TCPM WG                                            J. Touch
Internet Draft                                       Independent
Intended status: Informational                        M. Welzl
Obsoletes: 2140                                        S. Islam
Expires: May 2020                                    University of Oslo
November 19, 2019

TCP Control Block Interdependence
draft-ietf-tcpm-2140bis-01.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/1id-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 May 19, 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.

Abstract

This memo provides guidance to TCP implementers that are intended to help improve convergence to steady-state operation without affecting interoperability. It updates and replaces RFC 2140’s description of interdependent TCP control blocks and the ways that part of TCP state can be shared among similar concurrent or consecutive connections. TCP state includes a combination of parameters, such as connection state, current round-trip time estimates, congestion control information, and process information. Most of this state is maintained on a per-connection basis in the TCP Control Block (TCB), but implementations can (and do) share certain TCB information across connections to the same host. Such sharing is intended to improve overall transient transport performance, while maintaining backward-compatibility with existing implementations. The sharing described herein is limited to only the TCB initialization and so has no effect on the long-term behavior of TCP after a connection has been established.

Table of Contents

1. Introduction.......................................................3
2. Conventions used in this document..............................4
3. Terminology......................................................4
4. The TCP Control Block (TCB)....................................4
5. TCB Interdependence............................................5
6. An Example of Temporal Sharing................................6
7. An Example of Ensemble Sharing................................9
8. Compatibility Issues...........................................11
9. Implications.....................................................13
10. Implementation Observations................................14
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 can (and now do) 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 often used in the World-Wide Web [Be94], [Br02]. This sharing of state is intended to help TCP connections converge to steady-state behavior 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, irrespective of TCP port numbers. 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 [RFC6824].
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.

However, this document is intended to describe behavior that is already permitted by TCP implementers. As a result, it provides informative guidance but does not use such normative language, except when quoting other documents.

3. Terminology

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

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

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]:

[Page 4]
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)
    round-trip time and variance
    cong. window size (snd_cwnd)*
    cong. window size threshold (ssthresh)*
    max window size seen*
    sendMSS#
    MMS_S#
    MMS_R#
    PMTU#
    round-trip time and variance#

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 further 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 (#, e.g., MSS, MMS, PMTU, RTT), and the other is host-pair dependent in its aggregate (*, e.g., congestion window information, current window sizes, etc.).

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. An Example of 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. New TCBs can be initialized using context from past connections as follows:

TEMPORAL SHARING - TCB Initialization

<table>
<thead>
<tr>
<th>Cached TCB</th>
<th>New TCB</th>
</tr>
</thead>
<tbody>
<tr>
<td>old_MMS_S</td>
<td>old_MMS_S or not cached</td>
</tr>
<tr>
<td>old_MMS_R</td>
<td>old_MMS_R or not cached</td>
</tr>
<tr>
<td>old_sendMSS</td>
<td>old_sendMSS</td>
</tr>
<tr>
<td>old_PMTU</td>
<td>old_PMTU</td>
</tr>
<tr>
<td>old_RTT</td>
<td>old_RTT</td>
</tr>
<tr>
<td>old_RTTvar</td>
<td>old_RTTvar</td>
</tr>
<tr>
<td>old_option</td>
<td>(option specific)</td>
</tr>
<tr>
<td>old_ssthresh</td>
<td>old_ssthresh</td>
</tr>
<tr>
<td>old_snd_cwnd</td>
<td>old_snd_cwnd</td>
</tr>
</tbody>
</table>

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 [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 MMS_R and MMS_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.

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

<table>
<thead>
<tr>
<th>TEMPORAL SHARING - Option info</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cached</td>
<td>New</td>
</tr>
<tr>
<td>old_TFO_Cookie</td>
<td>old_TFO_Cookie</td>
<td></td>
</tr>
<tr>
<td>old_TFO_Failure</td>
<td>old_TFO_Failure</td>
<td></td>
</tr>
</tbody>
</table>
### TEMPORAL SHARING - Cache Updates

<table>
<thead>
<tr>
<th>Cached TCB</th>
<th>Current TCB</th>
<th>when?</th>
<th>New Cached TCB</th>
</tr>
</thead>
<tbody>
<tr>
<td>old_MMS_S</td>
<td>curr_MMS_S</td>
<td>OPEN</td>
<td>curr_MMS_S</td>
</tr>
<tr>
<td>old_MMS_R</td>
<td>curr_MMS_R</td>
<td>OPEN</td>
<td>curr_MMS_R</td>
</tr>
<tr>
<td>old_sendMSS</td>
<td>curr_sendMSS</td>
<td>MSSopt</td>
<td>curr_sendMSS</td>
</tr>
<tr>
<td>old_PMTU</td>
<td>curr_PMTU</td>
<td>PMTUD</td>
<td>curr_PMTU</td>
</tr>
<tr>
<td>old_RTT</td>
<td>curr_RTT</td>
<td>CLOSE</td>
<td>merge(curr,old)</td>
</tr>
<tr>
<td>old_RTTvar</td>
<td>curr_RTTvar</td>
<td>CLOSE</td>
<td>merge(curr,old)</td>
</tr>
<tr>
<td>old_option</td>
<td>curr_option</td>
<td>ESTAB</td>
<td>(depends on option)</td>
</tr>
<tr>
<td>old_ssthresh</td>
<td>curr_ssthresh</td>
<td>CLOSE</td>
<td>merge(curr,old)</td>
</tr>
<tr>
<td>old_snd_cwnd</td>
<td>curr_snd_cwnd</td>
<td>CLOSE</td>
<td>merge(curr,old)</td>
</tr>
</tbody>
</table>

Caching PMTU and sendMSS is trivial; reported values are cached, 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. There is no particular benefit to caching MMS_S and MMS_R as these are reported by the local IP stack.

TCP options are copied or merged depending on the details of each option, where "merge" is some function that combines the values of "curr" and "old". E.g., TFO state is updated when a connection is established and read before establishing a new connection.

RTT values are updated by formulae that merge the old and new values. Dynamic RTT estimation requires a sequence of RTT measurements. As a result, the cached RTT (and its variance) 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 for new
corrections.

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.

<table>
<thead>
<tr>
<th>Cached</th>
<th>Current</th>
<th>when?</th>
<th>New Cached</th>
</tr>
</thead>
<tbody>
<tr>
<td>old_TFO_Cookie</td>
<td>old_TFO_Cookie</td>
<td>ESTAB</td>
<td>old_TFO_Cookie</td>
</tr>
<tr>
<td>old_TFO_Failure</td>
<td>old_TFO_Failure</td>
<td>ESTAB</td>
<td>old_TFO_Failure</td>
</tr>
</tbody>
</table>

### 7. An Example of 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 ($N$).

<table>
<thead>
<tr>
<th>Cached TCB</th>
<th>New TCB</th>
</tr>
</thead>
<tbody>
<tr>
<td>old_MMS_S</td>
<td>old_MMS_S</td>
</tr>
<tr>
<td>old_MMS_R</td>
<td>old_MMS_R</td>
</tr>
<tr>
<td>old_sendMSS</td>
<td>old_sendMSS</td>
</tr>
<tr>
<td>old_PMTU</td>
<td>old_PMTU</td>
</tr>
<tr>
<td>old_RTT</td>
<td>old_RTT</td>
</tr>
<tr>
<td>old_RTTvar</td>
<td>old_RTTvar</td>
</tr>
<tr>
<td>old ssthresh sum</td>
<td>$f$(old ssthresh sum, $N$)</td>
</tr>
<tr>
<td>old snd_cwnd sum</td>
<td>$f$(old snd cwnd sum, $N$)</td>
</tr>
<tr>
<td>old_option</td>
<td>(option-specific)</td>
</tr>
</tbody>
</table>
Sections 8 and 9 discuss compatibility issues and implications of sharing the specific information listed above.

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

<table>
<thead>
<tr>
<th>ENSEMBLE SHARING</th>
<th>Option info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cached</td>
<td>New</td>
</tr>
<tr>
<td>old_TFO_Cookie</td>
<td>old_TFO_Cookie</td>
</tr>
<tr>
<td>old_TFO_Failure</td>
<td>old_TFO_Failure</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ENSEMBLE SHARING - Cache Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cached TCB</td>
</tr>
<tr>
<td>------------</td>
</tr>
<tr>
<td>old_MMS_S</td>
</tr>
<tr>
<td>old_MMS_R</td>
</tr>
<tr>
<td>old_sendMSS</td>
</tr>
<tr>
<td>old_PMTU</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>old_RTT</td>
</tr>
<tr>
<td>old_RTTvar</td>
</tr>
<tr>
<td>old_ssthresh</td>
</tr>
<tr>
<td>old_snd_cwnd</td>
</tr>
<tr>
<td>old_option</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Cached</th>
<th>Current</th>
<th>when?</th>
<th>New Cached</th>
</tr>
</thead>
<tbody>
<tr>
<td>old_TFO_Cookie</td>
<td>old_TFO_Cookie</td>
<td>ESTAB</td>
<td>old_TFO_Cookie</td>
</tr>
<tr>
<td>old_TFO_Failure</td>
<td>old_TFO_Failure</td>
<td>ESTAB</td>
<td>old_TFO_Failure</td>
</tr>
</tbody>
</table>

Any assumption of this sharing can be incorrect because identical endpoint address pairs may not share network paths. In current implementations, new congestion windows are set at an initial value of 4-10 segments [RFC3390][RFC6928], so that the sum of the current windows is increased for any new connection. This can have detrimental consequences where several connections share a highly congested link.

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 [Ba12] adjusts the initial cwnd depending on the cwnd values of ongoing connections. There have also been suggestions to use the kind of sharing mechanisms described in this document over long timescales to adapt TCP’s initial window automatically, as described further in Appendix A [To12].

8. Compatibility Issues

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 a single existing connection has converged to a congestion window of 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 [Hu12] 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 [Hu12, Du16], although this may benefit aggregate network performance.

Due to mechanisms like ECMP and LAG [RFC7424], TCP connections sharing the same host-pair may not always share the same path. This does not matter for host-specific information such as RWIN and TCP option state, such as TFOinfo. 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. 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].

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.

TCP is sometimes used in situations where packets of the same host-pair do not always take the same path. Multipath routing that relies on examining transport headers, such as ECMP and LAG, 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. TCB interdependence among the entire set sharing the same endpoint IP addresses should work without problems under these circumstances. Moreover, 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. 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).

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.

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 in a network-layer fashion, 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.

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 [RFC6824].

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.

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 or end. 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, TCB interdependence is more likely to exhibit the same behavior as unmodified, independent TCP connections.

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 for more information).
The table below describes the current implementation status for some TCB information in Linux kernel version 4.6, FreeBSD 10 and Windows (as of October 2016). In the table, "shared" only refers to temporal sharing.

<table>
<thead>
<tr>
<th>TCB data</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>old MMS_S</td>
<td>Not shared</td>
</tr>
<tr>
<td>old MMS_R</td>
<td>Not shared</td>
</tr>
<tr>
<td>old_sendMSS</td>
<td>Cached and shared in Linux (MSS)</td>
</tr>
<tr>
<td>old PMTU</td>
<td>Cached and shared in FreeBSD and Windows (PMTU)</td>
</tr>
<tr>
<td>old_RTT</td>
<td>Cached and shared in FreeBSD and Linux</td>
</tr>
<tr>
<td>old_RTTvar</td>
<td>Cached and shared in FreeBSD</td>
</tr>
<tr>
<td>old_TFOinfo</td>
<td>Cached and shared in Linux and Windows</td>
</tr>
<tr>
<td>old_snd_cwnd</td>
<td>Not shared</td>
</tr>
<tr>
<td>old_ssthresh</td>
<td>Cached and shared in FreeBSD and Linux:</td>
</tr>
<tr>
<td></td>
<td>FreeBSD: arithmetic</td>
</tr>
<tr>
<td></td>
<td>mean of ssthresh and previous value if a previous value exists;</td>
</tr>
<tr>
<td></td>
<td>Linux: depending on state, cwnd/2, ssthresh in most cases</td>
</tr>
</tbody>
</table>

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 RSS in both the send and receive direction, MSS and send-MSS 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 Appendix A 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, largely imported from [To12].

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

12. Security Considerations

These presented implementation methods do not have additional ramifications for explicit attacks. They may be susceptible to denial-of-service attacks if not otherwise secured. For example, an application can open a connection and set its window size to zero, denying service to any other subsequent connection between those hosts.

TCB sharing may be susceptible to denial-of-service attacks, wherever 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.

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 would experience performance degradation until their window grew appropriately.

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

14.2. Informative References


Internet-Draft    TCP Control Block Interdependence       November 2019

Control," RFC 5681 (Standards Track), Sep. 2009.


[ RFC6824] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., "TCP
Extensions for Multipath Operation with Multiple


[ RFC7231] Fielding, R., J. Reschke, Eds., "HTTP/1.1 Semantics and
Content," RFC-7231, June 2014.

(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] Belshe, M., Peon, R., Thomson, M., "Hypertext Transfer


[To12] Touch, J., "Automating the Initial Window in TCP," draft-
touch-tcpm-automatic-iw-03 (expired), July 2012.

15. Acknowledgments

The authors would like to thank for Praveen Balasubramanian for
information regarding TCB sharing in Windows, and Yuchung Cheng,
Lars Eggert, Ilpo Jarvinen and Michael Scharf for comments on
earlier versions of the draft. 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-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 variance, 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 options for which state is unsafe to be kept. This list is meant to avoid work duplication and should be removed upon publication.

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 - might be obsolete, though)
Bubba (DH exchange - might really be obsolete, though)
Trailer CS
SCPS capabilities
S-NACK
Records boundaries
Corruption experienced
SNAP
TCP Compression
Quickstart response
UTO
MPTCP (can we cache when this fails?)
TFO success

Safe but optional to keep state:

MSS
TFO failure (so we don’t try again, since it’s optional)

Safe and necessary to keep state:

TFP cookie (if TFO succeeded in the past)
Appendix C: Automating the Initial Window in TCP over Long Timescales

Note: this section is taken verbatim from [To12], updated to refer to itself as an appendix.

C.1. Introduction

TCP’s congestion control algorithm uses an initial window value (IW), both as a starting point for new connections and after one RTO or more [RFC2581][RFC2861]. 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 typical Internet connections with an maximum transmission units (MTUs) 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 last updated in 1998 experimentally, and moved to the standards track in 2002 [RFC2414][RFC3390]. There have been recent proposals to update the IW based on further increases in host and router capabilities and network capacity, some focusing on specific values (e.g., IW=10), and others prescribing a schedule for increases over time (e.g., IW=6 for 2011, increasing by 1-2 MSS per year).

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 size 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 list of 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 specific, 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

o MulDecr = 0.5

o AddIncr = 2 MSS

o Threshold = 0.05
We assume that the minimum IW (MinIW) should be as currently specified [RFC3390]. The maximum IW can be set to a fixed value [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. 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:
   
   \[ \text{IW} = \text{MaxIW}; \quad \# \text{assume this is in bytes, and an even number of MSS} \]

2. Upon starting a new connection
   
   \[ \text{CWND} = \text{IW}; \]
   \[ \text{conncount}++; \]
   \[ \text{IWnotchecked} = 1; \quad \# \text{true} \]

3. During a connection’s SYN-ACK processing, if SYN-ACK includes ECN, treat as if the IW is too large
   
   \[ \text{if} (\text{IWnotchecked} \&\& (\text{synackecn} == 1)) \{ \]
   \[ \quad \text{losscount}++; \]
   \[ \quad \text{IWnotchecked} = 0; \quad \# \text{never check again} \]
   \[ \} \]

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

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

Touch Expires May 19, 2020 [Page 27]
if (conncount > 1000) {
    if (losscount/conncount > threshold) {
        // the number of connections with errors is too high
        IW = IW * MulDecr;
    } else {
        IW = IW + AddIncr;
    }
}

We recognize that this algorithm can yield a false positive when the sequence number wraps around. This can be avoided using either PAWS [RFC7323] context or 64-bit internal sequence numbers (as in TCP-AO [RFC5925]). Alternately, false positives can be allowed since they are expected to be infrequent and thus will not affect the overall statistics of the algorithm.

The following additional constraints are imposed:

>> The automatic IW algorithm MUST initialize to MaxIW, in the absence of other context information.

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.

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 [RFC2140]). The mechanism is inspired by RFC 2140’s use of information across connections.

The receive window (RWIN) is not involved in this calculation. The size of RWIN 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.