meeting", IEN 177, March 1981, <<https://www.rfc-editor.org/ien/ien177.txt>>.
- [73] "Segmentation Offloads", Linux Networking Documentation , <<https://www.kernel.org/doc/html/latest/networking/segmentation-offloads.html>>.

#### Appendix A. Other Implementation Notes

This section includes additional notes and references on TCP implementation decisions that are currently not a part of the RFC series or included within the TCP standard. These items can be considered by implementers, but there was not yet a consensus to include them in the standard.

### A.1. IP Security Compartment and Precedence

The IPv4 specification [1] includes a precedence value in the (now obsolete) Type of Service field (TOS) field. It was modified in [21], and then obsolete by the definition of Differentiated Services (DiffServ) [4]. Setting and conveying TOS between the network layer, TCP implementation, and applications is obsolete, and replaced by DiffServ in the current TCP specification.

RFC 793 required checking the IP security compartment and precedence on incoming TCP segments for consistency within a connection, and with application requests. Each of these aspects of IP have become outdated, without specific updates to RFC 793. The issues with precedence were fixed by [26], which is Standards Track, and so this present TCP specification includes those changes. However, the state of IP security options that may be used by MLS systems is not as apparent in the IETF currently.

Resetting connections when incoming packets do not meet expected security compartment or precedence expectations has been recognized as a possible attack vector [64], and there has been discussion about amending the TCP specification to prevent connections from being aborted due to non-matching IP security compartment and DiffServ codepoint values.

#### A.1.1. Precedence

In DiffServ the former precedence values are treated as Class Selector codepoints, and methods for compatible treatment are described in the DiffServ architecture. The RFC 793/1122 TCP specification includes logic intending to have connections use the highest precedence requested by either endpoint application, and to keep the precedence consistent throughout a connection. This logic from the obsolete TOS is not applicable for DiffServ, and should not be included in TCP implementations, though changes to DiffServ values within a connection are discouraged. For discussion of this, see RFC 7657 (sec 5.1, 5.3, and 6) [51].

The obsolete TOS processing rules in TCP assumed bidirectional (or symmetric) precedence values used on a connection, but the DiffServ architecture is asymmetric. Problems with the old TCP logic in this regard were described in [26] and the solution described is to ignore IP precedence in TCP. Since RFC 2873 is a Standards Track document (although not marked as updating RFC 793), current implementations are expected to be robust to these conditions. Note that the DiffServ field value used in each direction is a part of the interface between TCP and the network layer, and values in use can be indicated both ways between TCP and the application.

### A.1.2. MLS Systems

The IP security option (IPSO) and compartment defined in [1] was refined in RFC 1038 that was later obsoleted by RFC 1108. The Commercial IP Security Option (CIPSO) is defined in FIPS-188 (withdrawn by NIST in 2015), and is supported by some vendors and operating systems. RFC 1108 is now Historic, though RFC 791 itself has not been updated to remove the IP security option. For IPv6, a similar option (CALIPSO) has been defined [37]. RFC 793 includes logic that includes the IP security/compartment information in treatment of TCP segments. References to the IP "security/compartment" in this document may be relevant for Multi-Level Secure (MLS) system implementers, but can be ignored for non-MLS implementations, consistent with running code on the Internet. See Appendix A.1 for further discussion. Note that RFC 5570 describes some MLS networking scenarios where IPSO, CIPSO, or CALIPSO may be used. In these special cases, TCP implementers should see section 7.3.1 of RFC 5570, and follow the guidance in that document.

### A.2. Sequence Number Validation

There are cases where the TCP sequence number validation rules can prevent ACK fields from being processed. This can result in connection issues, as described in [65], which includes descriptions of potential problems in conditions of simultaneous open, self-connects, simultaneous close, and simultaneous window probes. The document also describes potential changes to the TCP specification to mitigate the issue by expanding the acceptable sequence numbers.

In Internet usage of TCP, these conditions are rarely occurring. Common operating systems include different alternative mitigations, and the standard has not been updated yet to codify one of them, but implementers should consider the problems described in [65].

### A.3. Nagle Modification

In common operating systems, both the Nagle algorithm and delayed acknowledgements are implemented and enabled by default. TCP is used by many applications that have a request-response style of communication, where the combination of the Nagle algorithm and delayed acknowledgements can result in poor application performance. A modification to the Nagle algorithm is described in [69] that improves the situation for these applications.

This modification is implemented in some common operating systems, and does not impact TCP interoperability. Additionally, many applications simply disable Nagle, since this is generally supported by a socket option. The TCP standard has not been updated to include this Nagle modification, but implementers may find it beneficial to consider.

#### A.4. Low Watermark Settings

Some operating system kernel TCP implementations include socket options that allow specifying the number of bytes in the buffer until the socket layer will pass sent data to TCP (SO\_SNDLOWAT) or to the application on receiving (SO\_RCVLOWAT).

In addition, another socket option (TCP\_NOTSENT\_LOWAT) can be used to control the amount of unsent bytes in the write queue. This can help a sending TCP application to avoid creating large amounts of buffered data (and corresponding latency). As an example, this may be useful for applications that are multiplexing data from multiple upper level streams onto a connection, especially when streams may be a mix of interactive / real-time and bulk data transfer.

#### Appendix B. TCP Requirement Summary

This section is adapted from RFC 1122.

Note that there is no requirement related to PLPMTUD in this list, but that PLPMTUD is recommended.

FEATURE	ReqID	MUST	SHOULD	MAY	SHOULD NOT	OPTIONAL	Forbidden
Push flag							
Aggregate or queue un-pushed data	MAY-16			x			
Sender collapse successive PSH flags	SHLD-27		x				
SEND call can specify PUSH	MAY-15			x			
If cannot: sender buffer indefinitely	MUST-60					x	
If cannot: PSH last segment	MUST-61	x					
Notify receiving ALP of PSH	MAY-17			x			1

Send max size segment when possible	SHLD-28	x				
Window						
Treat as unsigned number	MUST-1	x				
Handle as 32-bit number	REC-1		x			
Shrink window from right	SHLD-14			x		
- Send new data when window shrinks	SHLD-15			x		
- Retransmit old unacked data within window	SHLD-16	x				
- Time out conn for data past right edge	SHLD-17			x		
Robust against shrinking window	MUST-34	x				
Receiver's window closed indefinitely	MAY-8		x			
Use standard probing logic	MUST-35	x				
Sender probe zero window	MUST-36	x				
First probe after RTO	SHLD-29		x			
Exponential backoff	SHLD-30		x			
Allow window stay zero indefinitely	MUST-37	x				
Retransmit old data beyond SND.UNA+SND.WND	MAY-7		x			
Process RST and URG even with zero window	MUST-66	x				
Urgent Data						
Include support for urgent pointer	MUST-30	x				
Pointer indicates first non-urgent octet	MUST-62	x				
Arbitrary length urgent data sequence	MUST-31	x				
Inform ALP asynchronously of urgent data	MUST-32	x				1
ALP can learn if/how much urgent data Q'd	MUST-33	x				1
ALP employ the urgent mechanism	SHLD-13			x		
TCP Options						
Support the mandatory option set	MUST-4	x				
Receive TCP option in any segment	MUST-5	x				
Ignore unsupported options	MUST-6	x				
Include length for all options except EOL+NOP	MUST-68	x				
Cope with illegal option length	MUST-7	x				
Process options regardless of word alignment	MUST-64	x				
Implement sending & receiving MSS option	MUST-14	x				
IPv4 Send MSS option unless 536	SHLD-5		x			
IPv6 Send MSS option unless 1220	SHLD-5		x			
Send MSS option always	MAY-3			x		
IPv4 Send-MSS default is 536	MUST-15	x				
IPv6 Send-MSS default is 1220	MUST-15	x				
Calculate effective send seg size	MUST-16	x				
MSS accounts for varying MTU	SHLD-6		x			
MSS not sent on non-SYN segments	MUST-65				x	
MSS value based on MMS_R	MUST-67	x				
Pad with zero	MUST-69	x				
TCP Checksums						
Sender compute checksum	MUST-2	x				

Receiver check checksum	MUST-3	x					
ISN Selection							
Include a clock-driven ISN generator component	MUST-8	x					
Secure ISN generator with a PRF component	SHLD-1		x				
PRF computable from outside the host	MUST-9					x	
Opening Connections							
Support simultaneous open attempts	MUST-10	x					
SYN-RECEIVED remembers last state	MUST-11	x					
Passive Open call interfere with others	MUST-41					x	
Function: simultan. LISTENS for same port	MUST-42	x					
Ask IP for src address for SYN if necc.	MUST-44	x					
Otherwise, use local addr of conn.	MUST-45	x					
OPEN to broadcast/multicast IP Address	MUST-46					x	
Silently discard seg to bcast/mcast addr	MUST-57	x					
Closing Connections							
RST can contain data	SHLD-2		x				
Inform application of aborted conn	MUST-12	x					
Half-duplex close connections	MAY-1			x			
Send RST to indicate data lost	SHLD-3		x				
In TIME-WAIT state for 2MSL seconds	MUST-13	x					
Accept SYN from TIME-WAIT state	MAY-2			x			
Use Timestamps to reduce TIME-WAIT	SHLD-4		x				
Retransmissions							
Implement exponential backoff, slow start, and congestion avoidance	MUST-19	x					
Retransmit with same IP ident	MAY-4			x			
Karn's algorithm	MUST-18	x					
Generating ACKs:							
Aggregate whenever possible	MUST-58	x					
Queue out-of-order segments	SHLD-31		x				
Process all Q'd before send ACK	MUST-59	x					
Send ACK for out-of-order segment	MAY-13			x			
Delayed ACKs	SHLD-18		x				
Delay < 0.5 seconds	MUST-40	x					
Every 2nd full-sized segment or 2*RMSS ACK'd	SHLD-19		x				
Receiver SWS-Avoidance Algorithm	MUST-39	x					
Sending data							
Configurable TTL	MUST-49	x					
Sender SWS-Avoidance Algorithm	MUST-38	x					
Nagle algorithm	SHLD-7		x				
Application can disable Nagle algorithm	MUST-17	x					



## Connection Failures:

Negative advice to IP on R1 retxs  
 Close connection on R2 retxs  
 ALP can set R2  
 Inform ALP of  $R1 \leq \text{retxs} < R2$   
 Recommended value for R1  
 Recommended value for R2  
 Same mechanism for SYN  
 R2 at least 3 minutes for SYN

MUST-20	x					
MUST-20	x					
MUST-21	x					1
SHLD-9		x				1
SHLD-10		x				
SHLD-11		x				
MUST-22	x					
MUST-23	x					

## Send Keep-alive Packets:

- Application can request
- Default is "off"
- Only send if idle for interval
- Interval configurable
- Default at least 2 hrs.
- Tolerant of lost ACKs
- Send with no data
- Configurable to send garbage octet

MAY-5			x			
MUST-24	x					
MUST-25	x					
MUST-26	x					
MUST-27	x					
MUST-28	x					
MUST-29	x					
SHLD-12		x				
MAY-6			x			

## IP Options

Ignore options TCP doesn't understand  
 Time Stamp support  
 Record Route support  
 Source Route:  
 ALP can specify  
 Overrides src rt in datagram  
 Build return route from src rt  
 Later src route overrides

MUST-50	x					
MAY-10			x			
MAY-11			x			
MUST-51	x					1
MUST-52	x					
MUST-53	x					
SHLD-24		x				

## Receiving ICMP Messages from IP

Dest. Unreach (0,1,5) => inform ALP  
 Abort on Dest. Unreach (0,1,5) =>nn  
 Dest. Unreach (2-4) => abort conn  
 Source Quench => silent discard  
 Abort on Time Exceeded =>  
 Abort on Param Problem =>

MUST-54	x					
SHLD-25		x				
MUST-56					x	
SHLD-26		x				
MUST-55	x					
MUST-56					x	
MUST-56					x	

## Address Validation

Reject OPEN call to invalid IP address  
 Reject SYN from invalid IP address  
 Silently discard SYN to bcast/mcast addr

MUST-46	x					
MUST-63	x					
MUST-57	x					

## TCP/ALP Interface Services

Error Report mechanism  
 ALP can disable Error Report Routine  
 ALP can specify DiffServ field for sending  
 Passed unchanged to IP

MUST-47	x					
SHLD-20		x				
MUST-48	x					
SHLD-22		x				

ALP can change DiffServ field during connection	SHLD-21		x				
ALP generally changing DiffServ during conn.	SHLD-23				x		
Pass received DiffServ field up to ALP	MAY-9			x			
FLUSH call	MAY-14			x			
Optional local IP addr parm. in OPEN	MUST-43	x					
RFC 5961 Support:							
Implement data injection protection	MAY-12			x			
Explicit Congestion Notification:							
Support ECN	SHLD-8		x				
Alternative Congestion Control:							
Implement alternative conformant algorithm(s)	MAY-18			x			
-----	-----	-	-	-	-	-	-

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

#### Author's Address

Wesley M. Eddy (editor)  
MTI Systems  
United States of America  
Email: wes@mti-systems.com

TCPM  
Internet-Draft  
Intended status: Standards Track  
Expires: January 11, 2021

M. Scharf  
Hochschule Esslingen  
V. Murgai

M. Jethanandani  
Kloud Services  
July 10, 2020

YANG Model for Transmission Control Protocol (TCP) Configuration  
draft-scharf-tcpm-yang-tcp-06

Abstract

This document specifies a YANG model for TCP on devices that are configured by network management protocols. The YANG model defines a container for all TCP connections and groupings of some of the parameters that can be imported and used in TCP implementations or by other models that need to configure TCP parameters. The model includes definitions from YANG Groupings for TCP Client and TCP Servers (I-D.ietf-netconf-tcp-client-server). The model is NMDA (RFC 8342) compliant.

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 January 11, 2021.

Copyright Notice

Copyright (c) 2020 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. Requirements Language . . . . .	3
2.1. Note to RFC Editor . . . . .	3
3. Model Overview . . . . .	4
3.1. Modeling Scope . . . . .	4
3.2. Model Design . . . . .	5
3.3. Tree Diagram . . . . .	6
4. TCP YANG Model . . . . .	6
5. IANA Considerations . . . . .	13
5.1. The IETF XML Registry . . . . .	13
5.2. The YANG Module Names Registry . . . . .	13
6. Security Considerations . . . . .	13
7. References . . . . .	13
7.1. Normative References . . . . .	13
7.2. Informative References . . . . .	15
Appendix A. Acknowledgements . . . . .	16
Appendix B. Changes compared to previous versions . . . . .	16
Appendix C. Examples . . . . .	17
C.1. Keepalive Configuration . . . . .	17
Authors' Addresses . . . . .	18

## 1. Introduction

The Transmission Control Protocol (TCP) [RFC0793] is used by many applications in the Internet, including control and management protocols. Therefore, TCP is implemented on network elements that can be configured via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. This document specifies a YANG [RFC7950] 1.1 model for configuring TCP on network elements that support YANG data models, and is Network Management Datastore Architecture (NMDA) [RFC8342] compliant. This module defines a container for TCP connection, and includes definitions from YANG Groupings for TCP Clients and TCP Servers [I-D.ietf-netconf-tcp-client-server]. The model has a narrow scope and focuses on fundamental TCP functions and basic statistics. The model can be augmented or updated to address more advanced or implementation-specific TCP features in the future.

Many protocol stacks on Internet hosts use other methods to configure TCP, such as operating system configuration or policies. Many TCP/IP stacks cannot be configured by network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. Moreover, many existing TCP/IP stacks do not use YANG data models. Such TCP implementations often have other means to configure the parameters listed in this document, which are outside the scope of this document.

This specification is orthogonal to the Management Information Base (MIB) for the Transmission Control Protocol (TCP) [RFC4022]. The basic statistics defined in this document follow the model of the TCP MIB. An TCP Extended Statistics MIB [RFC4898] is also available, but this document does not cover such extended statistics. It is possible also to translate a MIB into a YANG model, for instance using Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules [RFC6643]. However, this approach is not used in this document, as such a translated model would not be up-to-date.

There are other existing TCP-related YANG models, which are orthogonal to this specification. Examples are:

- o TCP header attributes are modeled in other models, such as YANG Data Model for Network Access Control Lists (ACLs) [RFC8519] and Distributed Denial-of-Service Open Thread Signaling (DOTS) Data Channel Specification [I-D.ietf-dots-data-channel].
- o TCP-related configuration of a NAT (e.g., NAT44, NAT64, Destination NAT, ...) is defined in A YANG Module for Network Address Translation (NAT) and Network Prefix Translation (NPT) [RFC8512] and A YANG Data Model for Dual-Stack Lite (DS-Lite) [RFC8513].

## 2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

### 2.1. Note to RFC Editor

This document uses several placeholder values throughout the document. Please replace them as follows and remove this note before publication.

RFC XXXX, where XXXX is the number assigned to this document at the time of publication.

2020-07-10 with the actual date of the publication of this document.

### 3. Model Overview

#### 3.1. Modeling Scope

TCP is implemented on many different system architectures. As a result, there are many different and often implementation-specific ways to configure parameters of the TCP protocol engine. In addition, in many TCP/IP stacks configuration exists for different scopes:

- o Global configuration: Many TCP implementations have configuration parameters that affect all TCP connections. Typical examples include enabling or disabling optional protocol features.
- o Interface configuration: It can be useful to use different TCP parameters on different interfaces, e.g., different device ports or IP interfaces. In that case, TCP parameters can be part of the interface configuration. Typical examples are the Maximum Segment Size (MSS) or configuration related to hardware offloading.
- o Connection parameters: Many implementations have means to influence the behavior of each TCP connection, e.g., on the programming interface used by applications. A typical example are socket options in the socket API, such as disabling the Nagle algorithm by TCP\_NODELAY. If an application uses such an interface, it is possible that the configuration of the application or application protocol includes TCP-related parameters. An example is the BGP YANG Model for Service Provider Networks [I-D.ietf-idr-bgp-model].
- o Policies: Setting of TCP parameters can also be part of system policies, templates, or profiles. An example would be the preferences defined in An Abstract Application Layer Interface to Transport Services [I-D.ietf-taps-interface].

As a result, there is no ground truth for setting certain TCP parameters, and traditionally different TCP implementations have used different modeling approaches. For instance, one implementation may define a given configuration parameter globally, while another one uses per-interface settings, and both approaches work well for the corresponding use cases. Also, different systems may use different default values. In addition, TCP can be implemented in different

ways and design choices by the protocol engine often affect configuration options.

Nonetheless, a number of TCP stack parameters require configuration by YANG models. This document therefore defines a minimal YANG model with fundamental parameters directly following from TCP standards.

An important use case is the TCP configuration on network elements such as routers, which often use YANG data models. The model therefore specifies TCP parameters that are important on such TCP stacks. A typical example is the support of TCP-AO [RFC5925]. TCP-AO is increasingly supported on routers to secure routing protocols such as BGP. In that case, TCP-AO configuration is required on routers.

Given an installed base, the model also allows enabling of the legacy TCP MD5 [RFC2385] signature option. As the TCP MD5 signature option is obsoleted by TCP-AO, it is strongly RECOMMENDED to use TCP-AO.

Similar to the TCP MIB [RFC4022], this document also specifies basic statistics and a TCP connection table.

- o Statistics: Counters for the number of active/passive opens, sent and received segments, errors, and possibly other detailed debugging information
- o TCP connection table: Access to status information for all TCP connections

This allows implementations of TCP MIB [RFC4022] to migrate to the YANG model defined in this memo.

### 3.2. Model Design

The YANG model defined in this document includes definitions from the YANG Groupings for TCP Clients and TCP Servers [I-D.ietf-netconf-tcp-client-server]. Similar to that model, this specification defines YANG groupings. This allows reuse of these groupings in different YANG data models. It is intended that these groupings will be used either standalone or for TCP-based protocols as part of a stack of protocol-specific configuration models. An example could be the BGP YANG Model for Service Provider Networks [I-D.ietf-idr-bgp-model].

### 3.3. Tree Diagram

This section provides a abridged tree diagram for the YANG module defined in this document. Annotations used in the diagram are defined in YANG Tree Diagrams [RFC8340].

```
module: ietf-tcp
  +--rw tcp!
    +--rw connections
    |   ...
    +--rw server {server}?
    |   ...
    +--rw client {client}?
    |   ...
    +--ro statistics {statistics}?
    |   ...
```

### 4. TCP YANG Model

<CODE BEGINS> file "ietf-tcp@2020-07-10.yang"

```
module ietf-tcp {
  yang-version "1.1";
  namespace "urn:ietf:params:xml:ns:yang:ietf-tcp";
  prefix "tcp";

  import ietf-yang-types {
    prefix "yang";
    reference
      "RFC 6991: Common YANG Data Types.";
  }
  import ietf-tcp-client {
    prefix "tcpc";
  }
  import ietf-tcp-server {
    prefix "tcps";
  }
  import ietf-tcp-common {
    prefix "tcpcmn";
  }
  import ietf-inet-types {
    prefix "inet";
  }

  organization
    "IETF TCPM Working Group";

  contact
```



```
"WG Web:    <http://tools.ietf.org/wg/tcpm>
WG List:    <tcpm@ietf.org>

Authors: Michael Scharf (michael.scharf at hs-esslingen dot de)
        Vishal Murgai (vmurgai at gmail dot com)
        Mahesh Jethanandani (mjethanandani at gmail dot com)";

description
  "This module focuses on fundamental and standard TCP functions
  that widely implemented. The model can be augmented to address
  more advanced or implementation specific TCP features.";

revision "2020-07-10" {
  description
    "Initial Version";
  reference
    "RFC XXX, TCP Configuration.";
}

// Features
feature server {
  description
    "TCP Server configuration supported.";
}

feature client {
  description
    "TCP Client configuration supported.";
}

feature statistics {
  description
    "This implementation supports statistics reporting.";
}

// TCP-AO Groupings

grouping ao {
  leaf enable-ao {
    type boolean;
    default "false";
    description
      "Enable support of TCP-Authentication Option (TCP-AO).";
  }

  leaf send-id {
    type uint8 {
      range "0..255";
    }
  }
}
```

```
    }
    must "../enable-ao = 'true'";
    description
        "The SendID is inserted as the KeyID of the TCP-AO option
        of outgoing segments.";
    reference
        "RFC 5925: The TCP Authentication Option.";
}

leaf recv-id {
    type uint8 {
        range "0..255";
    }
    must "../enable-ao = 'true'";
    description
        "The RecvID is matched against the TCP-AO KeyID of incoming
        segments.";
    reference
        "RFC 5925: The TCP Authentication Option.";
}

leaf include-tcp-options {
    type boolean;
    must "../enable-ao = 'true'";
    description
        "Include TCP options in HMAC calculation.";
}

leaf accept-ao-mismatch {
    type boolean;
    must "../enable-ao = 'true'";
    description
        "Accept packets with HMAC mismatch.";
}
description
    "Authentication Option (AO) for TCP.";
reference
    "RFC 5925: The TCP Authentication Option.";
}

// MD5 grouping

grouping md5 {
    description
        "Grouping for use in authenticating TCP sessions using MD5.";
    reference
        "RFC 2385: Protection of BGP Sessions via the TCP MD5
        Signature.";
```

```
    leaf enable-md5 {
        type boolean;
        default "false";
        description
            "Enable support of MD5 to authenticate a TCP session.";
    }
}

// TCP configuration

container tcp {
    presence "The container for TCP configuration.";

    description
        "TCP container.";

    container connections {
        list connection {
            key "local-address remote-address local-port remote-port";

            leaf local-address {
                type inet:ip-address;
                description
                    "Local address that forms the connection identifier.";
            }

            leaf remote-address {
                type inet:ip-address;
                description
                    "Remote address that forms the connection identifier.";
            }

            leaf local-port {
                type inet:port-number;
                description
                    "Local TCP port that forms the connection identifier.";
            }

            leaf remote-port {
                type inet:port-number;
                description
                    "Remote TCP port that forms the connection identifier.";
            }
        }

        container common {
            uses tcpcmn:tcp-common-grouping;

            choice authentication {
```

```
    case ao {
        uses ao;
        description
            "Use TCP-AO to secure the connection.";
    }

    case md5 {
        uses md5;
        description
            "Use TCP-MD5 to secure the connection.";
    }
    description
        "Choice of how to secure the TCP connection.";
}
description
    "Common definitions of TCP configuration. This includes
    parameters such as how to secure the connection,
    that can be part of either the client or server.";
}
description
    "Connection related parameters.";
}
description
    "A container of all TCP connections.";
}

container server {
    if-feature server;
    uses tcps:tcp-server-grouping;
    description
        "Definitions of TCP server configuration.";
}

container client {
    if-feature client;
    uses tcpc:tcp-client-grouping;
    description
        "Definitions of TCP client configuration.";
}

container statistics {
    if-feature statistics;
    config false;

    leaf active-opens {
        type yang:counter32;
        description
            "The number of times that TCP connections have made a direct
```

```
        transition to the SYN-SENT state from the CLOSED state.";
    }

    leaf passive-opens {
        type yang:counter32;
        description
            "The number of times TCP connections have made a direct
            transition to the SYN-RCVD state from the LISTEN state.";
    }

    leaf attempt-fails {
        type yang:counter32;
        description
            "The number of times that TCP connections have made a direct
            transition to the CLOSED state from either the SYN-SENT
            state or the SYN-RCVD state, plus the number of times that
            TCP connections have made a direct transition to the
            LISTEN state from the SYN-RCVD state.";
    }

    leaf establish-resets {
        type yang:counter32;
        description
            "The number of times that TCP connections have made a direct
            transition to the CLOSED state from either the ESTABLISHED
            state or the CLOSE-WAIT state.";
    }

    leaf currently-established {
        type yang:gauge32;
        description
            "The number of TCP connections for which the current state
            is either ESTABLISHED or CLOSE-WAIT.";
    }

    leaf in-segments {
        type yang:counter64;
        description
            "The total number of segments received, including those
            received in error. This count includes segments received
            on currently established connections.";
    }

    leaf out-segments {
        type yang:counter64;
        description
            "The total number of segments sent, including those on
            current connections but excluding those containing only
```

```
        retransmitted octets.";
    }

    leaf retransmitted-segments {
        type yang:counter32;
        description
            "The total number of segments retransmitted; that is, the
             number of TCP segments transmitted containing one or more
             previously transmitted octets.";
    }

    leaf in-errors {
        type yang:counter32;
        description
            "The total number of segments received in error (e.g., bad
             TCP checksums).";
    }

    leaf out-resets {
        type yang:counter32;
        description
            "The number of TCP segments sent containing the RST flag.";
    }

    action reset {
        description
            "Reset statistics action command.";
        input {
            leaf reset-at {
                type yang:date-and-time;
                description
                    "Time when the reset action needs to be
                     executed.";
            }
        }
        output {
            leaf reset-finished-at {
                type yang:date-and-time;
                description
                    "Time when the reset action command completed.";
            }
        }
    }
    description
        "Statistics across all connections.";
}
}
```

<CODE ENDS>

## 5. IANA Considerations

### 5.1. The IETF XML Registry

This document registers two URIs in the "ns" subregistry of the IETF XML Registry [RFC3688]. Following the format in IETF XML Registry [RFC3688], the following registrations are requested:

URI: urn:ietf:params:xml:ns:yang:ietf-tcp  
Registrant Contact: The TCPM WG of the IETF.  
XML: N/A, the requested URI is an XML namespace.

### 5.2. The YANG Module Names Registry

This document registers a YANG modules in the YANG Module Names registry YANG - A Data Modeling Language [RFC6020]. Following the format in YANG - A Data Modeling Language [RFC6020], the following registrations are requested:

name: ietf-tcp  
namespace: urn:ietf:params:xml:ns:yang:ietf-tcp  
prefix: tcp  
reference: RFC XXXX

## 6. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) described in Using the NETCONF protocol over SSH [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

## 7. References

### 7.1. Normative References

- [I-D.ietf-netconf-tcp-client-server]  
Watsen, K. and M. Scharf, "YANG Groupings for TCP Clients and TCP Servers", draft-ietf-netconf-tcp-client-server-06 (work in progress), June 2020.
- [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>>.
- [RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, DOI 10.17487/RFC2385, August 1998, <<https://www.rfc-editor.org/info/rfc2385>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <<https://www.rfc-editor.org/info/rfc5925>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.



- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.
- [RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <<https://www.rfc-editor.org/info/rfc8342>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

## 7.2. Informative References

- [I-D.ietf-dots-data-channel]  
Boucadair, M. and T. Reddy.K, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Data Channel Specification", draft-ietf-dots-data-channel-31 (work in progress), July 2019.
- [I-D.ietf-idr-bgp-model]  
Jethanandani, M., Patel, K., Hares, S., and J. Haas, "BGP YANG Model for Service Provider Networks", draft-ietf-idr-bgp-model-09 (work in progress), June 2020.
- [I-D.ietf-taps-interface]  
Trammell, B., Welzl, M., Enghardt, T., Fairhurst, G., Kuehlewind, M., Perkins, C., Tiesel, P., Wood, C., and T. Pauly, "An Abstract Application Layer Interface to Transport Services", draft-ietf-taps-interface-06 (work in progress), March 2020.
- [RFC4022] Raghunarayan, R., Ed., "Management Information Base for the Transmission Control Protocol (TCP)", RFC 4022, DOI 10.17487/RFC4022, March 2005, <<https://www.rfc-editor.org/info/rfc4022>>.

- [RFC4898] Mathis, M., Heffner, J., and R. Raghunarayan, "TCP Extended Statistics MIB", RFC 4898, DOI 10.17487/RFC4898, May 2007, <<https://www.rfc-editor.org/info/rfc4898>>.
- [RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIV2) MIB Modules to YANG Modules", RFC 6643, DOI 10.17487/RFC6643, July 2012, <<https://www.rfc-editor.org/info/rfc6643>>.
- [RFC8512] Boucadair, M., Ed., Sivakumar, S., Jacquenet, C., Vinapamula, S., and Q. Wu, "A YANG Module for Network Address Translation (NAT) and Network Prefix Translation (NPT)", RFC 8512, DOI 10.17487/RFC8512, January 2019, <<https://www.rfc-editor.org/info/rfc8512>>.
- [RFC8513] Boucadair, M., Jacquenet, C., and S. Sivakumar, "A YANG Data Model for Dual-Stack Lite (DS-Lite)", RFC 8513, DOI 10.17487/RFC8513, January 2019, <<https://www.rfc-editor.org/info/rfc8513>>.
- [RFC8519] Jethanandani, M., Agarwal, S., Huang, L., and D. Blair, "YANG Data Model for Network Access Control Lists (ACLs)", RFC 8519, DOI 10.17487/RFC8519, March 2019, <<https://www.rfc-editor.org/info/rfc8519>>.

#### Appendix A. Acknowledgements

Michael Scharf was supported by the StandICT.eu project, which is funded by the European Commission under the Horizon 2020 Programme.

The following persons have contributed to this document by reviews:  
Mohamed Boucadair

#### Appendix B. Changes compared to previous versions

Changes compared to draft-scharf-tcpm-yang-tcp-04

- o Removed congestion control
- o Removed global stack parameters

Changes compared to draft-scharf-tcpm-yang-tcp-03

- o Updated TCP-AO grouping
- o Added congestion control

Changes compared to draft-scharf-tcpm-yang-tcp-02

- o Initial proposal of a YANG model including base configuration parameters, TCP-AO configuration, and a connection list
- o Editorial bugfixes and outdated references reported by Mohamed Boucadair
- o Additional co-author Mahesh Jethanandani

Changes compared to draft-scharf-tcpm-yang-tcp-01

- o Alignment with [I-D.ietf-netconf-tcp-client-server]
- o Removing backward-compatibility to the TCP MIB
- o Additional co-author Vishal Murgai

Changes compared to draft-scharf-tcpm-yang-tcp-00

- o Editorial improvements

## Appendix C. Examples

### C.1. Keepalive Configuration

This particular example demonstrates how both a particular connection can be configured for keepalives.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  This example shows how TCP keepalive can be configured for
  a given connection. An idle connection is dropped after
  idle-time + (max-probes * probe-interval).
-->
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <tcp
    xmlns="urn:ietf:params:xml:ns:yang:ietf-tcp">
    <connections>
      <connection>
        <local-address>192.168.1.1</local-address>
        <remote-address>192.168.1.2</remote-address>
        <local-port>1025</local-port>
        <remote-port>80</remote-port>
        <common>
          <keepalives>
            <idle-time>5</idle-time>
            <max-probes>5</max-probes>
            <probe-interval>10</probe-interval>
          </keepalives>
        </common>
      </connection>
    </connections>
  <!--
    It is not clear why a server and client configuration is
    needed here even as they under a feature statement and therefore
    are required only if the feature is declared. Adding it so
    that yanglint allows this validation to run.
  -->
  <server>
    <local-address>192.168.1.1</local-address>
  </server>
  <client>
    <remote-address>192.168.1.2</remote-address>
  </client>
</tcp>
</config>
```

#### Authors' Addresses

Michael Scharf  
Hochschule Esslingen - University of Applied Sciences  
Flandernstr. 101  
Esslingen 73732  
Germany

Email: michael.scharf@hs-esslingen.de

Vishal Murgai

Email: vmurgai@gmail.com

Mahesh Jethanandani  
Kloud Services

Email: mjethanandani@gmail.com