TCPM WG
Internet Draft
Intended status: Informational
Obsoletes: 2140
Expires: July 2019

J. Touch
Independent Consultant
M. Welzl
S. Islam
University of Oslo
January 4, 2019

# TCP Control Block Interdependence
draft-touch-tcpm-2140bis-06.txt

## Status of this Memo

Abstract

This memo updates and replaces RFC 2140's description of interdependent TCP control blocks, in which part of the TCP state is 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

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 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. 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", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying significance described in RFC 2119.

In this document, the characters ">>" preceding an indented line(s) indicates a statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the portions of this RFC covered by these keywords.

## 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]:

```
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

| Cached TCB | New TCB |
|---|---|
| old_MMS_S | old_MMS_S or not cached |
| old_MMS_R | old_MMS_R or not cached |
| old_sendMSS | old_sendMSS |
| old_PMTU | old_PMTU |
| old_RTT | old_RTT |
| old_RTTvar | old_RTTvar |
| old_option | (option specific) |
| old_ssthresh | old_ssthresh |
| old_snd_cwnd | old_snd_cwnd |

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.

TEMPORAL SHARING - Option info

| Cached | New |
|---|---|
| old_TFO_Cookie | old_TFO_Cookie |
| old_TFO_Failure | old_TFO_Failure |

TEMPORAL SHARING - Cache Updates

| Cached TCB | Current TCB | when? | New Cached TCB |
|---|---|---|---|
| old_MMS_S | curr_ MMS_S | OPEN | curr MMS_S |
| old_MMS_R | curr_ MMS_R | OPEN | curr_MMS_R |
| old_sendMSS | curr_sendMSS | MSSopt | curr_sendMSS |
| old_PMTU | curr_PMTU | PMTUD | curr_PMTU |
| old_RTT | curr_RTT | CLOSE | merge(curr,old) |
| old_RTTvar | curr_RTTvar | CLOSE | merge(curr,old) |
| old_option | curr option | ESTAB | (depends on option) |
| old_ssthresh | curr_ssthresh | CLOSE | merge(curr,old) |
| old_snd_cwnd | curr_snd_cwnd | CLOSE | merge(curr,old) |

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 Fraqmentation 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 a more complicated mechanism [RFC1644][Ja86]. 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 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.

TEMPORAL SHARING - Option info Updates

| Cached | Current | when? | New Cached |
|---|---|---|---|
| old_TFO_Cookie | old_TFO_Cookie | ESTAB | old_TFO_Cookie |
| old_TFO_Failure | old_TFO_Failure | ESTAB | old_TFO_Failure |

## 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).

ENSEMBLE SHARING - TCB Initialization

| Cached TCB | New TCB |
|---|---|
| old_MMS_S | old_MMS_S |
| old_MMS_R | old_MMS_R |
| old_sendMSS | old_sendMSS |
| old_PMTU | old_PMTU |
| old_RTT | old_RTT |
| old_RTTvar | old_RTTvar |
| old ssthresh sum | f(old ssthresh sum, N) |
| old snd_cwnd sum | f(old snd cwnd sum, N) |
| old_option | (option-specific) |

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.

ENSEMBLE SHARING Option info

| Cached | New |
|---|---|
| old_TFO_Cookie | old_TFO_Cookie |
| old_TFO_Failure | old_TFO_Failure |

ENSEMBLE SHARING - Cache Updates

| Cached TCB | Current TCB | when? | New Cached TCB |
|---|---|---|---|
| old_MMS_S | curr_MMS_S | OPEN | curr_MMS_S |
| old_MMS_R | curr_MMS_R | OPEN | curr_MMS_R |
| old_sendMSS | curr_sendMSS | MSSopt | curr_sendMSS |
| old_PMTU | curr_PMTU | PMTUD /PLPMTUD | curr_PMTU |
| old_RTT | curr_RTT | update | rtt_update(old,curr) |
| old_RTTvar | curr_RTTvar | update | rtt_update(old,curr) |
| old ssthresh | curr ssthresh | update | adjust sum as appopriate |
| old snd_cwnd | curr snd_cwnd | update | adjust sum as appopriate |
| old_option | curr option | (depends) | (option specific) |

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 [Ja86].

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.

ENSEMBLE SHARING - Option info Updates

| Cached | Current | when? | New Cached |
|---|---|---|---|
| old_TFO_Cookie | old_TFO_Cookie | ESTAB | old_TFO_Cookie |
| old_TFO_Failure | old_TFO_Failure | ESTAB | old_TFO_Failure |

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] and T/TCP hinted that it should be initialized to the old window size [RFC1644]. In the former cases, the assumption is that new connections should behave as conservatively as possible. In the latter T/TCP case, no accommodation is made for concurrent aggregate behavior. The algorithm described in [Ba12] adjusts the initial cwnd depending on the cwnd values of ongoing connections.

## 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 [RFC5861] 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 always take the same path. Because ECMP and LAG examine TCP port numbers, they may not be supported when TCP segments are encapsulated, encrypted, or altered - for example, some Virtual Private Networks (VPNs) are known to use proprietary UDP encapsulation methods. Similarly, they cannot operate 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 [Ja86]. Our observations are for sharing state among TCP connections, which avoids some of the difficulties in an IP-layer solution. One such problem is determining the associated prior outgoing packet for an incoming packet, to infer RTT from the exchange. Because RTTs are

still determined inside the TCP layer, this is simpler 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. A discussion of sharing RTT information among protocols layered over IP, including UDP and TCP, occurred in [Ja86]. Although now deprecated, T/TCP 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.

| TCB data | Status |
|---|---|
| old MMS_S | Not shared |
| old MMS_R | Not shared |
| old_sendMSS | Cached and shared in Linux (MSS) |
| old PMTU | Cached and shared in FreeBSD and Windows (PMTU) |
| old_RTT | Cached and shared in FreeBSD and Linux |
| old_RTTvar | Cached and shared in FreeBSD |
| old TFOinfo | Cached and shared in Linux and Windows |
| old_snd_cwnd | Not shared |
| old_ssthresh | Cached and shared in FreeBSD and Linux:<br>FreeBSD: arithmetic<br>mean of ssthresh and previous value if<br>a previous value exists;<br>Linux: depending on state,<br>max(cwnd/2, ssthresh) in most cases |

## 11. Updates to RFC 2140

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

The detailed impact on TCB state addresses TCB parameters in greater detail, addressing 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 an appendix discussion on history, partly to reflect the deprecation of that protocol.

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

[RFC793] Postel, Jon, "Transmission Control Protocol," Network Working Group RFC-793/STD-7, ISI, Sept. 1981.

[RFC1191] Mogul, J., Deering, S., "Path MTU Discovery," RFC 1191, Nov. 1990.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2140] Touch, J., "TCP Control Block Interdependence", RFC 2140, April 1997.

[RFC4821] Mathis, M., Heffner, J., "Packetization Layer Path MTU Discovery," RFC 4821, Mar. 2007.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., Jain, A., "TCP Fast Open", RFC 7413, Dec. 2014.

[RFC8201] McCann, J., Deering. S., Mogul, J., Hinden, R. (Ed.), "Path MTU Discovery for IP version 6," RFC 8201, Jul. 2017.

14.2. Informative References

[Br02] Brownlee, N. and K. Claffy, "Understanding Internet Traffic Streams: Dragonflies and Tortoises", IEEE Communications Magazine p110-117, 2002.

[Be94] Berners-Lee, T., et al., "The World-Wide Web," Communications of the ACM, V37, Aug. 1994, pp. 76-82.

[Br94] Braden, B., "T/TCP -- Transaction TCP: Source Changes for Sun OS 4.1.3,", Release 1.0, USC/ISI, September 14, 1994.

[Co91] Comer, D., Stevens, D., Internetworking with TCP/IP, V2, Prentice-Hall, NJ, 1991.

[FreeBSD] FreeBSD source code, Release 2.10, http://www.freebsd.org/

[Ja86] Jacobson, V., (mail to public list "tcp-ip", no archive found), 1986.

[Du16] Dukkipati, N., Yuchung C., and Amin V., "Research Impacting the Practice of Congestion Control." ACM SIGCOMM CCR (editorial), on-line post, July, 2016.

[Hu01] Hugues, A., Touch, J., Heidemann, J., "Issues in Slow-Start Restart After Idle", draft-hughes-restart-00 (expired), Dec., 2001.

[Hu12] Hurtig, P., Brunstrom, A., "Enhanced metric caching for short TCP flows," 2012 IEEE International Conference on Communications (ICC), Ottawa, ON, 2012, pp. 1209-1213.

[Ba12] Barik, R., Welzl, M., Ferlin, S., Alay, O., " LISA: A Linked Slow-Start Algorithm for MPTCP", IEEE ICC, Kuala Lumpur, Malaysia, 23-27 May 2016.

[RFC1122] Braden, R. (ed), "Requirements for Internet Hosts -- Communication Layers", RFC-1122, Oct. 1989.

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

[RFC1379] Braden, R., "Transaction TCP -- Concepts," RFC-1379, September 1992.

[RFC2663] Srisuresh, P., Holdrege, M., "IP Network Address Translator (NAT) Terminology and Considerations", RFC-2663, August 1999.

[RFC3390] Allman, M., Floyd, S., Partridge, C., "Increasing TCP's Initial Window," RFC 3390, Oct. 2002.

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

[RFC3124] Balakrishnan, H., Seshan, S., "The Congestion Manager," RFC 3124, June 2001.

[RFC4340] Kohler, E., Handley, M., Floyd, S., "Datagram Congestion Control Protocol (DCCP)," RFC 4340, Mar. 2006.

[RFC4960] Stewart, R., (Ed.), "Stream Control Transmission Protocol," RFC4960, Sept. 2007.

[RFC5861] Allman, M., Paxson, V., Blanton, E., "TCP Congestion Control," RFC 5861, Sept. 2009.

[RFC5925] Touch, J., Mankin, A., Bonica, R., "The TCP Authentication Option," RFC 5925, June 2010.

[RFC6824] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 6824, Jan. 2013.

[RFC6928] Chu, J., Dukkipati, N., Cheng, Y., Mathis, M., "Increasing TCP's Initial Window," RFC 6928, Apr. 2013.

[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 Protocol Version 2 (HTTP/2)", RFC 7540, May 2015.

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

## 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.

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 the appendix

## 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

## 17. 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 [RFC1644].

## 18. Appendix B: Options

In addition to the options that can be cached and shared, this memo also lists all options for which state should *not* be kept. This list is meant to avoid work duplication and should be removed upon publication.

Obsolete (MUST NOT 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

MUST NOT 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

MAY keep state:

MSS

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

MUST keep state:

TFP cookie (if TFO succeeded in the past)