                    TCP Control Block Interdependence

Status of this Memo

   This memo provides information for the Internet community. This memo
   does not specify an Internet standard of any kind. Distribution of
   this memo is unlimited.

Abstract

   This memo makes the case for interdependent TCP control blocks, where
   part of the TCP state is shared among similar concurrent connections,
   or across similar connection instances. TCP state includes a
   combination of parameters, such as connection state, current round-
   trip time estimates, congestion control information, and process
   information. This state is currently maintained on a per-connection
   basis in the TCP control block, but should be shared across
   connections to the same host. The goal is to improve transient
   transport performance, while maintaining backward-compatibility with
   existing implementations.

   This document is a product of the LSAM project at ISI.

Introduction

   TCP is a connection-oriented reliable transport protocol layered over
   IP [9]. 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, the TCB
   is maintained on a per-connection basis. This document discusses the
   implications of that decision, and argues for an alternate
   implementation that shares some of this state across similar
   connection instances and among similar simultaneous connections. The
   resulting implementation can have better transient performance,
   especially for numerous short-lived and simultaneous connections, as
   often used in the World-Wide Web [1]. These changes affect only the
   TCB initialization, and so have no effect on the long-term behavior
   of TCP after a connection has been established.

The TCP Control Block (TCB)

   A TCB is associated with each connection, i.e., with each association
   of a pair of applications across the network. The TCB can be
   summarized as containing [9]:

        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

             micro-state

                 send and receive window state (size*, current number)
                 round-trip time and variance
                 cong. window size*
                 cong. window size threshold*
                 max windows seen*
                 MSS#
                 round-trip time and variance#

   The per-connection information is shown as split into macro-state and
   micro-state, terminology borrowed from [5]. Macro-state describes the
   finite state machine; we include the endpoint numbers and components
   (timers, flags) used to help maintain that state. This includes the
   protocol for establishing and maintaining shared state about the
   connection. 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, RTT), and the
   other is host-pair dependent in its aggregate (*, e.g., cong. window
   info., curr. window sizes).

TCB Interdependence

   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 [8]. T/TCP uses caches to maintain TCB information
   across instances, e.g., smoothed RTT, RTT variance, congestion
   avoidance threshold, and MSS [3]. These values are 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 is 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 current T/TCP implementation saves the MSS and
   aggregates the RTT parameters across multiple connections, but omits
   caching the congestion window information [4], as originally
   specified in [2]. There may be other values that may be cached, such
   as current window size, to permit new connections full access to
   accumulated channel resources.

   We observe that 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. Ensemble sharing occurs when a currently active
   connection to a host is used to initialize another (concurrent)
   connection to that host. T/TCP documents considered the temporal
   case; we consider both.

An Example of Temporal Sharing

   Temporal sharing of cached TCB data has been implemented in the SunOS
   4.1.3 T/TCP extensions [4] and the FreeBSD port of same [7]. As
   mentioned before, only the MSS and RTT parameters are cached, as
   originally specified in [2]. Later discussion of T/TCP suggested
   including congestion control parameters in this cache [3].

   The 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
   are initialized as follows; snd_cwnd reuse is not yet implemented,
   although discussed in the T/TCP concepts [2]:

               TEMPORAL SHARING - TCB Initialization

          Cached TCB   New TCB
          ----
          old-MSS   old-MSS

          old-RTT   old-RTT

          old-RTTvar   old-RTTvar

          old-snd_cwnd   old-snd_cwnd   (not yet impl.)

    Most cached TCB values are updated when a connection closes. An
    exception is MSS, which is updated whenever the MSS option is
    received in a TCP header.

               TEMPORAL SHARING - Cache Updates

     Cached TCB   Current TCB   when?   New Cached TCB
     ----
     old-MSS   curr-MSS   MSSopt   curr-MSS

     old-RTT   curr-RTT   CLOSE   old += (curr - old) >> 2

     old-RTTvar   curr-RTTvar   CLOSE   old += (curr - old) >> 2

     old-snd_cwnd   curr-snd_cwnd   CLOSE   curr-snd_cwnd   (not yet impl.)

    MSS caching is trivial; reported values are cached, and the most
    recent value is used. The cache is updated when the MSS option is
    received, so the cache always has the most recent MSS value from any
    connection. The cache is consulted only at connection establishment,
    and not otherwise updated, which means that MSS options do not affect
    current connections. The default MSS is never saved; only reported
    MSS values update the cache, so an explicit override is required to
    reduce the MSS.

    RTT values are updated by a more complicated mechanism [3], [8].
    Dynamic RTT estimation requires a sequence of RTT measurements, even
    though a single T/TCP transaction may not accumulate enough samples.
    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. Further, the method for averaging the RTT
    values is not the same as the method for computing the RTT values
    within a connection, so that the cached value may not be appropriate.

For temporal sharing, the cache requires updating only when a
connection closes, because the cached values will not yet be used to
initialize a new TCB. For the ensemble sharing, this is not the case,
as discussed below.

Other TCB variables may also be cached between sequential instances,
such as the congestion control window information. Old cache values
can be overwritten with the current TCB estimates, or a MAX or MIN
function can be used to merge the results, depending on the optimism
or pessimism of the reused values. For example, the congestion window
can be reused if there are no concurrent connections.

An Example of Ensemble Sharing

Sharing cached TCB data across concurrent connections requires
attention to the aggregate nature of some of the shared state.
Although MSS and RTT values can be shared by copying, it may not be
appropriate to copy congestion window information. At this point, we
present only the MSS and RTT rules:

              ENSEMBLE SHARING - TCB Initialization

              Cached TCB   New TCB
              ----
              old-MSS   old-MSS

              old-RTT   old-RTT

              old-RTTvar   old-RTTvar

                 ENSEMBLE SHARING - Cache Updates

          Cached TCB   Current TCB   when?   New Cached TCB
          ----
          old-MSS   curr-MSS   MSSopt   curr-MSS

          old-RTT   curr-RTT   update   rtt_update(old,curr)

          old-RTTvar   curr-RTTvar   update   rtt_update(old,curr)

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. An optimistic solution would

be to update cached data as early as possible, rather than only when
a connection is closing. Some T/TCP implementations do this for MSS
when the TCP MSS header option is received [4], although it is not
addressed specifically in the concepts or functional specification
[2][3].

In current T/TCP, RTT values are updated only after a CLOSE, which
does not benefit concurrent sessions. As mentioned in the temporal
case, averaging values between concurrent connections requires
incorporating new RTT measurements. 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 [8].

Congestion window size aggregation is more complicated in the
concurrent case. When there is an ensemble of connections, we need
to decide how that ensemble would have shared the congestion window,
in order to derive initial values for new TCBs. Because concurrent
connections between two hosts share network paths (usually), they
also share whatever capacity exists along that path. With regard to
congestion, the set of connections might behave as if it were
multiplexed prior to TCP, as if all data were part of a single
connection. As a result, the current window sizes would maintain a
constant sum, presuming sufficient offered load. This would go beyond
caching to truly sharing state, as in the RTT case.

We pause to note that any assumption of this sharing can be
incorrect, including this one. In current implementations, new
congestion windows are set at an initial value of one segment, 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, such as in trans-Atlantic
Web access.

There are several ways to initialize the congestion window in a new
TCB among an ensemble of current connections to a host, as shown
below. Current TCP implementations initialize it to one segment [9],
and T/TCP hinted that it should be initialized to the old window size
[3]. In the former, the assumption is that new connections should
behave as conservatively as possible. In the latter, no accommodation
is made to concurrent aggregate behavior.

In either case, the sum of window sizes can increase, rather than
remain constant. Another solution is to give each pending connection

its "fair share" of the available congestion window, and let the
connections balance from there. The assumption we make here is that
new connections are implicit requests for an equal share of available
link bandwidth which should be granted at the expense of current
connections. This may or may not be the appropriate function; we
propose that it be examined further.

                 ENSEMBLE SHARING - TCB Initialization
                 Some Options for Sharing Window-size

     Cached TCB   New TCB
     ----
     old-snd_cwnd   (current)   one segment

                         (T/TCP hint)   old-snd_cwnd

                         (proposed)   old-snd_cwnd/(N+1)
                                        subtract old-snd_cwnd/(N+1)/N
                                        from each concurrent

                 ENSEMBLE SHARING - Cache Updates

     Cached TCB   Current TCB   when?   New Cached TCB
     ----
     old-snd_cwnd curr-snd_cwnd   update (adjust sum as appropriate)

Compatibility Issues

   Current TCP implementations do not use TCB caching, with the
   exception of T/TCP variants [4][7]. New connections use the default
   initial values of all non-instantiated TCB variables. As a result,
   each connection calculates its own RTT measurements, MSS value, and
   congestion information. Eventually these values are updated for each
   connection.

   For the congestion and current window information, the initial values
   may not be consistent with the long-term aggregate behavior of a set
   of concurrent connections. If a single connection has a window of 4
   segments, new connections assume initial windows of 1 segment (the
   minimum), although the current connection's window doesn't decrease
   to accommodate this additional load. As a result, connections can
   mutually interfere. One example of this has been seen on trans-
   Atlantic links, where concurrent connections supporting Web traffic
   can collide because their initial windows are too large, even when
   set at one segment.

Because this proposal attempts to anticipate the aggregate steady-
state values of TCB state among a group or over time, it should avoid
the transient effects of new connections. In addition, because it
considers the ensemble and temporal properties of those aggregates,
it should also prevent the transients of short-lived or multiple
concurrent connections from adversely affecting the overall network
performance. We are performing analysis and experiments to validate
these assumptions.

Performance Considerations

Here we attempt to optimize transient behavior of TCP without
modifying its long-term properties. The predominant expense is in
maintaining the cached values, or in using per-host state rather than
per-connection state. In cases where performance is affected,
however, we note that the per-host information can be kept in per-
connection copies (as done now), because with higher performance
should come less interference between concurrent connections.

Sharing TCB state can occur only at connection establishment and
close (to update the cache), to minimize overhead, optimize transient
behavior, and minimize the effect on the steady-state. It is possible
that sharing state during a connection, as in the RTT or window-size
variables, may be of benefit, provided its implementation cost is not
high.

Implications

There are several implications to incorporating TCB interdependence
in TCP implementations. First, it may prevent the need for
application-layer multiplexing for performance enhancement [6].
Protocols like persistent-HTTP 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 MSS, RTT, and congestion
windows. By avoiding the so-called, "slow-start restart," performance
can be optimized. Our proposal provides the MSS, RTT, and OPEN
handshake avoidance of T/TCP, and the "slow-start restart avoidance"
of multiplexing, without requiring a multiplexing mechanism at the
application layer. This multiplexing will be complicated when
quality-of-service mechanisms (e.g., "integrated services
scheduling") are provided later.

Second, we are attempting to push some of the TCP implementation from
the traditional transport layer (in the ISO model [10]), to the
network layer. This acknowledges that some state currently maintained
as per-connection is in fact per-path, which we simplify as per-
host-pair. Transport protocols typically manage per-application-pair

   associations (per stream), and network protocols manage per-path
   associations (routing). Round-trip time, MSS, and congestion
   information is more appropriately handled in a network-layer fashion,
   aggregated among concurrent connections, and shared across connection
   instances.

   An earlier version of RTT sharing suggested implementing RTT state at
   the IP layer, rather than at the TCP layer [8]. 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 shared at
   the network layer.

   We also note that per-host-pair associations are not the limit of
   these techniques. It is possible that TCBs could be similarly shared
   between hosts on a LAN, because the predominant path can be LAN-LAN,
   rather than host-host.

   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 LAN. One possible
   optimization is to make that implicit feedback explicit, via extended
   information in the per-host TCP area.

Security Considerations

   These suggested implementation enhancements do not have additional
   ramifications for direct attacks. These enhancements 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 0, 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 on the LAN (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.

The solution is to limit the effect of compromised TCB values. TCBs
are compromised when they are modified directly by an application or
transmitted between hosts via unauthenticated means (e.g., by using a
dirty flag). TCBs that are not compromised by application
modification do not have any unique security ramifications. Note that
the proposed parameters for TCB sharing are not currently modifiable
by an application.

All shared TCBs MUST be validated against default minimum parameters
before used for new connections. This validation would not impact
performance, because it occurs only at TCB initialization. This
limits the effect of attacks on new connections, to reducing the
benefit of TCB sharing, resulting in the current default TCP
performance. For ongoing connections, the effect of incoming packets
on shared information should be both limited and validated against
constraints before use. This is a beneficial precaution for existing
TCP implementations as well.

TCBs modified by an application SHOULD not be shared, unless the new
connection sharing the compromised information has been given
explicit permission to use such information by the connection API. No
mechanism for that indication currently exists, but it could be
supported by an augmented API. This sharing restriction SHOULD be
implemented in both the host and the LAN. Sharing on a LAN SHOULD
utilize authentication to prevent undetected tampering of shared TCB
parameters. These restrictions limit the security impact of modified
TCBs both for connection initialization and for ongoing connections.

Finally, shared values MUST be limited to performance factors only.
Other information, such as TCP sequence numbers, when shared, are
already known to compromise security.

Acknowledgements

The author would like to thank the members of the High-Performance
Computing and Communications Division at ISI, notably Bill Manning,
Bob Braden, Jon Postel, Ted Faber, and Cliff Neuman for their
assistance in the development of this memo.

References

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

   [2] Braden, R., "Transaction TCP -- Concepts," RFC-1379,
       USC/Information Sciences Institute, September 1992.

   [3] Braden, R., "T/TCP -- TCP Extensions for Transactions Functional
       Specification," RFC-1644, USC/Information Sciences Institute,
       July 1994.

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

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

   [6] Fielding, R., et al., "Hypertext Transfer Protocol -- HTTP/1.1,"
       Work in Progress.

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

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

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

   [10] Tannenbaum, A., Computer Networks, Prentice-Hall, NJ, 1988.

Author's Address

   Joe Touch
   University of Southern California/Information Sciences Institute
   4676 Admiralty Way
   Marina del Rey, CA 90292-6695
   USA
   Phone: +1 310-822-1511 x151
   Fax: +1 310-823-6714
   URL: http://www.isi.edu/~touch
   Email: touch@isi.edu