ï»¿

                    TCP Control Block Interdependence

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.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

TCP is a connection-oriented reliable transport protocol layered over IP [RFC0793]. 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 or 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 the Stream Control Transmission Protocol (SCTP) [RFC4960] and the Datagram Congestion Control Protocol (DCCP) [RFC4340], as well as to 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, this document 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
precceded with a "+" may be part of the state maintained as TCP
connection state in the TCB of associated connections 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

ISN: Initial Sequence Number

+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 Path Maximum
       Transmission Unit (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 [RFC0793]

+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]

+ssthresh: 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 [RFC0793]:

    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 (ssthresh)*
            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 (of the given example:
sendMSS, MMS_R, MMS_S, RTT) or are already cached and shared on that
basis (of the given example: 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 shareable. 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:

```
+==============+================================+
| Cached TCB   | New TCB                        |
+==============+================================+
| old_MMS_S    | old_MMS_S or not cached (2)    |
+--------------+--------------------------------+
| old_MMS_R    | old_MMS_R or not cached (2)    |
+--------------+--------------------------------+
| old_sendMSS  | old_sendMSS                    |
+--------------+--------------------------------+
| old_PMTU     | old_PMTU (1)                   |
+--------------+--------------------------------+
| old_RTT      | old_RTT                        |
+--------------+--------------------------------+
| old_RTTVAR   | old_RTTVAR                     |
+--------------+--------------------------------+
| old_option   | (option specific)              |
+--------------+--------------------------------+
| old_ssthresh | old_ssthresh                   |
+--------------+--------------------------------+
| old_sendcwnd | old_sendcwnd                   |
+--------------+--------------------------------+
```

            Table 1: Temporal Sharing - TCB Initialization

(1) Note that PMTU is cached at the IP layer [RFC1191] [RFC4821].

(2) 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).

Table 2 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.

```
+==================+==================+
| Cached           | New              |
+==================+==================+
| old_TFO_cookie   | old_TFO_cookie   |
+------------------+------------------+
| old_TFO_failure  | old_TFO_failure  |
+------------------+------------------+
```

            Table 2: Temporal Sharing -
             Option Info Initialization

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 in Table 3.

```
+==============+===============+=============+==================+
| 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 (1) / | curr_PMTU        |
|              |               | PLPMTUD (1) |                  |
+--------------+---------------+-------------+------------------+
```

```
| 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) |
+--------------+---------------+------------+-----------------+
```

              Table 3: Temporal Sharing - Cache Updates

(1) 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 (new = (1-alpha)*old + alpha *new, where alpha is
in the range [0..1]).

Table 4 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.

```
+=================+=================+=======+=================+
| 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 |
+-----------------+-----------------+-------+-----------------+
```

           Table 4: Temporal Sharing - Option Info Updates

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; the
   MSS of all other segments are constrained by the value updated as
   included in the SYN.

   RTT values are updated by formulae that merge 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. For example, 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 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 state related to other TCP options can be shared
depends on the details of that option. For example, 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.

TFOinfo is cached when a connection is established.

State related to other TCP options might not be as readily cached.
For example, 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:

Cached TCB	New TCB
==================	==========================
old_MMS_S	old_MMS_S
old_MMS_R	old_MMS_R
old_sendMSS	old_sendMSS
old_PMTU	old_PMTU (1)
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)       |
            +-------------------+-------------------------+
```

            Table 5: Ensemble Sharing - TCB Initialization

(1) Note that PMTU is cached at the IP layer [RFC1191] [RFC4821].

In Table 5, 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".

Table 6 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.

```
            +=================+=================+
            | Cached          | New             |
            +=================+=================+
            | old_TFO_cookie  | old_TFO_cookie  |
            +-----------------+-----------------+
            | old_TFO_failure | old_TFO_failure |
            +-----------------+-----------------+
```

                    Table 6: Ensemble Sharing -
                    Option Info Initialization

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:

```
    +==============+===============+===========+=================+
    | 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+ /  | curr_PMTU       |
    |              |               | PLPMTUD+  |                 |
    +--------------+---------------+-----------+-----------------+
    | 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)       |
    +--------------+---------------+-----------+-----------------+
```

              Table 7: Ensemble Sharing - Cache Updates

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

In Table 7, rtt_update() is the function used to combine old and
current values, e.g., as a windowed average or exponentially decayed
average.

Table 8 gives an overview of option-specific information that can be

similarly shared.

```
+=================+=================+=======+=================+
| 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 |
+-----------------+-----------------+-------+-----------------+
```

           Table 8: Ensemble Sharing - Option Info Updates

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
   Table 7 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 4 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. 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 will assume
   initial windows of 10 segments [RFC6928] and the existing
   connection's window will not decrease to accommodate this additional
   load. As a consequence, the three 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 1 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.

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 Link Aggregation Group (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 Encapsulating Security Payload
(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; for example, 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 (e.g., rwnd), TCP option state (e.g., TFOinfo), or for
information that is already cached per-host (e.g., 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 the 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. For example, it may be appropriate to consider the
impact of a connection being in Fast Recovery [RFC5681] or some other
similar unusual feedback state that could inhibit or affect 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
Translation (NAT) [RFC2663], Network Address and Port Translation
(NAPT) [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 re-establishment
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) and update 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 its typical placement solely within the transport layer (in the ISO model) to the network layer. This acknowledges that some components of state are, 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 to 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).

Table 9 describes the current implementation status for TCB temporal
sharing in Windows as of December 2020, Apple variants (macOS, iOS,
iPadOS, tvOS, and watchOS) as of January 2021, Linux kernel version
5.10.3, and FreeBSD 12. Ensemble sharing is not yet implemented.

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

Table 9: KNOWN IMPLEMENTATION STATUS

* Note: In FreeBSD, new ssthresh is the mean of curr_ssthresh and
   its previous value if a previous value exists; in Linux, the
   calculation depends on state and is max(curr_cwnd/2, old_ssthresh)
   in most cases.

In Table 9, "Apple" refers to all Apple OSes, i.e., macOS (desktop/
laptop), iOS (phone), iPadOS (tablet), tvOS (video player), and
watchOS (smart watch), which all share the same Internet protocol
stack.

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 with
   greater specificity. It separates the way MSS is used in both send
   and receive directions, it separates the way both of these MSS values
   differ from sendMSS, it adds both path MTU and ssthresh, and it
   addresses the impact on state associated with TCP options.

   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 (which describes the history to TCB sharing)
   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 any set of connections in which
the TCB is shared, between connections in a single host, or between
hosts if TCB sharing is implemented within a subnet (see
"Implications" (Section 9)). 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. For example, if an application changes its TCB to have a
false and small window size, subsequent connections will experience
performance degradation until their window grows 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

This document has no IANA actions.

14. References

14.1. Normative References

   [RFC0793] Postel, J., "Transmission Control Protocol", STD 7,
              RFC 793, DOI 10.17487/RFC0793, September 1981,
              <https://www.rfc-editor.org/info/rfc793>.

   [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>.

   [RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191,
              DOI 10.17487/RFC1191, November 1990,
              <https://www.rfc-editor.org/info/rfc1191>.

   [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>.

   [RFC4821] 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>.

   [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>.

   [RFC7413] 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>.

   [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>.

   [RFC8201] 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>.

14.2. Informative References

   [Al10]   Allman, M., "Initial Congestion Window Specification",
              Work in Progress, Internet-Draft, draft-allman-tcpm-bump-
              initcwnd-00, 15 November 2010,
              <https://datatracker.ietf.org/doc/html/draft-allman-tcpm-
              bump-initcwnd-00>.

   [Ba12]   Barik, R., Welzl, M., Ferlin, S., and O. Alay, "LISA: A
              linked slow-start algorithm for MPTCP", IEEE ICC,
              DOI 10.1109/ICC.2016.7510786, May 2016,
              <https://doi.org/10.1109/ICC.2016.7510786>.

   [Ba20]   Bagnulo, M. and B. Briscoe, "ECN++: Adding Explicit
              Congestion Notification (ECN) to TCP Control Packets",
              Work in Progress, Internet-Draft, draft-ietf-tcpm-
              generalized-ecn-07, 16 February 2021,
              <https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-
              generalized-ecn-07>.

   [Be94]   Berners-Lee, T., Cailliau, C., Luotonen, A., Nielsen, H.,
              and A. Secret, "The World-Wide Web", Communications of the
              ACM V37, pp. 76-82, DOI 10.1145/179606.179671, August
              1994, <https://doi.org/10.1145/179606.179671>.

   [Br02]   Brownlee, N. and KC. Claffy, "Understanding Internet
              traffic streams: dragonflies and tortoises", IEEE
              Communications Magazine, pp. 110-117,
              DOI 10.1109/MCOM.2002.1039865, 2002,
              <https://doi.org/10.1109/MCOM.2002.1039865>.

   [Br94]   Braden, B., "T/TCP -- Transaction TCP: Source Changes for
              Sun OS 4.1.3", USC/ISI Release 1.0, September 1994.

   [Co91]   Comer, D. and D. Stevens, "Internetworking with TCP/IP",
              ISBN 10: 0134685059, ISBN 13: 9780134685052, 1991.

   [Du16]   Dukkipati, N., Cheng, Y., and A. Vahdat, "Research
              Impacting the Practice of Congestion Control", Computer
              Communication Review, The ACM SIGCOMM newsletter, July

                2016.

[FreeBSD]   FreeBSD, "The FreeBSD Project",
               <https://www.freebsd.org/>.

[Hu01]   Hughes, A., Touch, J., and J. Heidemann, "Issues in TCP
               Slow-Start Restart After Idle", Work in Progress,
               Internet-Draft, draft-hughes-restart-00, December 2001,
               <https://datatracker.ietf.org/doc/html/draft-hughes-
               restart-00>.

[Hu12]   Hurtig, P. and A. Brunstrom, "Enhanced metric caching for
               short TCP flows", IEEE International Conference on
               Communications, DOI 10.1109/ICC.2012.6364516, 2012,
               <https://doi.org/10.1109/ICC.2012.6364516>.

[IANA]   IANA, "Transmission Control Protocol (TCP) Parameters",
               <https://www.iana.org/assignments/tcp-parameters>.

[Is18]   Islam, S., Welzl, M., Hiorth, K., Hayes, D., Armitage, G.,
               and S. Gjessing, "ctrlTCP: Reducing latency through
               coupled, heterogeneous multi-flow TCP congestion control",
               IEEE INFOCOM 2018 - IEEE Conference on Computer
               Communications Workshops (INFOCOM WKSHPS),
               DOI 10.1109/INFCOMW.2018.8406887, April 2018,
               <https://doi.org/10.1109/INFCOMW.2018.8406887>.

[Ja88]   Jacobson, V. and M. Karels, "Congestion Avoidance and
               Control", SIGCOMM Symposium proceedings on Communications
               architectures and protocols, November 1988.

[RFC1379]   Braden, R., "Extending TCP for Transactions -- Concepts",
               RFC 1379, DOI 10.17487/RFC1379, November 1992,
               <https://www.rfc-editor.org/info/rfc1379>.

[RFC1644]   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>.

[RFC2001]   Stevens, W., "TCP Slow Start, Congestion Avoidance, Fast
               Retransmit, and Fast Recovery Algorithms", RFC 2001,
               DOI 10.17487/RFC2001, January 1997,
               <https://www.rfc-editor.org/info/rfc2001>.

[RFC2140]   Touch, J., "TCP Control Block Interdependence", RFC 2140,
               DOI 10.17487/RFC2140, April 1997,
               <https://www.rfc-editor.org/info/rfc2140>.

[RFC2414]   Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's
               Initial Window", RFC 2414, DOI 10.17487/RFC2414, September
               1998, <https://www.rfc-editor.org/info/rfc2414>.

[RFC2663]   Srisuresh, P. and M. Holdrege, "IP Network Address
               Translator (NAT) Terminology and Considerations",
               RFC 2663, DOI 10.17487/RFC2663, August 1999,
               <https://www.rfc-editor.org/info/rfc2663>.

[RFC3124]   Balakrishnan, H. and S. Seshan, "The Congestion Manager",
               RFC 3124, DOI 10.17487/RFC3124, June 2001,
               <https://www.rfc-editor.org/info/rfc3124>.

[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>.

[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>.

[RFC4960]   Stewart, R., Ed., "Stream Control Transmission Protocol",

                  RFC 4960, DOI 10.17487/RFC4960, September 2007,
                  <https://www.rfc-editor.org/info/rfc4960>.

   [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>.

   [RFC6437]   Amante, S., Carpenter, B., Jiang, S., and J. Rajahalme,
               "IPv6 Flow Label Specification", RFC 6437,
               DOI 10.17487/RFC6437, November 2011,
               <https://www.rfc-editor.org/info/rfc6437>.

   [RFC6691]   Borman, D., "TCP Options and Maximum Segment Size (MSS)",
               RFC 6691, DOI 10.17487/RFC6691, July 2012,
               <https://www.rfc-editor.org/info/rfc6691>.

   [RFC6928]   Chu, J., Dukkipati, 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>.

   [RFC7231]   Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
               Protocol (HTTP/1.1): Semantics and Content", RFC 7231,
               DOI 10.17487/RFC7231, June 2014,
               <https://www.rfc-editor.org/info/rfc7231>.

   [RFC7323]   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>.

   [RFC7424]   Krishnan, R., Yong, L., Ghanwani, A., So, N., and B.
               Khasnabish, "Mechanisms for Optimizing Link Aggregation
               Group (LAG) and Equal-Cost Multipath (ECMP) Component Link
               Utilization in Networks", RFC 7424, DOI 10.17487/RFC7424,
               January 2015, <https://www.rfc-editor.org/info/rfc7424>.

   [RFC7540]   Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext
               Transfer Protocol Version 2 (HTTP/2)", RFC 7540,
               DOI 10.17487/RFC7540, May 2015,
               <https://www.rfc-editor.org/info/rfc7540>.

   [RFC7661]   Fairhurst, G., Sathiaseelan, 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>.

   [RFC8684]   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>.

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
Sun OS 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, Section 3.1 of [RFC1644] 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

    Partial Order Connection Permitted

    Partial Order Service Profile

    CC

    CC.NEW

    CC.ECHO

    TCP Alternate Checksum Request

    TCP Alternate Checksum Data

No state to keep:

    End of Option List (EOL)

    No-Operation (NOP)

    Window Scale (WS)

    SACK

    Timestamps (TS)

    MD5 Signature Option

    TCP Authentication Option (TCP-AO)

    RFC3692-style Experiment 1

    RFC3692-style Experiment 2

Unsafe to keep state:

    Skeeter (DH exchange, known to be vulnerable)

    Bubba (DH exchange, known to be vulnerable)

    Trailer Checksum Option

    SCPS capabilities

    Selective Negative Acknowledgements (S-NACK)

    Records Boundaries

    Corruption experienced

SNAP

TCP Compression Filter

Quick-Start Response

User Timeout Option (UTO)

Multipath TCP (MPTCP) negotiation success (see below for
negotiation failure)

TCP Fast Open (TFO) negotiation success (see below for negotiation
failure)

Safe but optional to keep state:

Multipath TCP (MPTCP) negotiation failure (to avoid negotiation
retries)

Maximum Segment Size (MSS)

TCP Fast Open (TFO) negotiation failure (to avoid negotiation
retries)

Safe and necessary to keep state:

TCP Fast Open (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 aging 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; it was originally 1 maximum segment size (MSS)
and increased to the lesser of 4 MSSs or 4,380 bytes [RFC3390]
[RFC5681]. For a typical Internet connection with a maximum
transmission unit (MTU) of 1500 bytes, this permits 3 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 wide-scale deployment, 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:

* 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.

* Adapt to network feedback over long timescales, avoiding values
   that persistently cause network problems.

* Decrease the IW in the presence of sustained loss of IW segments,
   as determined over a number of different connections.

* Increase the IW in the absence of sustained loss of IW segments,
   as determined over a number of different connections.

* 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:

* MinIW = 3 MSS or 4,380 bytes (as per [RFC3390])

* MaxIW = 10 MSS (as per [RFC6928])

* MulDecr = 0.5

* AddIncr = 2 MSS

* Threshold = 0.05

We assume that the minimum IW (MinIW) should be as currently
specified as standard [RFC3390]. The maximum IW (MaxIW) 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 Additive Increase Multiplicative Decrease
(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 Section 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 re-sent 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
Protection Against Wrapped Sequences (PAWS) [RFC7323] context or
internal extended sequence number representations (as in TCP
Authentication Option (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 1 MSS when
    indicated during an evaluation interval.

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

*   An implementation MUST increase by no more than 2 MSSs 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 MSSs.

    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 that 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
in this appendix also benefits from persistent state kept across
reboots, which would also be useful to 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. Also, rwnd is not involved with
congestion control, which is the focus of the way this appendix
manages 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 MSSs 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.

Acknowledgments

Authors' Addresses

Joe Touch
Manhattan Beach, CA 90266
United States of America

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

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